Universally-Optimal Distributed Exact Min-Cut

Mohsen Ghaffari
ghaffari@mit.edu
MIT
Cambridge, MA, USA

Goran Zuzic
goran.zuzic@inf.ethz.ch
ETH Zürich
Zürich, Switzerland

ABSTRACT

We present a universally-optimal distributed algorithm for the exact weighted min-cut. The algorithm is guaranteed to complete in $O(D + \sqrt{n})$ rounds on every graph, recovering the recent result of Dory, Efron, Mukhopadhyay, and Nanongkai [STOC’21], but runs much faster on structured graphs. Specifically, the algorithm completes in $O(D)$ rounds on (weighted) planar graphs or, more generally, any (weighted) excluded-minor family.

We obtain this result by designing an aggregation-based algorithm: each node receives only an aggregate of the messages sent to it. While somewhat restrictive, recent work shows any such black-box algorithm can be simulated on any minor of the communication network. Furthermore, we observe this also allows for the addition of a small number of arbitrarily-connected virtual nodes to the network. We leverage these capabilities to design a min-cut algorithm that is significantly simpler compared to prior distributed work. We hope this paper showcases how working within this paradigm yields simple-to-design and ultra-efficient distributed algorithms for global problems.

Our main technical contribution is a distributed algorithm that, given any tree $T$, computes the minimum cut that $2$-respects $T$ (i.e., cuts at most 2 edges of $T$) in universally near-optimal time. Moreover, our algorithm gives a deterministic $O(D)$-round 2-respecting cut solution for excluded-minor families and a deterministic $O(D + \sqrt{n})$-round solution for general graphs, the latter resolving a question of Dory, et al. [STOC’21]

CCS CONCEPTS

• Mathematics of computing → Graph theory; Graph algorithms;
• Computing methodologies → Distributed algorithms; • Theory of computation → Network flows.

KEYWORDS

exact minimum cut, distributed algorithms, message-passing algorithms, CONGEST model, minor-aggregation model, low-congestion shortcuts, universal optimality

ACM Reference Format:
Mohsen Ghaffari and Goran Zuzic. 2022. Universally-Optimal Distributed Exact Min-Cut. In Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC ’22), July 25–29, 2022, Salerno, Italy. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3519270.3538429

1 INTRODUCTION

Computing the minimum cut in a graph is one of the fundamental and well-studied graph problems. This problem asks for computing the smallest collection of edges, in terms of their number in unweighted graphs and in terms of the total sum of their weights in weighted graphs, whose removal would disconnect the graph. This notion captures important properties of the network such as its robustness to failure—e.g., how many link failures can the network withstand before it gets disconnected—or communication bottlenecks—e.g., the smallest capacity of links connecting one set of nodes to the rest of the network. Over the past decade, we have witnessed significant developments on this problem in the distributed computing setting. To review these results, let us first recall the message-passing model of distributed graph algorithms.

Model. As standard, we work with the standard message-passing model of distributed computing (known as CONGEST [29]). The network is abstracted as an $n$-node connected undirected graph $G = (V, E)$ where each node represents one of the computers in the network (i.e., has its own processor and private memory). Communication takes place in synchronous rounds and per round, each node can send one $O(\log n)$-bit message to each of its neighbors. The nodes do not know the topology of the network at the start of the algorithm (except for each knowing its own neighbors, and perhaps some estimates on the total number of nodes $n$ and the network diameter $D$). Initially, nodes only know their unique $O(\log n)$-bit ID and the IDs of adjacent nodes. In the end, each node should know its own part of the output, e.g., the size of the minimum cut and which adjacent edges are in the computed cut.

State of the art on distributed computation of min-cut. The initial progress on distributed algorithms on min-cut focused on approximations. Ghaffari and Kuhn [11] gave a randomized algorithm that computes a $2 + \varepsilon$ approximation, for an arbitrarily small positive constant $\varepsilon$, of minimum cut in $O(D + \sqrt{n})$ rounds for weighted graphs. They also showed, by a minor adaptation of the lower bound of Das Sarma et al. [4], that any non-trivial approximation of minimum cut in weighted graphs requires $\tilde{O}(D + \sqrt{n})$ rounds. For unweighted graphs, their lower bound degrades to $\tilde{O}(D + \sqrt{n/\lambda})$ where $\lambda$ denotes the minimum cut size. Nanongkai and Su [26] improved the approximation factor to a $1 + \varepsilon$ while maintaining the same $\tilde{O}(D + \sqrt{n})$ round complexity. Progress on exact computation was more scarce, until a result of Daga, Henzinger, Nanongkai, and Saranurak [3] that obtained the first sublinear-time algorithm for unweighted graphs. Concretely, their algorithm computes the exact minimum cut in $O(n^{1-1/335}D/353 + n^{1-1/706})$ rounds in unweighted graphs. Ghaffari, Nowicki, and Thorup [14] then provided
a different exact algorithm for unweighted graphs that improved the round complexity further to \(O((n^{0.8}D^{0.2} + n^{0.9}))\). Also, Parter [28] gave an algorithm with round complexity \((\Delta D)^{O(1)}\) for computing the exact unweighted min-cut, where \(\Delta\) denotes the min-cut size; this, in particular, runs in poly\((D)\) for unweighted graphs with constant min-cut size. Finally, in a recent breakthrough, Dory, Efron, Mukhopadhyay, and Nanongkai [5] presented an algorithm that achieves the worst-case optