Faster Algorithms for Edge Connectivity via Random 2-Out Contractions

Mohsen Ghaffari[*]
ETH Zurich
ghaffari@inf.ethz.ch

Krzysztof Nowicki[†]
Univ. of Wroclaw
knowicki@cs.uni.wroc.pl

Mikkel Thorup[‡]
BARC, Univ. of Copenhagen
mikkel2thorup@gmail.com

Abstract

We provide a simple new randomized contraction approach to the global minimum cut problem for simple undirected graphs. The contractions exploit 2-out edge sampling from each vertex rather than the standard uniform edge sampling. We demonstrate the power of our new approach by obtaining better algorithms for sequential, distributed, and parallel models of computation. Our end results include the following randomized algorithms for computing edge connectivity, with high probability[1]:

- Two *sequential* algorithms with complexities $O(m \log n)$ and $O(m+n \log^3 n)$. These improve on a long line of developments including a celebrated $O(m \log^3 n)$ algorithm of Karger [STOC'96] and the state of the art $O(m \log^2 n (\log \log n)^2)$ algorithm of Henzinger et al. [SODA'17]. Moreover, our $O(m + n \log^3 n)$ algorithm is optimal when $m = \Omega(n \log^3 n)$.
- An $\tilde{O}(n^{0.8} D^{0.2} + n^{0.9})$ round *distributed* algorithm, where D denotes the graph diameter. This improves substantially on a recent breakthrough of Daga et al.[STOC'19], which achieved a round complexity of $\tilde{O}(n^{1-1/353} D^{1/353} + n^{1-1/706})$, hence providing the first sublinear distributed algorithm for exactly computing the edge connectivity.
- The first $O(1)$ round algorithm for the *massively parallel computation* setting with linear memory per machine.

[*]Mohsen Ghaffari's research is supported by the Swiss National Foundation, under project number 200021_184735.

[†]Krzysztof Nowicki's research is supported by the National Science Centre, Poland grant 2017/25/B/ST6/02010.

[‡]Mikkel Thorup's research is supported by his Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research and by his Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

[1]We use the phrase *with high probability* (whp) to indicate that a statement holds with probability $1 - O(n^{-\gamma})$, for any desired constant $\gamma \geq 1$.

Contents

1 Introduction and Related Work

Computing the minimum cut is one of the classic graph problems, with a range of applications — e.g., in analyzing the failure robustness of a network or in identifying the communication bottlenecks — and it has been studied extensively since the 1960s [FF56, FF62, GH61]. Of particular interest to the present paper is the case of simple undirected graphs: here, the objective is to identify the smallest set of edges whose removal disconnects the graph. This is often called the *edge connectivity* problem. Our main contribution is to propose a simple randomized contraction process that, when combined with some other ideas, leads to faster algorithms for the edge connectivity problem in a number of computational settings. Next, in Section 1.1, we overview some of the previous algorithmic developments, in Section 1.2 we state our improvements for various computational settings, and in Section 1.3, we provide a brief overview of this randomized contraction process.

1.1 Related Work

Here, we discuss three lines of developments from prior work that are most directly related to our work. Some other results that are relevant as a point of comparison for our algorithms will be mentioned later, when describing our particular results for different computational settings. Moreover, we refer to [KT15, Section 1] for a nice survey of other work on this problem.

(I) In 1993, Karger presented his random contraction idea for minimum cut [Kar93]: contract randomly chosen edges, one by one, until only two nodes remain. The edges in between are the minimum cut with probability $\Omega(1/n^2)$. Thus, via $O(n^2 \log n)$ repetitions, we can identify the minimum cut, with high probability. Thanks to its extreme simplicity and elegance and a range of corollaries, this has by now become a well-known result and a standard ingredient of many textbooks and classes on algorithms. Shortly after, Karger and Stein [KS93] presented a bootstrapped version of this contraction idea, which sets up a recursion that judiciously chooses which instances to re-run, and gives an algorithm for the minimum cut problem with time complexity $O(n^2 \log^3 n)$.

(II) In 1996, Karger [Kar96] provided the first algorithm for minimum cut that has a near-linear complexity in the number of edges; it runs in $O(m \log^3 n)$ time. This algorithm uses a different approach: it performs a certain packing of spanning trees, a la Tutte and Nash-Williams [Tut61, NW61], and then reads the cuts defined by removal of any two edges from a tree, and reports the minimum such cut. This near-linear time algorithm is randomized (and Monte Carlo) and the question of obtaining a deterministic near-linear time algorithm (or even a near-linear time Las Vegas randomized algorithm) remained open for a long time.

(III) In 2015, Kawarabayashi and Thorup [KT15] gave the first such deterministic algorithm, which in $O(m \log^{12} n)$ time computes a minimum cut in simple graphs (i.e., solves the edge connectivity problem). Their key new idea was to exploit that in simple graphs, all non-singleton min-cuts are very sparse, and hence we can contract all edges that are not in sparse cuts. These contractions lead to a graph with $\tilde{O}(m/\delta)$ edges, where δ denotes the minimum degree, while preserving all non-singleton[2] minimum cuts. This sparser (multi-)graph is then solved using older and slower algorithms. To find the sparse cuts, Kawarabayashi and Thorup [KT15] used a deterministic near-linear time diffusion-based algorithm, inspired by page rank [PBMW99]. Later, Henzinger, Rao, and Wang [HRW17] provided a faster deterministic algorithm following a similar framework, but based on computing flows to find sparse cuts. Overall, this algorithm computes the minimum cut in $O(m \log^2 n (\log \log n)^2)$ time. This improved on the $O(m \log^3 n)$ algorithm of Karger [Kar96] and is the state of the art time complexity for edge connectivity. We remark that Karger's algorithm [Kar96] works also for weighted graphs, while those of [KT15], [HRW17], and ours are limited to simple unweighted graphs (i.e., the edge connectivity problem).

[2] A cut in which exactly one node is on one side and all the other nodes are on the other side is called a *singleton* or a *trivial* cut. It is trivial to read the size of all singleton cuts — i.e., node degrees — and identify their minimum.

1.2 Our Results

In the next four subsubsections, we overview the algorithmic improvements that we obtain for computing edge connectivity in various computational settings. Tables 1 and 2 summarize the previous state of the art as well as our algorithms. As a rough summary, we note that our improvement is more substantial quantitatively for settings of distributed and massively parallel computation — which includes a *polynomially improved distributed algorithm* and the *first constant time massively parallel algorithm* with linear local memory as detailed in Table 2. In contrast, the sequential improvements — which includes an *optimal sequential algorithm* for graphs with at least $n \log^3 n$ edges, as detailed in Table 1 — is probably accessible and interesting for a broad range of readers.

1.2.1 Improvements in Sequential Algorithms

Our Contribution: For the sequential setting, our main end-result is summarized below:

Theorem 1.1. *Given a simple input graph G, with n vertices and m edges it is possible to find its minimum cut in $\min\{\mathcal{O}(m + n \log^3 n), \mathcal{O}(m \log n)\}$ time, with high probability. Moreover, we can obtain a cactus representation of all the minimum cuts in $\min\{\mathcal{O}(m+n \log^{O(1)} n), \mathcal{O}(m \log n)\}$ time.*

The proof appears in Theorems 4.2 and 4.6. For minimum cut, the $O(m + n \log^3 n)$ bound ensures that the algorithm has an optimal complexity whenever $m = \Omega(n \log^3 n)$. The $O(m \log n)$ part of the complexity bound is interesting for sparser graphs. It moreover improves on the state of the art $O(m \log^2 n \cdot (\log \log n)^2)$ algorithm [HRW17].

1.2.2 Improvements in Distributed Algorithms

Setting: We use the standard message-passing model of distributed computing (i.e., CONGEST [Pel00]): there is one processor on each graph node, which initially knows only its own edges, and per round each processor can send one $O(\log n)$-bit message to each of its neighbors. At the end, each processor should know its own part of the output, e.g., which of its edges are in the identified cut.

State of the Art: Recently, Daga et al. [DHNS19] gave the first distributed algorithm that computes the exact min-cut in simple graphs in sublinear number of rounds. Their algorithm runs in $\tilde{O}(n^{1-1/353} D^{1/353} + n^{1-1/706})$ rounds, where D denotes the network diameter. In contrast, for graphs with a small edge-connectivity λ, a sublinear-time algorithm was known due to Nanongkai and Su [NS14], with round complexity $\tilde{O}((D + \sqrt{n})\lambda^4)$. They also provide a $1 + \varepsilon$ approximation for any constant $\varepsilon > 0$, which runs in $\tilde{O}(D + \sqrt{n})$ rounds, and was an improvement on a $2 + \varepsilon$ approximation of Ghaffari and Kuhn [MK13] with a similar round complexity.

Our Contribution: We obtain an algorithm that runs in $\tilde{\mathcal{O}}\big(n^{0.8} D^{0.2} + n^{0.9}\big)$ rounds, which provides a considerable improvement on the barely sublinear complexity of Daga et al. [DHNS19]. The overall algorithm is also considerably simpler.

Theorem 1.2. *Given a simple input graph G, with n vertices and m edges it is possible to find its minimum cut in the* CONGEST *model in $\tilde{\mathcal{O}}\big(n^{0.8} D^{0.2} + n^{0.9}\big)$ rounds, with high probability.*

1.2.3 Improvements in Massively Parallel Algorithms

Setting: Parallel algorithms and especially those for modern settings of parallel computation (such as MapReduce [DG04], Hadoop [Whi12], Spark [ZCF+10], and Dryad [IBY+07]) have been receiving increasing attention recently, due to the need for processing large graphs. We work with the Massively Parallel Computation (MPC) model, which was introduced by Karloff et al. [KSV10] and has by now become a standard theoretical model for the study of massively parallel graph algorithms.

	Sequential RAM	Classic Parallel PRAM	
	work	work	depth
simple graphs, previous results	$\mathcal{O}(m \log^2 n (\log \log n)^2)$ [HRW17]	$\mathcal{O}(m \log^4 n)$	$\mathcal{O}(\log^3 n)$[GG18]
simple graphs, ***our results***	$\min\{\mathcal{O}(m \log n), \mathcal{O}(m + n \log^3 n)\}$	$\mathcal{O}(m \log^2 n + n \log^4 n)$	$\mathcal{O}(\log^3 n)$

Table 1: Comparison of results, in the Sequential and PRAM settings. See Sections 1.2.1 and 1.2.4.

In the MPC model, the graph is distributed among a number of machines. Each machine has a limited memory S — known as the *local memory* — and thus can send or receive at most S words, per round. The number of machines M is typically assumed to be just enough to fit all the edges, i.e., $O(m/S)$ or slightly higher. We refer to $M \cdot S$ as the *global memory*. The main measure is the number of rounds needed to solve the problem, given a predetermined limited local memory.

State of the Art: In the super-linear regime of local memory where $S = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$, many graph problems —particularly, including minimum cut [LMSV11] — can be solved in $O(1)$ rounds, using a relatively simple filtering idea. Much of the recent activities in the area has been on achieving similarly fast algorithms for various problems in the much harder memory regimes where S in nearly linear or even sublinear in n [CLM+18, GGK+18, BEG+18, ASS+18, ABB+19, GU19, BBD+19, BFU19, ASW19, GKMS19, CFG+19, BHH19, GKU19]. For minimum cut, in the nearly linear memory regime where $S = \tilde{O}(n)$ regime, the best known algorithm is much slower and runs in $O(\log^3 n)$ rounds [LMSV11].

Our Contribution: We give the first algorithm with $O(1)$ round complexity while using only $\mathcal{O}(n)$ memory per machine and $\mathcal{O}(m + n \log^5 n)$ global memory. This settles the complexity of minimum cut in the nearly-linear memory regime.

Theorem 1.3. *Given a simple input graph G, with n vertices and m edges it is possible to find its minimum cut in $\mathcal{O}(1)$ rounds, with high probability, using $\mathcal{O}(n)$ local memory per machine and $\mathcal{O}(m + n \log^5 n)$ global memory.*

1.2.4 Improvements in PRAM Parallel Algorithms

For the standard PRAM model of parallel algorithms (Concurrent Write Exclusive Read), our algorithm improves the total work while achieving the same depth complexity as the state of the art [GG18]. We get an algorithm with $\mathcal{O}(\log^3 n)$ depth and $\mathcal{O}(m \log^2 n + n \log^4 n)$ work. This improves on the work complexity of the state of the art algorithm of Geissman and Gianinazzi [GG18], which has $\mathcal{O}(\log^3 n)$ depth and $\mathcal{O}(m \log^4 n)$ work.

Theorem 1.4. *There exists a* CREW PRAM *algorithm that for a simple graph with n vertices and m edges computes its minimum cut using $\mathcal{O}(m \log^2 n + n \log^4 n)$ work, with $\mathcal{O}(\log^3 n)$ depth. The algorithm returns a correct answer with high probability.*

1.3 Our Method, In a Nutshell

Our main technical contribution is a simple, and plausibly practical, randomized contraction process that transforms any n-node graph with minimum cut λ to a multi-graph with $O(n/\lambda)$ vertices and only $O(n)$ edges, while preserving all non-singleton[3] minimum cuts with high probability. This can

[3] A cut in which exactly one node is on one side is called a *singleton* cut, or sometimes a *trivial* cut. In most computational settings, it is trivial to read the size of all singleton cuts — that is, degrees of nodes — and identify their minimum. Thus, the problem of computing the edge connectivity effectively boils down to assuming the minimum cut size is at most the minimum degree and identifying the smallest non-singleton cut.

	Modern Parallel MPC			Distributed CONGEST
	local memory	total memory	rounds	rounds
simple graphs, previous results	$\mathcal{O}(n)$ $\mathcal{O}(n^{1+\varepsilon})$	$\mathcal{O}(m)$ $\mathcal{O}(m)$	$\mathcal{O}(\log^3 n)$ [LMSV11] $\mathcal{O}(1)$ [LMSV11]	$\tilde{\mathcal{O}}(n^{1-1/353}D^{1/353} + n^{1-1/706})$ [DHNS19]
simple graphs, ***our results***	$\mathcal{O}(n)$	$\mathcal{O}(m + n\log^5 n)$	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(n^{0.8}D^{0.2} + n^{0.9})$

Table 2: Comparison of results, in the massively parallel computation and distributed computation settings. The previous results for the MPC model show two lines, for different regimes of local memory. See Sections 1.2.2 and 1.2.3.

also be viewed as a simple *graph compression* for (non-trivial) minimum cuts. We then solve the minimum cut problem on this remaining sparse (multi-)graph, using known algorithms.

The aforementioned contraction process itself has two parts, and some careful repetition for success amplification, as we overview next: (A) The main novelty of this paper is the first contraction part, which we refer to as *random 2-out contraction*: for each node v, we randomly choose 2 of its edges (with replacement and independent of other choices) — we view these as "outgoing" edges from node v, proposed by v for contraction — and we contract all chosen edges simultaneously. We prove that this reduces the number of vertices to $O(n/\lambda)$ while preserving any non-trivial minimum cut with a constant probability. In fact, we show the number of vertices is in $O(n/\delta) \leq O(n/\lambda)$, where δ denotes the minimum degree. Furthermore, the contraction preserves any non-singleton cut with size at most $2-\varepsilon$ factor of the minimum cut size, with a constant probability, for any constant $\varepsilon \in (0, 1]$. (B) For the second part, we use some known techniques (e.g., sparse-certificates or a modification of Karger's contraction [Kar93] with an early stopping rule), which transforms the graph after the first part of contractions to have also only $O(n)$ edges—i.e., within an $O(\lambda)$ factor of the number of vertices—while preserving any cut of size $O(\lambda)$. (C) Finally, we use $O(\log n)$ repetitions of the combination of these two parts, and a carefully designed "majority" voting per edge, to amplify the success probability and conclude that with high probability, all non-trivial minimum cuts are preserved, while having $O(n/\delta) \leq O(n/\lambda)$ vertices and $O(n)$ edges.

Related Work on k-Out: We note that random k-out subgraphs have been studied in the literature of random graphs. See for instance the work of Frieze et al. [FJ17] which shows that if $\delta \geq (1/2+\epsilon)n$, then a k-out subgraph (to be precise, sampling k edges per node and without replacement) is whp k-connected, for $k = O(1)$. We leverage a somewhat opposite property of k-out: that with some constant probability, it does not contract any edge for a singleton minimum cut (hence, the subgraph is not even connected), while still significantly reducing the number of *vertices*.

In a very recent paper [HKT+19], Holm et al. have shown that if $k \geq c \log n$, then the k-out contraction of a simple graph with n vertices has only $O(n/k)$ edges (their k-out definition is slightly different, but easily converted to ours). If such a result was true for $k = 2$, then this would simplify our constructions, but proving it for close to constant k seems far out of reach with current techniques. The probability of destroying a min-cut with a k-out sample grows exponentially in k, so the techniques from [HKT+19] are not relevant to our min-cut computation. Conversely, the results presented here have no impact on the targets from [HKT+19]. After 2-out sampling, we do contract edges with highly connected end-points to get down to $O(n)$ edges, but this is only valid because we only care about small cuts. In short, the only relation to [HKT+19] is that both our work and theirs study and show algorithmic benefits of k-out sampling.

Roadmap: In Section 2, we describe our contraction process and state its guarantees. Later, in Section 3, we outline how by using this contraction process and some other algorithmic ideas, we obtain faster min cut algorithms for various computational settings. The details of the implementations in various computational settings appear later, in separate sections.

2 Our Contraction Process

Basic Definitions and Notations: We are working with a simple and we use n to denote its number of vertices, m to denote its number of edges, δ to denote its minimum degree, and λ to denote its edge connectivity, i.e., the smallest number of edges whose removal disconnect the graph. When dealing with different graphs, we may use subscript notation to say which graph we are working with. For a graph H, we let n_H denote the number of nodes, m_H the number of edges, δ_H the minimum degree, and λ_H the edge connectivity of H.

We define a cut in a graph by the set of vertices that are on one side of this cut. If a cut is defined by some set of vertices $S \subset V$, the edges of the cut are the edges of the graph that have exactly one endpoint in S. Furthermore, we denote the set of edges of cut S by $C(S)$, and the size of a cut $C(S)$ by $|C(S)|$. We call the cut $C(S)$ a non-singleton iff $|S| > 1$. We say that $C(S)$ is a minimum cut, if for each cut S', we have $|C(S)| \leq |C(S')|$. For a given value $\alpha \geq 1$, we say that a cut $C(S)$ is an α minimum cut, or α-small, iff $|C(S')| \leq \alpha\lambda$.

For any edge set D from G, we denote by G/D the result of contracting the edges from D in G. In this paper, we identify a cut with the cut edges connecting the two sides. The contraction of D preserve a given cut C if and only if $C \cap D = \emptyset$. The understanding here is that all edges have identifiers, that is, they are not just vertex pairs, so when we do contractions and remove self-loops, it is well-defined which edges survived.

2.1 Contraction Outline

Our main result is captured by the following statement. While stating this result, to make things concrete, we also mention the sequential time related for implementing it. The complexity for other computational settings is discussed in the later sections.

Theorem 2.1. *Let G be a simple graph with m edges, n nodes, and min-degree δ. Fix an arbitrary constant $\varepsilon \in (0, 1]$. In $O(m \log n)$ time, we can randomly contract the graph to a multi-graph $\widehat{G}$ with $O(n/\delta)$ nodes and $O(n)$ edges such that, whp, $\widehat{G}$ preserves all non-trivial $(2-\varepsilon)$-min-cuts of G.*

At the heart of the above result is a contraction captured by Theorem 2.2 which preserves each particular small cut with a *constant* probability. We discuss later in Section 2.4 how we *amplify the success probability* so that we preserve all nearly minimum cuts whp, thus giving the above theorem, without sacrificing the number of nodes or edges.

Theorem 2.2. *Let G be a simple graph with m edges, n nodes, and min-degree δ, and fix an arbitrary constant $\varepsilon \in (0, 1]$. Then, in $O(m)$ time, we can randomly contract the graph down to $O(n/\delta)$ nodes and $O(n)$ edges such that, for any fixed non-trivial $(2-\varepsilon)$-min-cut, we preserve the cut with at least with a constant probability $p_\varepsilon > 0$.*

Outline of the Contraction Process of Theorem 2.2: Our contraction process has two parts. The first part is contracting a random 2-out and, as formally stated in Theorem 2.4, we show that this step reduces the number of vertices to $O(n/\delta)$. The second part, as formally stated in Lemma 2.9, then reduces the number of edges to $O(n)$. For this second part, a different method which achieves the same result but allows more efficient parallel implementations is provided in Lemma 6.2. Each of these processes preserves any particular non-singleton $(2-\varepsilon)$ minimum cut with at least a constant probability. Hence, their composition has a constant probability of that cut. After both contraction processes, we have $\mathcal{O}\left(\frac{n}{\delta}\right) = \mathcal{O}(\frac{n}{\lambda})$ nodes and $\mathcal{O}\left(\frac{n}{\delta} \cdot \lambda\right) = \mathcal{O}(n)$ edges.

We comment that for our distributed algorithm, we actually do not need the part about reducing the number of edges. However, instead, we desire and prove another nice property from the random out contractions: that the summation of the diameters of the 3-out is $\tilde{\mathcal{O}}\left(\frac{n}{\delta}\right)$. In fact, we show in

Section 5 that by choosing a subset of the edges of the 3-out, we can define $\tilde{\mathcal{O}}(\frac{n}{\delta})$ components, each with $O(\log^2 n)$ diameter (clearly, contracting this subset of 3-out preserves any cut, if the full set of that 3-out preserved it).

We next discuss the two parts of the contraction process for reducing the number of vertices and edges, separately, in the next two subsections. In the last subsection of this section, we discuss how we amplify to success to preserve all non singleton $(2-\varepsilon)$ minimum cuts.

2.2 Reducing the number of vertices

Here, we propose and analyze an extremely simple contraction process: each node proposes k randomly sampled incident edges and we contract all proposed edges. More formally, we contract all connected components of a subgraph obtained by a random selection of k incident edges for each vertex of the graph (sampled independently from the original graph, with repetitions). We call this random subgraph a random *k-out subgraph*, and we call the related contraction process a *random k-out contraction*. We show that a random 2-out contraction reduces the number of nodes to $O(n/\delta)$ whp, while preserving any fixed nontrivial nearly minimum cut with a constant probability. Next, we formalize the notion of random k-out contractions, and prove their properties.

2.2.1 Random k-out contractions

Definition 2.3. *Let consider a graph $G = (V, E)$. Let I_j, for $j \in \{1, \ldots, k\}$, be a set of edges obtained by selecting for each node a uniformly random incident edge.*

- *we call a graph $G^{(k)} = (V, \bigcup_{j=1}^{k} I_j)$ a random k-out subgraph of G*
- *a random k-out contraction of a graph G, is a multigraph obtained by contracting connected components of $G^{(k)}$*
- *we say that a node v supplied an edge e to the set I_j, if during the construction of I_j, the random selected edge for v was e,*
- *we use the phrase "cut $C(S)$ is preserved by the random k-out contraction" to indicate that $C(S) \cap \bigcup_{j=1}^{k} I_j = \emptyset$.*

Firstly, we show the following properties of random 2-out contractions.

Theorem 2.4. *A random 2-out contraction of a graph with n vertices and minimum degree δ has $\mathcal{O}(\frac{n}{\delta})$ vertices, with high probability, and preserves any fixed non singleton $(2-\varepsilon)$ minimum cut, for any constant $\varepsilon \in (0, 1]$, with some constant probability at least $p_\varepsilon > 0$.*

Proof Outline. The proof consists of two parts, which are presented in two separate subsubsections. First, in Section 2.2.2, we prove that the number of nodes after contractions is $\mathcal{O}(\frac{n}{\delta})$, whp. Then, in Section 2.2.3, we show that a random 1-out contraction preserves a fixed non-singleton $(2-\varepsilon)$ minimum cut with probability that is at least some positive constant $q_\varepsilon > 0$. This implies that a 2-out contraction (which is simply 2 independent 1-out contractions) has probability of preserving this cut at least $p_\varepsilon = q_\varepsilon^2 > 0$. □

2.2.2 Number of nodes after 2-out contraction

In this part of proving Theorem 2.4, we bound the number of nodes after a 2-out contraction.

Lemma 2.5. *The number of connected components in a random 2-out subgraph of a simple graph with n vertices and the minimum degree δ is $\mathcal{O}(\frac{n}{\delta})$, with high probability.*

Proof. We consider a random process that starts with a graph $G' = (V, \emptyset)$ and gradually adds to G' the edges of a random 2-out subgraph $G^{(2)}$. During this process, each vertex can be in one of three states: *processed*, *active*, or *unprocessed*. The process is performed in phases. Each phase starts in an arbitrary unprocessed vertex v (marked as *active*) and builds a set of vertices reachable from v (which also became *active* as long as they are not *processed*) by the random edges of active vertices. During the process, we maintain a counter κ that is incremented only at the end of a phase that creates a new connected component in $G^{(2)}$, and only if that component is smaller than some threshold value x. The final bound on the number of connected components is the final value of $\kappa + \frac{n}{x}$. Basically, *processed* vertices are the vertices that are in connected components that are taken into account either in the counter κ or they are among at most $\frac{n}{x}$ components of size at least x. Thus, if at the end of a phase we can reach a *processed* vertex from v, we did not create a connected component that is not already included in the bound, and we do not increase the counter κ.

Let us consider following random variables X_i: if phase i ended with increasing κ, $X_i = 1$, otherwise $X_i = 0$. In other words, X_i takes value 1 only if the ith phase ends with creating new connected component that is smaller than x. Furthermore, if at the end of the phase, we can reach a processed vertex from v then $X_i = 0$, therefore

$$P\left(X_i = 1 | X_1 X_2 \dots X_{i-1}\right) \leq P\left(X_i = 1 | X_1 X_2 \dots X_{i-1} \wedge \text{all processed vertices not reachable from } v\right).$$

Bound on the probability of incrementing counter κ: Let A_v denotes the set of active vertices in the current phase that started in v. We start with $A_v = \{v\}$, and then we repeat the following sampling events, one by one. If there is a vertex u from A_v where we have not generated all its sample edges, we pick the first such vertex u added in A_v and generate its next sample edge (u, w) uniformly at random among the at least δ edges incident to u. We say that the sample is "*caught*" if $w \in A_v$. We terminate when there are no more samples to do from A_v and mark all vertices from A_v as *processed*.

If A_v ends up at final size x, we know we have performed exactly $2x$ sampling events, and that the samples were caught exactly $x + 1$ times. The probability that a given sample is caught is at most $(x - 1)/\delta$. The order in which we generate the samples is completely defined in the above process. There are $\binom{2x}{x+1}$ choices for placing the $x + 1$ caught samples among all samples. The probability that we get this particular sequence of caught and not-caught samples is bounded by the probability that the subsequence of $x + 1$ samples that are supposed to get caught actually get caught. This happens with probability at most $((x - 1)/\delta)^{x+1}$. We conclude that the probability that we terminate with $|A_v| = x$ is upperbounded by $P_x = \binom{2x}{x+1}((x - 1)/\delta)^{x+1}$. Then $P_1 = 0$ and $P_2 = 4/\delta^3$. Moreover, for $x \geq 2$, we have

$$P_{x+1}/P_x = \frac{\binom{2x+2}{x+2}(x/\delta)^{x+2}}{\binom{2x}{x+1}((x-1)/\delta)^{x+1}} = \frac{(2x+2)(2x+1)}{(x+2)x}\left(\frac{x}{x-1}\right)^{x+1}\frac{x}{\delta} < 4e^{\frac{x+1}{x-1}}\frac{x}{\delta} \leq \frac{4e^3 x}{\delta}.$$

The last expression is bounded by $1/2$ for $x \leq x^* = \delta/(8e^3)$. Thus the probability that we end up with $|A_v| \leq x^*$ is bounded by

$$P_{\leq x^*} = \sum_{x=2}^{x^*} P_x < 2P_2 \leq 8/\delta^3.$$

Value of κ at the end of the process: We have

$$P\left(X_i = 1 | X_1 \dots X_{i-1}\right) \leq P\left(X_i = 1 | X_1 \dots X_{i-1} \wedge \text{all processed vertices not reachable from } v\right) \leq 8/\delta^3.$$

Hence, variables $X_i|X_1X_2\dots X_{i-1}$ are stochastically dominated by independent random variables Y_i that take value 1 with probability $8/\delta^3$. We can conclude that $P\left(\sum_i X_i \leq \mu\right) < P\left(\sum_i Y_i > \mu\right)$ [Doe18, Lemma 8.7]. By a Chernoff bound, for any $\mu \geq 8n/\delta^3 \geq E\left[\sum_i Y_i\right]$, and for $\varepsilon < 1$ we have

$$P\left(\sum_i X_i > (1+\varepsilon)\mu\right) < P\left(\sum_i Y_i > (1+\varepsilon)\mu\right) \leq \exp\left(-\frac{\mu\varepsilon^2}{3}\right). \tag{1}$$

while for $\varepsilon \geq 1$ we have

$$P\left(\sum_i X_i > (1+\varepsilon)\mu\right) < P\left(\sum_i Y_i > (1+\varepsilon)\mu\right) \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu} \tag{2}$$

For $\delta \leq \sqrt{n}$, by Eq. (1), the value of κ is larger than $\frac{n}{\delta}$, with probability at most $\exp\left(-\Theta(\sqrt{n})\right)$. For $\delta \geq \sqrt{n}$, $n/\delta^3 \leq 1/\sqrt{n}$, hence we can apply Eq. (2) with $\mu = 1/\sqrt{n}$ and $1+\varepsilon = \sqrt{n}$, which gives that the value of κ is larger than some constant c (which is always $\mathcal{O}(\frac{n}{\delta})$) with probability at most $(e/n)^{\frac{c}{2}}$. Therefore, for any $x \leq x^*$ the number of connected components of $G^{(2)}$ smaller than x is $\mathcal{O}(\frac{n}{\delta})$, with high probability. Thus, for $x = x^*$, we get the number of all connected components of $G^{(2)}$ is $\mathcal{O}(\frac{n}{\delta}) + \mathcal{O}(\frac{n}{x^*}) = \mathcal{O}(\frac{n}{\delta})$. □

2.2.3 Preserving a fixed non singleton minimum cut

In this part of the proof of Theorem 2.4, we analyze the probability of preserving a fixed non singleton minimum cut $C(S)$. Before that, we recall a small helper inequality:

Claim 2.6. *For any x and y such that $0 < x \leq y < 1$, we have $1 - x > \left(e^{-\frac{1}{1-y}}\right)^x$*

Proof. This inequality follows from the fact that for any $\alpha > 1$, $(1-\frac{1}{\alpha})^{\alpha-1} > e^{-1}$. If we set $x = \frac{1}{\alpha}$, we have $(1-x)^{\frac{1}{x}-1} > e^{-1}$, which implies $1 - x > e^{-\frac{1}{1-x}x} \geq \left(e^{-\frac{1}{1-y}}\right)^x$. □

Lemma 2.7. *Probability of preserving a fixed non singleton $(2-\varepsilon)$ minimum cut $C(S)$, for any constant $\varepsilon \in (0,1]$, by a random 1-out contraction is at least some constant $q_\varepsilon > 0$.*

Proof. Let us denote by:

- $c(v)$ the number of edges incident to v that are in $C(S)$,
- $d(v)$ the degree of a node v,
- $N(S)$ the set of nodes incident to the edges in $C(S)$.

The probability that a random 1-out contraction does not contract any edge from $C(S)$ is

$$\prod_{v \in N(S)} \left(1 - \frac{c(v)}{d(v)}\right).$$

To analyze this expression, we first argue that for any node $v \in N(S)$, $\frac{c(v)}{d(v)} \leq x < 1$, for some constant x. For that, let us denote by $\alpha = |C(S)|$. Then, the size of a cut defined by the set of vertices $S \setminus \{v\}$ is $\alpha + d(v) - 2c(v)$. On the one hand, we have that $C(S)$ is $(2-\varepsilon)$ minimum cut, hence $|C(S \setminus \{v\})| \leq (2-\varepsilon)\lambda + d(v) - 2c(v)$. On the other hand, $|C(S \setminus \{v\})| \geq \lambda$. Putting those two things together gives that $(1-\varepsilon)\lambda + d(v) > 2c(v)$, which implies that $\frac{c(v)}{d(v)} \leq \frac{(1-\varepsilon)\lambda+1}{2d(v)} \leq 1 - \frac{\varepsilon}{2}$.

Now, we are ready to analyze $\prod_{v\in N(S)}\left(1-\frac{c(v)}{d(v)}\right)$, which is the probability that a random 1-out contraction does not contract any edge from $C(S)$. We know that for each v value of $\frac{c(v)}{d(v)}$ is upper bounded by $1-\varepsilon/2<1$, hence we can use $\frac{c(v)}{d(v)}$ as x and $1-\frac{\varepsilon}{2}$ as y in Claim 2.6. Let $z=e^{-\frac{1}{1-y}}=e^{-\frac{1}{\frac{\varepsilon}{2}}}$. Notice that for any $\varepsilon\in(0,1]$, we have $z\in(0,e^{-2}]\subseteq(0,0.2)$. Then, we have:

$$\prod_{v\in N(S)}\left(1-\frac{c(v)}{d(v)}\right)>z^{\sum_{v\in N(S)}\frac{c(v)}{d(v)}}$$

Furthermore, since a degree of a vertex can not be smaller than λ, we can conclude that the probability that we do not contract any edge from $C(S)$ is at least

$$z^{\sum_{v\in N(S)}\frac{c(v)}{d(v)}}\geq z^{\sum_{v\in N(S)}\frac{c(v)}{\lambda}}=z^{2(2-\varepsilon)\lambda/\lambda}=z^{2(2-\varepsilon)}=e^{-4(2-\varepsilon)/\varepsilon}>0.$$

□

2.3 Reducing the number of edges

In this section, we explain how we reduce the number of edges in the graph resulting after 2-out contractions — which we now know to have $O(n/\lambda)$ vertices — down to $O(n)$ edges, while preserving the minimum cut. The method is based on sparse certificates and it preserves the minimum cut deterministically. A second method, based on Karger's random contraction process but stopped earlier, is presented in Section 6, as it allows for a more efficient implementation in parallel models.

Reducing the number of edges via sparse certificates: We can reduce the number of edges in the contracted graph from Theorem 2.4 using the general k-edge connectivity certificate of Ibaraki and Nagamochi [NI92]. They have shown an algorithm that, given a multigraph M and a number k, in linear time can find a subgraph H with less than kn edges so that if a cut has c edges in M, then the cut between same vertex sets has at least $\min\{k,c\}$ edges in H. Cuts with at most k edges in M are thus fully preserved in H. This way H *is a certificate of k-edge connectivity in M*. Based on this, [KT19] suggested contracting all edges e not in H. Since the end-points of an edge not in H must be k-edge connected in M, and thus $k+1$ connected in H, contracting e preserves any cut of size at most k. Summing up, we get

Lemma 2.8 ([KT19, NI92])**.** *Given a multigraph M with m_M edges and n_M nodes and a number k, in $O(m_M)$ time, we can identify and contract an edge set D such that M/D preserves all cuts of size at most k in M, and such that M/D has at most $n_M k$ edges.*

Given a simple graph G, we can first apply Theorem 2.4 and then Lemma 2.8 with $k=2\delta\geq 2\lambda$, to get the following lemma statement:

Lemma 2.9. *Given a simple graph G with n nodes, m edges, and minimum degree δ, in $O(m)$ time we can perform some randomized contractions yielding a graph $\widehat{G}$ with $O(n/\delta)$ nodes and $O(n)$ edges which with at least a constant probability p_ε preserves any given non-trivial $(2-\varepsilon)$-small cut.*

2.4 Amplifying success and preserving all nontrivial small cuts

We next prove Theorem 2.1, by using Theorem 2.2 and a careful "*repetition and voting*".

Proof of Theorem 2.1. To prove this statement, we build a process that amplifies the success probability of Theorem 2.2 and preserves a particular given non-trivial $(2-\varepsilon)$-small cut C of G with

high probability $1-n^{-\gamma}$. Karger [Kar00] has proved that the number of $(2-\varepsilon)$-small cuts is $O(n^3)$. Hence, by a union bound, we conclude that all non-trivial $(2-\varepsilon)$-small cut of G are preserved with high probability $1-1/n^{\gamma'}$ where $\gamma' = \gamma - 3$ is an arbitrarily large constant.

To build such a contraction with amplified success, we apply Lemma 2.9 for $q = O((\log n)/p_\varepsilon) = O(\log n)$ times, with independent random variables, yielding contracted multigraphs $\widehat{G}_1, \ldots, \widehat{G}_q$. Each $\widehat{G}_i$ preserves our non-trivial $(2-\varepsilon)$-small cut C with probability at least p_ε, so the expected number of $\widehat{G}_i$ that preserve C is at least $\mu = p_\varepsilon q$. Using a standard Chernoff bound (see, e.g., [MR95, Theorem 4.2]), the probability that less than $r = \mu/2 = p_\varepsilon q/2$ of the $\widehat{G}_i$ preserve C is upper bounded by $\exp(-\mu/8) = \exp(-p_\varepsilon q)$. For any given γ, this is $O(n^{-\gamma})$ for $q \geq 8(\ln n)\gamma/p_\varepsilon$.

We now take each edge e in G, and ask how many $\widehat{G}_i$ it is preserved in. Since each $\widehat{G}_i$ has $O(n)$ edges, it can only preserve $O(n)$ edges. Therefore the total number of edge preservation from all the $\widehat{G}_i$ is $q \cdot O(n)$. Therefore, the number of edges that are preserved at least r times is at most $q \cdot O(n)/r = O(\gamma n/p_\varepsilon) = O(n)$. If an edge is not preserved r times, then, by assumption, it is not in any non-trivial $(2-\varepsilon)$-small cut, so we contract it.

Since all but $O(n)$ edges of G got contracted, the resulting graph $\widehat{G}$ has at most $O(n)$ edges. Moreover, our contractions did not contract any edge from any non-trivial $(2-\varepsilon)$-min-cuts, whp. Finally, note that the original graph G had min-degree δ. As proved in [KT19], if a cut of G has size less than δ, it must have at least δ vertices on either side. Therefore, if a node in $\widehat{G}$ has degree below δ, then it must be contracted from at least δ vertices in G, so we have at most n/δ nodes in $\widehat{G}$ with degree below δ. On the other hand, we can have at most $2m_{\widehat{G}}/\delta = O(n/\delta)$ nodes in $\widehat{G}$ with degree at least δ. Hence, we conclude that the total number of nodes in $\widehat{G}$ is $O(n/\delta)$.

Overall, we spend $O(mq) = O(m \log n)$ time, both on generating the q contracted multigraphs $\widehat{G}_i$ and on counting for each edge in G how many $\widehat{G}_i$ it is preserved in (since surviving edges preserve their id doing contractions, we can with each edge id record which $\widehat{G}_i$ it is preserved in). □

3 General Algorithm Outline and Overview of Applications

We now overview the applications of Theorem 2.1 to various models of computing. On the very top level, all the algorithms we present fit the following description:

1. Compute the contraction $\widehat{G}$ of the input graph G as indicated by Theorem 2.1.
2. Compute the minimum cut of $\widehat{G}$ using an algorithm that works for multi-graphs.
3. If the computed cut of $\widehat{G}$ is smaller than minimum degree of G, output it as a minimum cut of G. Otherwise, output the minimum degree of G (and the corresponding vertex) as a (singleton) minimum cut.

Next, we give a brief description of our algorithms for different models. More detailed versions of these algorithms, as well as the formal definitions of the PRAM, MPC and CONGEST models, follow in the subsequent sections.

3.1 Sequential model

In Section 4, we give two sequential algorithms for computing edge connectivity with high probability, both of which follow the above outline. The first algorithm has a complexity of $O(m \log n)$ and follows rather directly from combining Theorem 2.1 with running the minimum cut algorithm of Gabow [Gab91] on the contracted graph $\widehat{G}$. The detailed description is presented Section 4.1.

The second algorithm has a complexity of $O(m + n \log^3 n)$. For this algorithm, in Section 4.2, we present a more elaborate way of implementing a contraction similar to the one provided by

Theorem 2.1 but in just $O(m+n \log n\ \alpha(n,n))$ time, where α denotes the inverse Ackerman function. This process is based on a careful usage of the union-find data structure, and some other probabilistic ideas. Then, we solve the minimum cut problem on the resulting multi-graph, which has $O(n)$ edges, using Karger's algorithm [Kar00] in $O(n \log^3 n)$ time, for a total complexity of $O(m+n \log n\ \alpha(n,n)+ n \log^3 n) = O(m + n \log^3 n)$. We note that any improvement on Karger's algorithm for multi-graphs would immediately lead to an improvement in our algorithm.

3.2 CONGEST model

In Section 5, we provide a distributed algorithm that solves the edge connectivity problem in $\tilde{O}(n^{0.8}D^{0.2} + n^{0.9})$ rounds (as stated in Theorem 1.2). This improves substantially on a recent breakthrough of Daga et al. [DHNS19] that achieved the first sublinear round complexity, running in $\tilde{O}(n^{1-1/353}D^{1/353} + n^{1-1/706})$ rounds. Furthermore, the new algorithm is considerably simpler. This result is presented . We next review the setup and the outline how this improvement is achieved.

Model Description: The CONGEST model is a synchronous message passing model for networked computation. The communication network is abstracted as a graph $G = (V, E)$, with n vertices. There is one processor on each node of the network, each with unique identifier from $\{1, \ldots, \text{poly}(n)\}$. Initially, the network topology is not known to the nodes, except for some global parameters such as as constant factor upper bound on the number n of nodes. The computation is performed in synchronous rounds, each round consists of the phase of (possibly unlimited) local computation and the phase of communication. In the communication phase, each processor is allowed to send a message of size $\mathcal{O}(\log n)$ to each of its neighbors.

For the Minimum Cut problem, the question is to identify the edges with the smallest cardinality whose removal disconnects the network. The output will be in a distributed format, meaning that each processor/node should know its own edges in the identified cut.

The algorithm of Daga et al.: The algorithm of Daga et al. [DHNS19] is actually a mixture of a new algorithm designed for graphs with large edge connectivity, run along an older algorithm of Nanongkai and Su [NS14], which is well-suited for graphs with small edge connectivity and runs in $\tilde{O}((D+\sqrt{n})\lambda^4)$. The new algorithm runs in $O(n/\delta^{1/88}) \in O(n/\lambda^{1/88})$ rounds, hence the faster of the two algorithms gives the round complexity of $\tilde{O}(n^{1-1/353}D^{1/353}+n^{1-1/706})$. A key component in this $O(n/\lambda^{1/88})$ rounds algorithm is a procedure based on some variant of expander decompositions that carefully determines parts of the graph that can be contracted, while preserves any non-singleton minimum cut. To be more precise, they provide an algorithm that in sublinear time of $\tilde{O}(n/\delta^{1/88})$ rounds, finds a number $k = O(n/\delta^{44})$ of disjoint connected subgraphs $V_1, \ldots, V_k$, with a total induced diameter of $\sum_{i=1}^{k} D(V_i) = O(n/\delta^{1/40})$, such that contracting these subgraphs preserves any non-singleton minimum cut. They then explain an algorithm that in $\tilde{O}(D+k+\sum_{i=1}^{k} D(V_i) = O(n/\delta^{1/40})$ extra rounds, identifies the minimum cut of the contracted graph.

Our Improvement: Our proposal is to replace the clever and somewhat sophisticated procedure of Daga et al. [DHNS19] for finding these contractions with just a random 3-out contractions[4]. From Lemma 2.5, we know that the number k of components is $\tilde{O}(n/\delta)$. In Lemma 5.1, we prove an additional nice property: that the components of 3-out has a summation of diameters $\sum_{i=1}^{k} D(V_i) = O(n/\delta)$. Hence, just plugging 3-out in the framework of Daga et al. [DHNS19], we can improve their $O(n/\delta^{1/88}) \in O(n/\lambda^{1/88})$ round algorithm to run in just $O(n/\delta) = O(n/\lambda)$ rounds. Combining this again with the $\tilde{O}((D + \sqrt{n})\lambda^4)$-round algorithm of Nanongkai and Su [NS14] gives us our claimed $\tilde{O}(n^{0.8}D^{0.2} + n^{0.9})$ round complexity[5].

[4]We note that in fact 2-out would also suffice here. However, the proof for the counterpart of Lemma 5.1 for 2-out is slightly more detailed, and thus, we leave that to the full version of this paper.

[5]We believe that by plugging in the result of our contraction — concretely, $\tilde{O}(n/\delta)$ components, each of diameter

3.3 MPC model

In Section 6, we give an algorithm that computes a minimum cut of a simple graph in $\mathcal{O}(1)$ round of the MPC model, which proves Theorem 1.3. The algorithm is based on our contraction process, after which the input graph is sufficiently small and can be gathered in the memory of a single machine, which can compute the result locally. In Section 6, we focus on efficient implementation of the contraction process.

Lemma 6.1. *Given a simple input graph G, with n vertices, m edges, and minimum degree δ it is possible to execute $\Theta(\log n)$ contraction processes, each resulting with a graph with $\mathcal{O}\left(\frac{n}{\delta}\right)$ vertices and $\mathcal{O}(n)$ edges, in $\mathcal{O}(1)$ rounds of the MPC model, with $\mathcal{O}(m) + \mathcal{O}(n \log^5 n)$ global memory, and $\mathcal{O}(n)$ memory limit for a single machine. With high probability one of computed contractions preserves a fixed non singleton minimum cut of G.*

The first part of the contraction process (i.e., contracting a random 2-out) requires us to generate $G^{(2)}$ and identify its connected components. To generate $G^{(2)}$, each vertex decides independently. Since $G^{(2)}$ has only $\mathcal{O}(n)$ edges, we can gather them in $\mathcal{O}(1)$ rounds in the memory of a single machine which then identifies the connected components locally. To reduce the number of edges, instead of using sparse certificates, we use Karger's uniform random contractions, with an early stop. That is, we contract edges one by one, uniformly at random, until the number of edges reaches $O(n)$. We will show that this process preserves the minimum cut with a constant probability. Furthermore, it allows us to provide a more efficient implementation in the MPC model, with just $O(n)$ local memory. To implement this process, we use a known connection between the edges selected for contraction in Karger's process and the edges of a minimum spanning tree of the graph, when we set iid random weights for the edges. Then, the implementation needs to identify this MST, as well as a *cut off* point in Karger's process — discarding some of the edges of the MST — so that the number of edges remaining uncontracted is $O(n)$. To solve these two problems, we make use of the following two tools, in a black-box fashion:

- A parallel MST computation [GN18, JN18], which allows in $O(1)$ rounds computes an MST and thus identifies the contracted edges and their order of contraction, if we were to run the full contraction process of Karger.
- An approach based on linear graph sketching [AGM12], which allows us to estimate the number of uncontracted edges corresponding to each time of the contraction process (i.e., weight threshold in the MST), all in $O(1)$ rounds, hence allowing us to identify a *cut-off* point for the contraction process where the number of remaining edges falls in $O(n)$.

3.4 PRAM model

In Section 7, we give an implementation of the Minimum Cut algorithm for simple graphs in the CREW PRAM model that proves Theorem 1.4. As in the case of sequential model, the algorithm consists of the contraction process followed by application of the state of the art algorithm for general graph. In Section 7 we focus on providing an implementation of the contraction process.

$O(\log^2 n)$, such that contracting them preserves any particular non-trivial minimum-cut with a constant probability, as we shown in Remark 5.2 — into the algorithm of Nanongkai and Su [NS14], we can improve also the complexity of their algorithm to $\tilde{O}((D + \sqrt{n/\lambda})\lambda^4)$. We have discussed this with one of the authors Hsin-Hao Su. However, this claim should be taken with a grain of salt until all the details are written. If correct, that would lead to a further improved overall round complexity of $\tilde{O}(n^{0.8}D^{0.2} + n^{8/9})$. Moreover, it would give a $(1+\varepsilon)$-approximation of minimum cut with round complexity $\tilde{O}((D + \sqrt{n/\lambda}))$, for any constant $\varepsilon > 0$, hence matching the lower bound of Ghaffari and Kuhn [MK13] for any distributed approximation algorithm on simple graphs.

Lemma 7.1. *Given a simple input graph G, with n vertices, m edges, and minimum degree δ it is possible to execute $\Theta(\log n)$ contraction processes, each resulting with a graph with $\mathcal{O}\left(\frac{n}{\delta}\right)$ vertices and $\mathcal{O}(n)$ edges, on a* CREW PRAM *machine, with $\mathcal{O}(m \log^2 n)$ work and depth $\mathcal{O}(\log^2 n)$. With high probability one of computed contractions preserves a fixed non singleton minimum cut of G.*

For the first part of the contraction process (Theorem 2.4), one has to compute connected components, which can be computed in CREW PRAM model in $\mathcal{O}(m)$ work and $\mathcal{O}(\log n)$ depth [PR99], with high probability. For the second part, similar to the MPC model, we use the second method of edge sparsification via Karger contractions, as outlined in Section 3.3. For that, to identify the *cut-off* point in the contraction sequence, we use a Minimum Spanning Maximal-Forest(MSF) algorithm [PR99] together with a binary search (over the random weights) approach: binary search gives us some weight threshold for the edges that we use for computing MSF, and the MSF algorithm show us what are the connected components corresponding to contracted super-vertices. Since we have $\mathcal{O}(\log n)$ sequential queries for the binary search part and $\mathcal{O}(\log n)$ depth for MSF algorithm, in total we require $\mathcal{O}(\log^2 n)$ depth for this part of the algorithm. The work that is being done in this part is $\mathcal{O}(m \log^2 n)$, as we have to compute $\Theta(\log^2 n)$ Minimum Spanning Maximal-Forests, and each requires $\mathcal{O}(m)$ work.

4 Minimum Cut in the sequential model

4.1 An $O(m \log n)$ Algorithm for Minimum Cut

To find the edge connectivity of G in $O(m \log n)$ time, we essentially just apply Gabow's algorithm [Gab91] to the contracted multigraph $\widehat{G}$ from Theorem 2.1. We note that within this time bound, using another algorithm of Gabow [Gab16], we can find the cactus representation described in [DKL76] which elegantly represents all min-cuts of G. In particular, we use Gabow's result in the form described in the lemma below with $M = \widehat{G}$, $m_H = O(n)$, $n_H = O(n/\delta)$, and $k = \delta$, yielding a running time of $O(km_H \log n_H) = O(n\delta \log n) = O(m \log n)$.

Lemma 4.1 ([Gab91])**.** *Given a multigraph M with m_M edges and n_M nodes, and a number k, in $O(km_M \log n_M)$ time, we can decide if the edge connectivity λ_M is below k, and if so, decide λ_M it exactly.*

Proof. Gabow [Gab91] states the running time as $O(m_M + n_M \lambda_M^2 \log n_M))$ where λ_M is the edge connectivity of M. Since λ_M is no bigger than the average degree, his bound is bounded by $O(m_M \lambda_M \log n_M))$. Moreover, using Matula's linear-time approximation algorithm [WM93], we can decide the edge connectivity of λ_M within a factor 3, hence either decide that it is bigger than k, which we report, or that it is at most $3k$, implying that Gabow's algorithm runs in $O(m_M \lambda_M \log n_M)) = O(m_M k \log n_M)$ time. □

We are now ready to show how we find the edge connectivity of G in $O(m \log n)$ time.

Theorem 4.2. *Let G be a simple graph with m edges and n nodes. Then, in $O(m \log n)$ time, whp, we can find the edge connectivity of G as well as a cactus representation of all its minimum cuts.*

Proof. As described above, we apply Lemma 4.1 to $\widehat{G}$ with $k = \delta$ in $O(n\delta \log n) = O(m \log n)$ time. If the edge connectivity of $\widehat{G}$ is above δ, the edge connectivity of G is δ, and all min-cuts are trivial. Moreover, we can easily build a min-cut cactus representation for this case: a star graph with two parallel edges to all min-degree vertices in G. Otherwise the edge connectivity of G is the minimum of δ and that of $\widehat{G}$. In this case, we can apply Lemma 4.1 to $\widehat{G}$, which runs in $O(n\delta \log n) = O(m \log n)$ time. Furthermore, we can also apply Gabow's cactus algorithm [Gab16] to $\widehat{G}$ in $O(m \log n)$ time. In [KT19], it is detailed how we convert the cactus of $\widehat{G}$ to one of G. □

Corollary for Dynamic Graphs: Goranci et al. [GHT18] have shown how the edge contraction from [KT19] preserving all 3/2-small cuts can be used in an incremental (edge insertions only) algorithm maintaining edge connectivity. By plugging Theorem 2.1 instead of the algorithm from [KT19] in their framework, we get the following corollary.

Corollary 4.3. *We can maintain the exact edge connectivity of an incremental dynamic simple graph, whp, in $O(\log n)$ amortized time per edge insertion.*

We note the above bound holds against an adaptive user where future updates may depend on answers to previous queries.

4.2 Faster Contractions and an $O(m + n\log^3 n)$ Algorithm for Minimum Cut

We now present a faster contraction algorithm for dense graphs. Later in Theorem 4.6, we explain how this leads to an $O(m + n\log^3 n)$ algorithm for edge connectivity.

4.2.1 Faster Contraction via Data Structures

Theorem 4.4. *Let G be a simple graph with m edges, n nodes, and min-degree δ. We have a randomized algorithm, running in $O(m + n\log(n)\alpha(n, n))$ time, contracting edges of G so that the resulting multigraph $\widehat{G}$ has $O(n)$ edges and $O(n/\delta)$ nodes, and preserves all non-trivial $(2-\varepsilon)$-small cuts of G whp.*

Note: In the above statement, α is the extremely slow-growing inverse Ackermann function that Tarjan [Tar75] used to bound the complexity of the union-find data structure. He showed that union-find with u unions and f finds over s elements, initialized as a singleton sets, can be supported in $O(s + \alpha(f, u)f)$ total time. Here α is decreasing in $\lceil f/u \rceil$. We have $\alpha(f, u) = \alpha(u, u)$ if $f \leq u$ and $\alpha(f, u) = O(1)$ if, say, $f \geq u \log\log\log u$. In general, we will use union-find to grow certain forests efficiently, in the following classic way. We are growing a forest F, and the union-find sets are the node sets spanned by the trees in F. Initially, the forest has no edges, and the nodes are singleton set. If we get an edge (u, v), we can use finds on u and v to check if they are spanned by a tree in F. If not, we can add (u, v) to the forest.

Outline: Our main tool to prove Theorem 4.4 is the following on-line data structure version of the contraction process in Lemma 2.9. After presenting this lemma, we use it to prove Theorem 4.4.

Lemma 4.5. *Let G be a given a simple graph with n nodes, m edges, and minimum degree δ. We will construct a randomized $O(n)$ space data structure $\widehat{D}$ that we feed edges from G in any order, but without repetitions. When given an edge, the data structure will answer "preserve" or "contract". The order we feed edges to $\widehat{D}$ may adaptively depend on previous answers made be $\widehat{D}$. The data structure provides the following two guarantees:*

- *The data structure $\widehat{D}$ answers preserve to at most $O(n)$ edges.*
- *Let e^* be any edge of G belonging to some $(2-\varepsilon)$-small cut C^* of G. Then, with probability at least $p_\varepsilon/2$, $\widehat{D}$ will answer preserve if queried on e^* [6]. Here p_ε is the constant probability from Theorem 2.4.*

Finally, if the number of edges fed to $\widehat{D}$ is f, then the total time spent by the data structure is $O(n + \alpha(f, n)f)$.

[6] As a subtle point, note that we are *not* claiming that we with constant probability simultaneously will preserve all edges from C^*.

Proof. The proof has steps that mimic that process of Lemma 2.9, but in a more efficient way and as a data structure. In particular, we will have a part for 2-out contraction, and a more elaborate part that mimics the effect of sparse-certificates. Next, we present these two parts.

First part: Given the graph G, first we make a 2-out sample S. We color all the components of the graph with edge set S in $O(n)$ time so that different components have different colors, i.e., we identify the components. For each vertex v, we store its component color $c(v)$. These component colors are the vertices in G/S. An edge (u, v) from G corresponds to an edge $(c(u), c(v))$ in G/S. Here edges preserve their edge identifies, so if another edge (u', v') has $c(u) = c(u')$ and $c(v) = c(v')$, then $(c(u), c(v))$ and $(c(u'), c(v'))$ are viewed as distinct parallel edges.

By Theorem 2.4, whp, G/S has $n_{G/S} = O(n/\delta)$ vertices. If this is not the case, we create a trivial data structure contract to all edges, so assume $n_{G/S} = O(n/\delta)$ vertices. By Theorem 2.4, the probability that C^* is preserved in G/S is at least p_ε. Assume below that C^* is preserved in G/S.

Second part: This part intends to mimic the effect of sparse certificates—intuitively (though, not formally) similar to growing 4δ maximal forests, one after another. In particular, we initialize $\ell = 4\delta$ union-find data structures to grow edge-disjoint forests $F_1, \ldots, F_\ell$ over the vertices in G/S. Initially, there are no edges in the forests. From a union-find perspective, we can think of it as if we have ℓ disjoint copies of the nodes in G/S, so to ask if $c(u)$ is connected to $c(v)$ in F_i, we ask if $c(u)_i$ is in the same set as $c(v)_i$.

When given an edge (u, v) from G, we can ask if it is connected in some F_i in the sense that $c(u)$ and $c(v)$ belong to the same tree in F_i. If not, we can $(c(u), c(v))$ to F_i. Note that if (u, v) got contracted in G/S, then $c(u) = c(v)$, and then $c(u)$ and $c(v)$ are trivially connected in every forest F_i. We will follow the rule that each edge (u, v) from G may be added as an edge $(c(u), c(v))$ to a single forest F_i. This way the forests remain edge-disjoint, so if $c(u)$ and $c(v)$ are connected in k different forests F_i, then $c(u)$ and $c(v)$ are k-connected by the edges added to all the forests. This implies that u and v must be $k + 1$ connected in G/S.

Now, consider our edge $e^* = (u^*, v^*)$ from our $(2 - \varepsilon)$-small cut C^* of G which we assumed was preserved in G/S. Then $c(u^*)$ and $c(v^*)$ are at most $|C| < 2\delta$ connected in G/S, so $c(u^*)$ and $c(v^*)$ are connected in less than half of the 4δ forests F_i. This leads to the following randomized algorithm to handle a new edge (u, v) from G. We randomly pick $s = \lceil \log_2 2/p_\varepsilon \rceil = O(1)$ indices $i_1, \ldots, i_s \in \{1, \ldots, \ell\}$ and ask if $c(u)$ and $c(v)$ are connected in all the forests $F_{i_1}, \ldots, F_{i_s}$. If not, we add $(c(u), c(v))$ to a single one of them, and answer "preserve". Otherwise, we answer 'contract". The latter would be a mistake on e^*, but assuming that G/S preserved C^*, we know that $c(u^*)$ and $c(v^*)$ are connected in less than half the forests, and then the probability of this mistake is bounded by $1/2^s < p_\varepsilon/2$.

Overall error probability: For the overall error probability on answering "contract" on e^*, we also have to consider the possibility that the G/S did not preserve C^*. This happened with at most $1 - p_\varepsilon$. We conclude that the probability that we answer "preserve" on e^* is at least $p_\varepsilon/2$ as desired.

The number of preserved edges: The edges we add to F_i form a forest over the $O(n/\delta)$ nodes in G/S, so we can only add $O(n/\delta)$ edges to each of the $\ell = 4\delta$ different F_i. Thus we conclude that there are at most $O(n)$ edges that we add to the edge-disjoint forests F_i.

Time complexity: Each time we get an edge, we check connectivity of $s = O(1)$ forest, so we make $O(f)$ find operations. We have ℓ copies of each node in G/S, so the total number of elements is $\ell n_{G/S} = O(n)$. We conclude that the total time spent by our data structure is $O(n + \alpha(f, n)f)$. □

Having this helper data structure version of a single contraction process, we are now ready to prove Theorem 4.4. That is, we present an $O(m + n \log(n)\alpha(n, n))$ time contraction down to $O(n)$ edges and $O(n/\delta)$, while preserving all non-singleton $(2 - \epsilon)$ minimum cuts, with high probability.

Proof of Theorem 4.4. We are going to use the data structure from Lemma 4.5 in much the same way as we used contracted graph from Lemma 2.9. Concretely, we want to amplify the success using certain repetition and voting rules so that we preserve all non-singleton $(2-\epsilon)$ minimum cuts, with high probability. Below we present a direct translation, and later we show how to tune it.

The procedure: Let $p'_\varepsilon = p_\varepsilon/2$ be the error probability from Lemma 4.5. We apply the lemma $q' = O((\log n)/p'_\varepsilon) = O(\log n)$ times, initializing independent data structures $\widehat{D}_1, \ldots, \widehat{D}_{q'}$.

Consider any edge e^* belonging to some $(2-\varepsilon)$-small cut C^* of G. Each $\widehat{D}_i$ preserves e^* with probability at least p'_ε, so the expected number of $\widehat{D}_i$ preserving e^* is at least $\mu' = p'_\varepsilon q'$. By Chernoff, the probability that less than $r' = \mu'/2 = p'_\varepsilon q'/2$ of the $\widehat{S}_i$ want to preserve e^* is bounded by $\exp(-\mu'/8) = \exp(-p'_\varepsilon q')$. For any given γ, this is $O(n^{-\gamma})$ for $q' \geq 8(\ln n)\gamma/p'_\varepsilon = 16(\ln n)\gamma/p_\varepsilon$.

We can now take each edge e in G, and query all the $\widehat{S}_i$ counting how many answer preserve. If this number is less than r', then we contract e. Otherwise we say that e was *voted preserved*. However e could still be lost as a self-loop due to other contractions.

The number of preserved edges: Since each $\widehat{S}_i$ answers preserve for $O(n)$ edges, the total number of preserve answers from all the $\widehat{S}_i$ is $q' \cdot O(n)$. The number of edges that are preserved r' times is therefore $q \cdot O(n)/r' = O(\gamma n/p_\varepsilon) = O(n)$. All other edges are contracted, so the resulting graph $\widehat{G}$ ends up with $O(n)$ edges, as desired for Theorem 4.4.

We preserving all small cuts: As described above, our edge e^* from some $(2-\varepsilon)$-small cut was voted preserved with probability $1 - O(n^{-\gamma})$. This implies that with probability $1 - O(n^{-\gamma'})$ with $\gamma' = \gamma - 2$, every edge e belonging to any $(2-\varepsilon)$-small cut is preserved. In particular, given any $(2-\varepsilon)$-small cut C, we get that all edges in C are voted preserved meaning that none of them are contracted directly. Because C is a cut this implies that all of C survives the contractions, that is, none of the edges in C can be lost as self-loops due to other contractions. Thus we conclude that, whp, $\widehat{G}$ preserves all $(2-\varepsilon)$-small cuts C in G.

An issue with the complexity, and fixing it: Unfortunately, our total run time is still bad because for every edge in G, we query all q data structures $\widehat{D}_i$. However, as our last trick, we maintain a union-find data structure telling which vertices in G that have already been identified due to previous contractions. We now run through the edges of G as before, but if we get to an edge (u, v) where u and v have already been identified, then we skip the edge since contracting it would have no effect. All the above analysis on the properties of $\widehat{G}$ is still valid.

With the above change, checking if end-points have already been identified, we claim that we can make at most $O(n)$ queries to the data structures. We already saw that we could only vote preserve $O(n)$ times. However, if the data structures instead vote to contract (u, v), then this is a real contraction reducing the number of vertices in $\widehat{G}$, so this can happen at most $n-1$ times.

Every time we query the data structures, we query all $q = O(\log n)$ of them. In each we have $f = O(n)$ queries, so the time spent in each is $O(\alpha(n, n)n)$, adding up to a total time of $O(n(\log n)\alpha(n, n))$. In addition, the top-level union-find data structure over the contracted vertices in $\widehat{G}$ uses $O(n + \alpha(m, n)m) \ll O(m + n \log\log\log n)$ time, so our total time bound is $O(n(\log n)\alpha(n, n))$. This completes the proof of Theorem 4.4. □

4.2.2 Faster Minimum Cut in Dense Graphs

We can now complete our $O(m + n\log^3 n)$ algorithm.

Theorem 4.6. *We can find the edge connectivity and some min-cut of a simple graph G with m edges and n nodes in $O(m+n\log^3 n)$ whp. We can also find the cactus representation of all min-cuts of G in $O(m + n\log^{O(1)} n)$ time.*

Proof. Much like we used Gabow's algorithm's [Gab91, Gab16] on the contracted graph in Theorem 2.1 to prove Theorem 4.2, we now apply Karger's [Kar00] edge connectivity algorithm to the contracted graph in Theorem 4.4, and his algorithm with Panigrahi [KP09] to get the cactus representation. □

Note that because we only spend $O(m + n(\log n)\alpha(n, n))$ time on constructing the contracted graphs, we would instantly get better results, if somebody found an improvement to Karger's algorithm [Kar00]. However, due to parallel edge resulting from contractions, it has to be an algorithm working for general graphs, so we cannot, e.g., use the recent algorithm of Henzinger et al. [HRW17].

5 Minimum Cut in the CONGEST model

To obtain the $\tilde{\mathcal{O}}\big(n^{0.8}D^{0.2} + n^{0.9}\big)$ bound claimed by Theorem 1.2, we use the general approach proposed by Daga et al. [DHNS19], except that we replace a key component of their work (a contraction based on expander decompositions that preserves nontrivial minimum cuts) with random 3-out contractions. This leads to a substantial simplification and time complexity improvement.

In their work [DHNS19], Daga et al. propose an algorithm that given a partition of vertices into disjoint connected sets $P = \{V_1 \cup V_2 \cdots \cup V_k\}$, such that the edges on non singleton minimum cuts are only between the sets of vertices — i.e., contracting these sets preserves non-trivial minimum cut — finds the minimum cut of the graph G/P in time $\tilde{\mathcal{O}}(D(V)+k+\sum_{i=1}^{k} D(V_i))$. Here, G/P is a graph in which we contract all sets $V_1, V_2 \dots V_k$ into vertices and $D(S)$ is a diameter of a graph induced by S. We will make use of this algorithm, in a black-box fashion. But let us briefly discuss how Daga et al. used it: They provide an algorithm based on *expander decompositions* that in sublinear time of $\tilde{O}(n/\delta^{1/88})$, finds such a partition with sublinear $k = O(n/\delta^{44})$ and $\sum_{i=1}^{k} D(V_i) = O(n/\delta^{1/40})$. This leads to an $\tilde{O}(n/\delta^{1/88})$ rounds algorithm, which is sublinear for graph with sufficiently large minimum cut. They then obtain their round complexity by combining this algorithm for graphs with large minimum cuts with the algorithm by Nanongkai and Su [NS14] for graphs with small minimum cut.

To get a faster algorithm, we observe that random 3-out contractions provide a vastly simpler and also more efficient way of contracting the graph into fewer nodes (where contractions have small diameter on average), while preserving non-trivial minimum cuts, as desired in the algorithm of Daga et al. [DHNS19]. As a result, by plugging in 3-out contractions in the approach of Daga et al., we obtain an $\tilde{O}(n/\delta)$ round algorithm. This is a substantial improvement on the $\tilde{O}(n/\delta^{1/88})$ round algorithm of Daga et al. (notice that these algorithms will be applied for graphs with large minimum cut and thus large δ). Again, combining this with the algorithm of Nanongkai and Su [NS14] for graphs with small minimum cut gives the final round complexity.

In the next two subsections, we first analyze the diameter of the components of a 3-out, and then plug this property into the framework of Daga et al. [DHNS19] to obtain our faster distributed edge connectivity algorithm.

5.1 Diameter of the components of 3-out subgraph

We now upper bound the total diameter of the connected components of the 3-out subgraph[7].

Lemma 5.1. *With high probability, the summation of the diameters of the components of a random 3-out is in $O(n \operatorname{poly}(\log n)/\delta)$.*

[7] As mentioned before, a similar statement holds for a random 2-out. However, the proof is slightly more detailed, and thus, we leave that to the full version of this paper.

Proof Sketch. If $\delta \leq \log^2 n$, the statement holds trivially. Suppose $\delta \geq \log^2 n$. We analyze the diameter in two phases. In the first phase, we expose a 2-out, which provides that we have $\mathcal{O}(n/\delta)$ connected components, hence we have $\mathcal{O}(n/\delta)$ connected components of size $\mathcal{O}(\log n)$.

Now, we consider components that have diameter $\Omega(\log n)$ and decompose each of these components into fragments of $L = \Theta(\log n)$ length: for that, we take a BFS tree of the component and chop it in levels that are multiples of $\Theta(\log n)$. Hence, each fragment has diameter $O(\log n)$ and also at least $L = \Theta(\log n)$ nodes. We call these *normal* fragments. Those $O(n/\delta)$ components that have size less than $O(\log n)$ are called *small* and we call their nodes *small nodes*.

In the second phase, we expose the 3rd choice of each vertex, as follows: let us start with an arbitrary vertex v. If v is one of the components with less than $\Theta(\log n)$ vertices, we stop the process and say that v *arrives at a small fragment*. Otherwise, we expose the 3rd choice of the nodes in the fragment of v, one by one. Each edge connects us to some fragment. If any of these fragments is a small fragment, again we stop the process and say that v *arrived at a small fragment*. If the total number of nodes in the fragments reachable in this step is at least $\delta/2$, we stop the process. Otherwise, the next edge has at least a $(\delta - \delta/2)/\delta$ chance of connecting us to a new fragment. Hence, over the course of the $O(\log n)$ edges of one fragment, whp, we should either get to fragments with a total of at least $\delta/2$ nodes, or get to $\Omega(\log n)$ new fragments.

We continue the process in a BFS manner, level by level (one fragment per level), and expose the 3rd choice from nodes of each of the new fragments. That is, we go over the newly found fragments one by one, and inside each, we expose the edges of each of the nodes sequentially. The process continues one fragment depth per iteration, until we either arrive at a small fragment or the total number of nodes reachable has exceeded $\delta/2$. So long as neither of these two happened, per layer, with high probability, each fragment connects us to at least $\Omega(\log n)$ new fragments. Here, *new* means that these are fragments that were not reachable by the connections of the previously exposed fragments. Hence, the number of fragments is growing exponentially with the (fragment) depth of the BFS. Therefore, within $O(\log \delta)$ BFS layers, whp, we have either arrived at either at least $\delta/2$ vertices or one small small fragment. Moreover, this holds with high probability for any starting vertex v: within $R = O(\log \delta \log n)$ distance in the exposed edges, it can reach to either at least $\delta/2$ vertices or to one of the $O(n/\delta)$ small nodes.

Let us consider an arbitrary connected component $\mathcal{C}$ of the 3-out. Let $N_{\mathcal{C}}$ be the number of vertices in it. Let s and t be two vertices of $\mathcal{C}$ that have the maximum distance from each other, in $\mathcal{C}$, say they have distance $d_{\mathcal{C}}$ (which is the diameter of $\mathcal{C}$). Moreover, suppose that $\mathcal{C}$ contains $X_{\mathcal{C}}$ many small nodes. We first show that $d_{\mathcal{C}} \leq (\frac{N_{\mathcal{C}}}{\delta} + X_{\mathcal{C}}) \cdot O(\log^2 n)$. Take the shortest path between s and t and mark $d_{\mathcal{C}}/O(\log^2 n)$ vertices that are pairwise at least $2R + 1 = \Omega(\log^2 n)$ apart from each other. From each of these marked vertices, within R distance, we reach either $\delta/2$ nodes or at least one small node. These $O(\log^2 n)$ neighborhoods are disjoint, as the marked nodes have distance at least $2R + 1$ on the shortest path between s and t and thus also on the shortest path connecting them. Since there are only $X_{\mathcal{C}}$ small nodes in this component, there must be at least $d_{\mathcal{C}}/O(\log^2 n) - X_{\mathcal{C}}$ marked vertices that reach at least $\delta/2$ nodes, in their R distance. Hence, $N_{\mathcal{C}} \geq (d_{\mathcal{C}}/O(\log^2 n) - X_{\mathcal{C}})\delta/2$. This implies $d_{\mathcal{C}} \leq (\frac{N_{\mathcal{C}}}{\delta} + X_{\mathcal{C}}) \cdot O(\log^2 n)$. Now, summing up the diameters of all components, we get

$$\begin{aligned}\sum_{\text{component } \mathcal{C}} d_{\mathcal{C}} &\leq \sum_{\text{component } \mathcal{C}} \left(\left(\frac{N_{\mathcal{C}}}{\delta} + X_{\mathcal{C}}\right) \cdot O(\log^2 n)\right) \\ &= \mathcal{O}(\log^2 n)\left(\sum_{\text{component } \mathcal{C}} \frac{N_{\mathcal{C}}}{\delta} + \sum_{\text{component } \mathcal{C}} X_{\mathcal{C}}\right) \\ &\leq \mathcal{O}(\log^2 n)(\mathcal{O}(n/\delta) + O(n/\delta)) \\ &\leq \mathcal{O}\left(\frac{n \log^2 n}{\delta}\right).\end{aligned}$$

□

Remark 5.2. *We can choose a subset of the edges of the* 3*-out so that the spanning subgraph defined by them has* $O(n/\delta \operatorname{poly}(\log n))$ *components, each with diameter* $\operatorname{poly}(\log n)$*. If contracting the* 3*-out preserves a cut, contracting this subset of edges also preserves it.*

Proof Sketch. Consider the spanning graph defined by the edges of a 3-out (and discard all graph edges not in it). Mark each node randomly with probability $O(\log n)/\delta$. Also, mark all small nodes. Then, perform a BFS from each marked node for $R = O(\log^2 n)$ depth, all simultaneously, while assigning each node to the first BFS that reaches it. Hence, we are growing vertex-disjoint BFS trees. As shown in the proof of Lemma 5.1, each node has either at least $\delta/2$ nodes or at least one small node in its distance $R = O(\log^2 n)$. Hence, each node will be in one of the BFS trees, with high probability. Each BFS tree is one of our components, and we remove the contracted edges connecting different final BFS trees. Since we have $n \operatorname{poly}(\log n)/\delta$ randomly marked nodes and $\mathcal{O}(n/\delta)$ small nodes, the total number of BFSs and thus also the total number of components is $O(n/\delta \operatorname{poly}(\log n))$. The diameter of each component is clearly at most $2R = O(\log^2 n)$. □

5.2 Improved Distributed Algorithm

Lemma 5.3. *Given any simple input graph* G *with* n *vertices,* m *edges, minimum degree* δ*, and minimum cut size* λ*, it is possible to identify its minimum cut in* $\tilde{\mathcal{O}}\left(\frac{n}{\delta}\right) = \tilde{\mathcal{O}}\left(\frac{n}{\lambda}\right)$ *rounds of the* CONGEST *model, w.h.p.*

Proof. Consider a 3-out contraction of the input graph. Since we consider simple graphs, we have that the graph diameter has a small diameter $D(G) = \mathcal{O}\left(\frac{n}{\delta}\right)$. By Lemma 2.5 we have that in the contracted graph we have at most $\mathcal{O}\left(\frac{n}{\delta}\right)$ vertices. By Lemma 5.1 we have that in the 3 out contraction the sum of diameters of contracted subgraphs is $\tilde{\mathcal{O}}\left(\frac{n}{\delta}\right)$. Therefore, we can identify the minimum cut of the graph obtained by a 3-out contraction in $\tilde{\mathcal{O}}\left(\frac{n}{\delta} + \frac{n}{\delta} + \frac{n \operatorname{poly}\log n}{\delta}\right) = \tilde{\mathcal{O}}\left(\frac{n}{\delta}\right) = \tilde{\mathcal{O}}\left(\frac{n}{\lambda}\right)$ rounds, by applying the algorithm of Daga et al. [DHNS19]. □

By combining Lemma 5.3 with the algorithm of [NS14], which is best suited for graphs with small edge connectivity λ, we get our round complexity of $\tilde{\mathcal{O}}\left(n^{0.8}D^{0.2} + n^{0.9}\right)$, as claimed in Theorem 1.2:

Proof of Theorem 1.2. In Lemma 5.3, we provided an algorithm with round complexity $\tilde{\mathcal{O}}\left(\frac{n}{\lambda}\right)$. Nanongkai and Su [NS14] gave an algorithm that runs in $\mathcal{O}(\lambda^4 \log^2 n(D + \sqrt{n} \log^* n)) = \tilde{\mathcal{O}}(\lambda^4(D + \sqrt{n}))$ rounds. We now explain that by running both algorithms and taking the faster of the two, we can obtain an algorithm with complexity $\tilde{\mathcal{O}}\left(n^{0.8}D^{0.2} + n^{0.9}\right)$. For graphs with diameter $D \in \mathcal{O}(\sqrt{n})$, the second algorithm requires $\tilde{\mathcal{O}}(\sqrt{n}\lambda^4)$ rounds. Taking the minimum of this and our algorithm that runs in $\tilde{\mathcal{O}}\left(\frac{n}{\lambda}\right)$ gives an algorithm that runs in $\tilde{\mathcal{O}}\left(n^{0.9}\right)$ rounds. For graphs with diameter $D \in \Omega(\sqrt{n})$,

the algorithm of Nanongkai and Su runs in $\tilde{\mathcal{O}}(D\lambda^4)$ rounds. Taking the minimum of this and our algorithm that runs in $\tilde{\mathcal{O}}\left(\frac{n}{\lambda}\right)$ gives an algorithm that runs in $\tilde{\mathcal{O}}(n^{0.8}D^{0.2})$ rounds. Hence, running both algorithms and taking the faster of the two runs in $\tilde{\mathcal{O}}\left(n^{0.8}D^{0.2}+n^{0.9}\right)$ time. □

6 Minimum Cut in the MPC model

The MPC model is a model of parallel (or distributed) computing, in which the computation is executed in synchronous rounds, by a set of p machines, each with *local memory* of size S. Every rounds consists of the phase of local computation and the phase of communication. In the phase of local computation each machine can execute some, possibly unbounded computation (although we could consider only computation that takes time polynomial in S, or even only $\tilde{\mathcal{O}}(S)$ step computations). In the phase of communication, the machines simultaneously exchange $\mathcal{O}(\log n)$-bit messages, in a way that each machine is sender and receiver up to $\mathcal{O}(S)$ messages.

The *global memory* is the total amount of memory that is available, i.e. if there are p machines, each with S memory, their global memory is pS. Ideally, for an input of size N, the values of p and S are chosen in a way that the global memory is $\mathcal{O}(N)$. Assuming that the global memory limit is set to be $\mathcal{O}(N)$, we have two main quality measures of the algorithms in the MPC model: the first one is the number of rounds that are required by the algorithm to finish computation, the second is the limit on the local memory of a single machine.

For graph problems in the MPC model, we distinguish two significantly different variants of the MPC model, depending on relation between the limit on local memory S and the number of vertices of the input graph, which is usually denoted by n. More precisely, those variants are $S \in \tilde{\Theta}(n)$ and $S \in \mathcal{O}(n^{1-\varepsilon})$ for some constant $\varepsilon > 0$. In this paper, we focus on the variant in which the limit on the local memory is $\mathcal{O}(n)$ words, each of length $\mathcal{O}(\log n)$ bits.

Sometimes, we also consider the algorithms that have global memory limit larger than N. It can be achieved in two ways: by setting higher limit on the local memory of a single machine, or by increasing the number of machines. The first variant of this relaxation is stronger – a single machine with enlarged memory limit can simulate several machines with a smaller limit. Therefore, if we have two MPC machines, both with the same global memory, the one with larger limit on the local memory can simulate the one with a smaller limit on local memory.

In particular, in this section we give a Minimum Cut algorithm for simple graphs in the MPC model, that proves Theorem 1.3, i.e. the algorithm requires $\mathcal{O}(1)$ rounds of computation, uses $\mathcal{O}(m + n\log^5 n)$ global memory, while respecting $\mathcal{O}(n)$ memory limit on a singe machine.

6.1 Minimum Cut for simple graphs in $\mathcal{O}(1)$ rounds of the MPC model: Outline

The remaining part of this section is dedicated to give a proof of Theorem 1.3. To do so, it is enough to use a variant of contraction process that preserves a fixed non singleton minimum cut with a constant probability, described in in Theorem 2.2. The reason is that a simple implementation of probability amplification method described in Section 2.4 requires us to execute $\Theta(\log n)$ contraction processes from Theorem 2.2. Since each of the results fits into local memory of a single machine, at this point we can just identify a minimum cut of each of contracted graphs, and take the smallest as a minimum cut of the input graph (if it is no larger than the minimum degree.

In order to provide an efficient Minimum Cut algorithm it is enough to provide an efficient implementation of the algorithm emerging from Theorem 2.2.

Lemma 6.1. *Given a simple input graph G, with n vertices, m edges, and minimum degree δ it is possible to execute $\Theta(\log n)$ contraction processes, each resulting with a graph with $\mathcal{O}\left(\frac{n}{\delta}\right)$ vertices and $\mathcal{O}(n)$ edges, in $\mathcal{O}(1)$ rounds of the MPC model, with $\mathcal{O}(m)+\mathcal{O}(n\log^5 n)$ global memory, and $\mathcal{O}(n)$*

memory limit for a single machine. With high probability one of computed contractions preserves a fixed non singleton minimum cut of G.

In the remaining part of this section we provide an algorithm that proves Lemma 6.1. We can easily implement the 2-out contraction process, as captured by Theorem 2.4, which is the first part of Theorem 2.2. This is a simple connected components computation on a graphs that have $\mathcal{O}(n)$ edges each. Having only linear number of edges allows us to gather all the edges in the memory of a single machine and solve the problem locally. The more challenging part is reduce the number of edges to $\mathcal{O}(n)$. We next describe how we do this in $\mathcal{O}(1)$ round of computation with $O(n)$ local memory and using $\mathcal{O}(m) + \tilde{\mathcal{O}}(n)$ global memory. In order to achieve a local memory of $O(n)$, we present a second randomized method for reducing the number of edges, which can be seen as a counterpart of the Nagamochi-Ibaraki sparse certificates, stated in Lemma 2.8. In this second method, instead we use Karger's random contraction method, but stopped early enough. Next, we outline this as a general method, and then explain how to implement that in the MPC model.

Reducing the number of edges via Karger contractions: We next explain Karger's random contraction process [Kar93] and how it can help us to reduce the number of edges while preserving the minimum cut. A single random contraction on a graph is executed as follows. We select an edge uniformly at random, replace both endpoints of this edge with a single super-vertex and remove any self loops that were created in the process (note that this process does not remove parallel edges). A random contraction process of Karger is a sequence of single contractions, each executed on the result of the previous one.

Lemma 6.2. *Random contraction process of Karger applied on an unweighted multigraph M with n_M vertices and minimum cut size λ_M, and executed so long as the when contracted graph G' has $\Omega(n_M\lambda_M)$ edges, has a constant probability of contracting an edge from a fixed $(2-\varepsilon)$ minimum cut, for a constant ε. Hence, we can perform this process iteratively, after a 2-out contraction, so long as $\Omega(n)$ edges remain and obtain a contracted graph $\widehat{G}$ with $O(n/\delta)$ nodes and $O(n)$ edges, which with at least a constant probability p_ε preserves any given non-trivial $(2-\varepsilon)$-small cut.*

Proof. We argue about the first part of the lemma. The guarantee about the application after 2-out follows immediately, as Theorem 2.4 shows that 2-out has $O(n/\delta) \leq O(n/\lambda)$ vertices.

As long as the number of edges in the graph is larger than $cn_M\lambda$, the probability that by contracting an edge chosen uniformly at random we contract an edge from a fixed (2-ε) minimum cut is at most $\frac{(2-\varepsilon)\lambda}{cn_M\lambda_M} = \frac{(2-\varepsilon)}{cn_M}$. Since the number of contractions that can be executed on a n_M node graph is at most $n_M - 1$, we have that the probability that a contraction process that did not reduced the number of edges below $c\lambda_M n_M$ threshold contracted some edge from a fixed minimum cut is at least

$$\left(1 - \frac{(2-\varepsilon)}{cn_M}\right)^{n_M} > e^{-\frac{(2-\varepsilon)}{cn_M}n_M} = e^{-\frac{(2-\varepsilon)}{c}}$$

Since we can not contract a graph to a single node without contracting an edge from the fixed $(2-\varepsilon)$ minimum cut, at some point the contraction process reduced the number of edges to $\mathcal{O}(n_M\lambda)$. □

To apply this theorem algorithmically, we have to be able to stop the contraction process just after the first step that reduced the number of edges below the threshold $cn\lambda$, as continuation of the contraction process would decrease the probability of preserving the fixed minimum cut. Given a contraction process, we define the *cut-off* problem as problem of identifying the first contraction after which the number of the remaining edges is $\mathcal{O}(n_M\lambda)$.

Contractions via MST, and the Cut-Off Problem: As first observed by Karger [Kar93], one can think about the random contraction process as computing MST via Kruskal's algorithm [Kru56]

on a graph with weights assigned randomly from a large enough set of weights. This view provides as alternative formulation of the *cut-off* problem — i.e., identifying the first contraction after which the number of the remaining edges is $\mathcal{O}(n_M \lambda_M)$ —, which we use in our implementation.

To understand the connection between random contraction process and computing MST on a graph with randomly assigned weights, let us consider the following implementation of the contraction process. By definition of a single contraction, in each step of the process, we want to select an edge uniformly at random. One way to do that is to apply a random permutation on the edges, and for each contraction use the first edge (in the ordering defined by the random permutation), that is still between the (super-)vertices. One way of defining a random permutation on a set of elements is to assign to them random weights and sort them according to the assigned edges. As long as during the assignment there was no collisions, this ordering defines uniformly random permutation on the elements. Thus, we can argue that the random contraction process with random ordering defined by the randomly assigned weights, does exactly the same thing as the Kruskal's MST algorithm [Kru56] on the graph with exactly the same weights. In each step, both algorithms use exactly the same edge: Kruskal's algorithm adds it to the MST (and merges some two connected components), and contraction algorithms contracts this edge (and replaces two (super)vertices into one). Hence, after k steps of both algorithms, the set of connected components in Kruskal's algorithm corresponds to the set of (super)vertices of the contraction process. To avoid the collisions during the random assignment of the weights, we can choose them from the range $\{1, \ldots, n_M^{10}\}$, which is small enough to represent the weights on $\mathcal{O}(\log n)$ bit words, but large enough to provide high probability that there is no collision during the assignment.

For our purpose we want to stop the contraction process at the first contraction that caused the number of remaining edges to drop below some $cn_M\lambda_M$ threshold. As we already mentioned, we can express this problem using language of MST computation: we want to identify the smallest weight x, such that the number of edges between connected components of the graph induced by the edges lighter than x is smaller than $cn_M\lambda_M$. Hence, in order to identify a *cut-off* point, it is enough to compute the number of edges between connected components of the input graph after each step of Kruskal's algorithm. In fact, it is enough to know a constant approximation of this number, as the only thing we care about is to get less than $cn_M\lambda_M$ edges, for *some* c. In order to make Lemma 6.2 work, it is enough to provide that c is *a constant* and we do not need to know its exact value.

6.2 Karger's contraction process in MPC

As mentioned above, the Karger contraction process can be computed via MST computation. To implement several processes in parallel, we use an approach presented in the paper [GN18], which uses parallel MST computation to do so.

The $\mathcal{O}(1)$ round Congested Clique algorithm for the MST problem [JN18] can be implemented in $\mathcal{O}(m)+\mathcal{O}(n\log^3 n)$ total communication, which means that in can be executed on the MPC machine with total memory $\mathcal{O}(m+n\log^3 n)$, and local memory $\mathcal{O}(n)$. Hence, for graphs with $m \in \mathcal{O}(n\log^2 n)$, execution of $\Theta(\log n)$ independent instances of the algorithm requires $\mathcal{O}(\log n \cdot (m + n\log^3 n)) = \mathcal{O}(n\log^4 n)$ global memory.

In order to give a $\mathcal{O}(m)$ algorithm that works for denser graphs, we use the fact that the MST algorithm is based on the Karger-Klein-Tarjan random sampling (KKT sampling) approach to the MST problem. On the top level, it allows to reduce a single instance of the MST problem to two, such that the first has $\mathcal{O}(mp)$ edges and the second has $\mathcal{O}(np^{-1})$.

More importantly, for $p^{-1} \in \Omega(\log n)$, the total communication required by the reduction step is $\mathcal{O}(np^{-1}\log^2 n)$ [PS16, Now18]. Hence, if at the beginning we use one step of KKT sampling, with $p^{-1} \in \Theta(\log n)$, the first instance has only $\mathcal{O}\left(\frac{m}{\log n}\right)$ edges. Since we have $\Theta(\log n)$ contraction processes to simulate, the total communication complexity of solving all first instances in parallel is

$\mathcal{O}\Big(\log n \cdot \Big(\frac{m}{\log n} + n\log^3 n\Big)\Big) = \mathcal{O}(m + n\log^4 n)$. The reduction step, in all MST instances we solve in parallel, requires $\mathcal{O}(\log n \cdot (n\log^3 n)) = \mathcal{O}(n\log^4 n)$ global memory. Each of the second instances has $\mathcal{O}(n\log n)$ edges, hence we can solve $\Theta(\log n)$ of them using $\mathcal{O}(\log n \cdot (n\log n + n\log^3 n)) = \mathcal{O}(n\log^4 n)$ global memory. Summarizing, solving $\mathcal{O}(\log n)$ instances of the MST problem requires $\mathcal{O}(m + n\log^4 n)$ global memory.

The computed MST's give us an ordering in which the contraction processes merge connected components. Still, for each MST / contraction process, we have to identify the *cut-off* point – a step in the contraction process after which the number of the edges between the connected components / super-vertices cross some threshold cn.

6.3 Identifying a *cut-off* point for Karger's contractions in MPC

The idea is to compute a constant approximation of the number of edges that are left after each contraction. If we have a guarantee that the exact value is at most α times smaller / larger than the actual value, and we pick the threshold value cn to be sufficiently high, then the actual number of edges is at least $\frac{cn}{\alpha}$, which is sufficiently large to provide that we do not contract an edge from a fixed minimum cut with at least positive constant probability. On the other hand, if we cross the threshold, we have that the number of edges is at most $\alpha \cdot cn$, which is $\mathcal{O}(n)$. To compute approximation of the number of edges after each contraction step, we propose a sketch based approach. Firstly, let us recall an implementation of linear graph sketches, and its application to approximate degree estimation. Then, we show an MPC algorithm, that given a sequence of contractions (i.e. computed MST) of a graph, can compute a constant approximation of the number of the remaining edges after each step of contraction process.

Degree estimation via linear sketches: One of the base ideas behind linear graph sketches is that it is possible to encode the set of edges with exactly one endpoint in some set of vertices C on a vector $v(C)$, in such a way that:

- the number of non zero entries in $N(C)$ is twice the number of the edges that have exactly one endpoint in C (actually one can decode the set of edges, but here decoding size is sufficient)
- the encoding function $N(C)$ is linear, i.e. for two sets $C_1, C_2 \subseteq V$, $N(C_1)+N(C_2) = N(C_1 \cup C_2)$

The other building block is a linear sparse recovery data structure. Such data structure allows to encode vectors on some small space, recover the non zero coordinates of a vector if the vector is sparse and it is linear. This means that if a connected component C has small degree, we recover all the edges in a following way:

- compute $N(v)$ for each vertex v from C
- compute a sketch of $N(v)$ for each vertex v from C
- add the sketches of all v, to obtain a sketch of C
- extract the edges from a sketch of C

The example implementation of the sparse recovery data structure is provided by Cormode and Firmani [CF14]: given that we want to recover coordinates of a vector with at most $\mathcal{O}(1)$ non zero coordinates, the size of the data structure is $\mathcal{O}(\log n)$ bits.

On the top level, the linear graph sketch consists of $\mathcal{O}(\log n)$ instances of a linear data structure that allows us to recover, with a positive constant probability, the edges with exactly one endpoint in C, if the degree of C is $\mathcal{O}(1)$. To obtain a random edge, it selects a uniformly random edge from the recovered set. Let us recall the analysis of this implementation of linear sketch provided by Cormode and Firmani [CF14]. Let G_i be a random subgraph of G, in which we include each edge with probability $\frac{1}{2^i}$. The linear sketch is an application of $\mathcal{O}(1)$ recovery data structure, to G_i, for all $i \in \{1, \ldots, 2\log n\}$. Let us consider a connected component C - there exists j, such that

the expected degree of C in G_j is at least $\frac{s}{4}$ and at most $\frac{s}{2}$, for some $s \in \mathcal{O}(1)$. With constant probability degree of C in G_j is positive and smaller than s, hence s sparse recovery data structure can extract the edges outgoing from C in G_j.

The idea of extracting approximate degree of C is to use $\Theta(\log n)$ independent sets of sketches, and count the number of successful recovery on level i, for all $i \in \{1, \ldots, 2\log n\}$. Since some levels have at least positive constant probability of success, having $\Theta(\log n)$ trials give high probability, that some levels will succeed $\Theta(\log n)$ times. If the number of trials is sufficiently large, the number of successful trials will be the largest for j such that the expected value of degree of C in G_j is at least $\frac{s}{4}$ and at most $\frac{s}{2}$. Hence, identifying j gives us an approximate degree of C in G.

The space complexity of a single sketch is $\mathcal{O}(\log n)$ words (one word / $\mathcal{O}(\log n)$ bits per G_i), hence whole degree estimator consist of $\mathcal{O}(\log^2 n)$ words.

Using degree estimators to find a *cut-off* point: The proposed degree estimator is a linear function and works with high probability. The idea is to compute such estimator for each vertex of the graph, and sum them up accordingly to the contractions that are executed on the graph. Since, there are at most $\mathcal{O}(n)$ steps, and the estimators are correct in each of them with high probability, by the union bound they work properly in all steps, with high probability, even though the estimators in consecutive steps are dependent.

The problem is that total amount of information is too large to keep in the memory of a single machine. To bypass this issue, we use the following observation: if we consider a contraction process at some moment, with set of super-vertices V', executed for k steps, it contracts up to $2k$ members of V'. Hence, if k is $\frac{n}{\log^2 n}$, we can gather all the sketches of nodes that are being contracted in a single machine.

Therefore, one can compute approximate degrees of all connected components (or super-vertices) after each step of the contraction process, by splitting whole contraction process into $\Theta(\log^2 n)$ parts, each of length $\Theta\left(\frac{n}{\log^2 n}\right)$, and using the degree estimators. Let consider a single part. Even tough a single machine can gather all the relevant estimators in the local memory, it does not give an algorithm for computing the relevant estimators. The number of the vertex in the relevant super-vertices can be even linear in n - for such super-vertices, we can use the fact that those estimators are linear functions and compute them a two stage manner - firstly, we compute estimators for some partition of such super-vertex into subsets of size $\frac{n}{\log^2 n}$, then we add them up. The total number of vertices in the active super-vertices can not be larger than n, therefore for a single the total number of additional machines required for computing estimators for a single part is $\mathcal{O}(\log^2 n)$. Since, there are $\Theta(\log^2 n)$ parts, for which we compute the estimators, the total number of machines that we need for a single contraction process is $\mathcal{O}(\log^4 n)$, and $\mathcal{O}(\log^5 n)$ for all $\Theta(\log n)$ contraction processes.

Having approximate degrees of all super-vertices that appear during the contraction process, we can gather them in the memory of a single machine, which then can compute approximate number of remaining edges after each contraction, hence can identify a *cut-off* point locally.

6.4 Computing the edges in contracted graphs in MPC

Given that we have a division into connected components (corresponding to contracted vertices) we want to compute the set of edges between them. The naive approach would be to check for each edge, whether the endpoints are in the same connected component or not. This unfortunately, requires $\Theta(m)$ memory for a single contraction process, and $\Theta(m \log n)$ for $\Theta(\log n)$ contraction processes. To bypass this issue, we use the approach based on *fingerprints* [Rab81].

The idea is roughly based on the fact that we can treat the labels of connected components of each vertex in $\Theta(\log n)$ contraction processes as $\Theta(\log^2 n)$ bit strings. Then, we can compute a hash value from some polynomial range, for each $\Theta(\log^2 n)$ bit label. Since we have only $\mathcal{O}(n)$ labels, if we use sufficiently large range of hashing function, with high probability there would be

no collision. Since, the range is polynomial, the value of hash function can be encoded on $\mathcal{O}(1)$ words ($\mathcal{O}(\log n)$ bits). Then, for each edge, we compare fingerprints of the endpoints, if they are the same, the edge does not connect two different super-vertices in any contraction. Hence, in order to compute all edges that connect different super-vertices, it is enough to consider only the edges with the endpoints with different fingerprints.

The number of those edges is at most $\mathcal{O}(n)$ per contraction process, hence $\mathcal{O}(n \log n)$ in total. For each edge, we need to know $\Theta(\log n)$ identifiers of super-vertices, one per contraction process. Thus in total the memory and communication used by the algorithm computing the edges between the super-vertices is $\mathcal{O}(m) + \mathcal{O}(n \log^2 n)$.

7 Minimum Cut in the PRAM model

In this section we provide a CREW PRAM algorithm for the Minimum Cut problem for simple graphs, which is a proof of Theorem 1.4.

The CREW PRAM model is a model of parallel computing. The PRAM machine consists of a set of p processors, and some unbounded shared memory. The computations are performed in synchronous steps, and in each step each processor may read from $\mathcal{O}(1)$ memory cells, evaluate some $\mathcal{O}(1)$ step computable function on read values, and write something to $\mathcal{O}(1)$ memory cells. More precisely, we consider CREW PRAM model, which extends to *Concurrent-Read-Exclusive-Write* PRAM, which means that we allow multiple processors to read from the same memory cell, but we forbid multiple processors to write to a single memory cell in a single step of computation.

To define the complexity of an algorithm in the CREW PRAM model, one can use the *Work-Depth* model [Ble96]. In this model, we perceive a computation as a directed acyclic graph, in which each vertex corresponds to a single step of a processor, its in-edges correspond to the inputs of evaluated function, and out-edges correspond to the results of the evaluated function. In other words, we put an edge between two vertices, if the output of the function evaluated by one vertex is an input of the function evaluated in the other vertex. The work of the algorithm is the number of vertices in the graph, and the depth of the algorithm is the longest directed path in the graph of computation.

The state of the art algorithm for the weighted Minimum Cut problem has $\mathcal{O}(m \log^4 n)$ work and $\mathcal{O}(\log^3 n)$ depth [GG18]. The application of Theorem 2.2 allows us to reduce the number of edges to $\mathcal{O}(n)$. In this section we show an PRAM implementation of Theorem 2.2 that allows us to execute $\Theta(\log n)$ contraction processes in $\mathcal{O}(m \log^2 n)$ work and $\mathcal{O}(\log^2 n)$ depth. The next step is to apply the technique from Section 2.4, which can be done in $\mathcal{O}(m + n \log n)$ work with $\mathcal{O}(\log n)$ depth, and gives a $\mathcal{O}(n)$ edge multigraph preserving $(2-\varepsilon)$-minimum cuts with high probability, as stated in Theorem 2.1. To complete Minimum Cut computation, we still have to identify Minimum Cut in the resulting multigraph, for which we can use the state of the art algorithm for general graphs [GG18], which has $\mathcal{O}(\log^3 n)$ depth, but only $\tilde{\mathcal{O}}(n)$ work.

Lemma 7.1. *Given a simple input graph G, with n vertices, m edges, and minimum degree δ it is possible to compute a $\mathcal{O}(n)$ edge multigraph, that preserves all non singleton $(2-\varepsilon)$-minimum cuts with high probability on a* CREW PRAM *machine, with $\mathcal{O}(m \log^2 n)$ work and depth $\mathcal{O}(\log^2 n)$. With high probability one of computed contractions preserves a fixed non singleton minimum cut of G.*

In the remaining part this section, we prove Lemma 7.1, and briefly discuss that composition of Lemma 7.1 with the state of the art algorithm for general graphs [GG18] proves Theorem 1.4.

Reducing the number of vertices: The implementation of Theorem 2.4 boils down to computing connected components of $\Theta(\log n)$ random 2-out contraction of a graph. Firstly, we generate $\Theta(\log n)$ random 2-out subgraphs. Given that each vertex knows its degree, it can be done in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(1)$ depth. Computing degrees for all vertices can be done in $\mathcal{O}(m)$ work and $\mathcal{O}(\log n)$ depth, which in total gives a $\mathcal{O}(m + n \log n)$ work and $\mathcal{O}(\log n)$ depth protocol. Then we compute connected

components for each of random 2-out subgraphs, which for all $\Theta(\log n)$ instances requires $\mathcal{O}(n \log n)$ work and $\mathcal{O}(\log n)$ depth [PR99].

Reducing the number of edges: The second step of the algorithm is to execute random contraction processes via computing MST on the graph with randomly assigned weights. Since there are $\Theta(m \log n)$ weights to be assigned, this step requires $\mathcal{O}(m \log n)$ work and depth $\mathcal{O}(1)$. In order to identify the *cut-off* point, we want to identify the smallest weight x such that in graph limited only to the weights smaller than x the number of the edges between connected components is smaller than some threshold cn. To identify such x we can use binary search – for each value of x we test, we can compute a MSF in $\mathcal{O}(\log n)$ steps, with $\mathcal{O}(m)$ work, and identify all edges that are between connected components in additional $\mathcal{O}(\log n)$ steps and $\mathcal{O}(m)$ work. Hence, for a given x, computing the number of the edges between the connected components of a graph limited to the edges lighter than x requires $\mathcal{O}(\log n)$ steps and $\mathcal{O}(m)$ work. Since the random weights are selected from some polynomial range, binary search for a single contraction process requires querying $\mathcal{O}(\log n)$ possible values of x. Since the number of queries is $\Theta(\log n)$, total number of steps (depth) of this part of the algorithm is $\mathcal{O}(\log^2 n)$, and the total work that is done is $\Theta(\log^2 n) \cdot \mathcal{O}(m)$.

Merging the results of contractions: At this point, we have $\Theta(\log n)$ multigraphs and each has only $\mathcal{O}(n)$ edges. By using the probability amplifying technique described in Section 2.4, we can transform them into a single graph with $\mathcal{O}(n)$ edges that preserves a all non singleton $(2-\epsilon)$ minimum cuts. To do so, it is enough to compute for each edge what is the number of contraction processes, which preserved that edge and keep only those that were preserved $r \in \Omega(\log n)$ times, and contract all other edges. More precisely, if E_r is the set of edges we want to preserve, we contract all connected components of the graph $G' = (V, E \setminus E_r)$.

In order to compute the number of contraction processes that preserved an edge, we can simply scan over all the results of contraction processes, which requires $\mathcal{O}(n \log n)$ work, and has depth $\mathcal{O}(\log n)$, and split the set of edges into $E \setminus E_r$ and E_r, which can be done via parallel prefix computation in $\mathcal{O}(m)$ work, with depth $\mathcal{O}(\log n)$. The last part is a connected component computation, which can be done in $\mathcal{O}(m)$ work and $\mathcal{O}(\log n)$ depth, and relabeling the edges of E_r so that for each edge we could know what are the identifiers of the endpoints after contractions, which also can be done in $\mathcal{O}(m)$ work and $\mathcal{O}(\log n)$. Therefore, merging the results of contractions from Theorem 2.2 into a graph from Theorem 2.1 requires $\mathcal{O}(m + n \log n)$ work and $\mathcal{O}(\log n)$ depth. This concludes the proof of Lemma 7.1.

Computing Minimum Cut: At this point, we have a single multigraph with $\mathcal{O}(n)$ edges that preserves a minimum cut with high probability. The state of the art CREW PRAM algorithm can compute its minimum cut in $\mathcal{O}(n \log^4 n)$ work, with depth $\mathcal{O}(\log^3 n)$.

Therefore, the whole algorithm require $\mathcal{O}(m \log^2 n + n \log^4 n)$ work and has $\mathcal{O}(\log^2 n + \log^3 n) = \mathcal{O}(\log^3 n)$ depth, which concludes the proof of Theorem 1.4.

Acknowledgment

The first two authors are thankful to the Uber driver in Wroclaw whose late arrival provided ample time for a conversation about (massively parallel) algorithms for min-cut; it was during that conversation that the idea of random out contractions was sparked. We are also thankful to Danupon Nanongkai for discussions about the work in Daga et al. [DHNS19] and for informing us that if we can upper bound the diameter of the components in 3-out, we can further improve our distributed algorithms using the algorithm of Daga et al. [DHNS19]; that led us to prove Lemma 5.1, which improved our distributed round complexity from $\tilde{O}(n^{1-1/9}D^{1/9} + n^{1-1/18})$ to $\tilde{O}(n^{0.8}D^{0.2} + n^{0.9})$.

References

[ABB+19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet edcs: algorithms for matching and vertex cover on massive graphs. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2019.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.

[ASS+18] Alexandr Andoni, Clifford Stein, Zhao Song, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 674–685, 2018.

[ASW19] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, page to appear, 2019.

[BBD+19] Soheil Behnezhad, Sebastian Brandt, Masha Derakhshan, Manuela Fischer, Mohammad-Taghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, page to appear, 2019.

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: quantum and mapreduce. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 1170–1189, 2018.

[BFU19] Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in mpc: Fast mis on trees with strongly sublinear memory. In *26th International Colloquium on Structural Information and Communication Complexity*, page to appear, 2019.

[BHH19] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G Harris. Exponentially faster massively parallel maximal matching. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, page to appear, 2019.

[Ble96] Guy E. Blelloch. Programming parallel algorithms. *Commun. ACM*, 39(3):85–97, March 1996.

[CF14] Graham Cormode and Donatella Firmani. A unifying framework for ℓ0-sampling algorithms. *Distrib. Parallel Databases*, 32(3):315–335, September 2014.

[CFG+19] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of (delta + 1)-coloring in congested clique, massively parallel computation, and centralized local computation. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, page to appear, 2019.

[CLM+18] Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 471–484, 2018.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation (OSDI)*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, page to appear. ACM, 2019.

[DKL76] Efim A. Dinitz, A. V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka, Moskow, 1976. (in Russian).

[Doe18] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. *CoRR*, abs/1801.06733, 2018.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FF62] Lestor R Ford and DR Fulkerson. Flows in networks. 1962.

[FJ17] Alan Frieze and Tony Johansson. On random k-out subgraphs of large graphs. *Random Structures & Algorithms*, 50(2):143–157, 2017.

[Gab91] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 112–122. ACM, 1991.

[Gab16] Harold N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Trans. Algorithms*, 12(2):24:1–24:73, 2016. Announced at FOCS'91.

[GG18] Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 1–11, New York, NY, USA, 2018. ACM.

[GGK+18] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*. arXiv:1802.08237, 2018.

[GH61] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[GHT18] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Trans. Algorithms*, 14(2):17:1–17:21, 2018.

[GKMS19] Buddhima Gamlath, Sagar Kale, Slobodan Mitrović, and Ola Svensson. Weighted matchings via unweighted augmentations. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, page to appear, 2019.

[GKU19] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, page to appear, 2019.

[GN18] Mohsen Ghaffari and Krzysztof Nowicki. Congested clique algorithms for the minimum cut problem. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 357–366, New York, NY, USA, 2018. ACM.

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 1636–1653, 2019.

[HKT+19] Jacob Holm, Valerie King, Mikkel Thorup, Or Zamir, and Uri Zwick. Random k-out subgraph leaves only $O(n/k)$ inter-component edges, 2019. To appear at FOCS'19.

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

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[JN18] Tomasz Jurdziński and Krzysztof Nowicki. MST in O(1) Rounds of Congested Clique. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 2620–2632, 2018.

[Kar93] David Karger. Global min-cuts in *RNC* and other ramifications of a simple mincut algorithm. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 21–30, 01 1993.

[Kar96] David R. Karger. Minimum cuts in near-linear time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 56–63, 1996.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000.

[KP09] David R. Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proc. 20th SODA*, pages 246–255, 2009.

[Kru56] Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[KS93] David R. Karger and Clifford Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 757–765, 1993.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 938–948, 2010.

[KT15] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 665–674, 2015.

[KT19] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019.

[LMSV11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 85–94, 2011.

[MK13] M. Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 1–15, 2013.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[Now18] Krzysztof Nowicki. Random sampling applied to the MST problem in the node congested clique model. *CoRR*, abs/1807.08738, 2018.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 439–453, 2014.

[NW61] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. of the London Math. Society*, 36:445–450, 1961.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PR99] Seth Pettie and Vijaya Ramachandran. A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. In Dorit S. Hochbaum, Klaus Jansen, José D. P. Rolim, and Alistair Sinclair, editors, *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, pages 233–244, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[PS16] Sriram V. Pemmaraju and Vivek B. Sardeshmukh. Super-Fast MST Algorithms in the Congested Clique Using o(m) Messages. In *FSTTCS 2016*, pages 47:1–47:15, 2016.

[Rab81] Michael O. Rabin. Fingerprinting by random polynomials. 1981.

[Tar75] R. E. Tarjan. Efficiency of a good but not linear set union algorithms. *J. ACM*, 22:215–225, 1975.

[Tut61] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. of the London Math. Society*, 36:221–230, 1961.

[Whi12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[WM93] David W. Matula. A linear time 2+epsilon approximation algorithm for edge connectivity. pages 500–504, 01 1993.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.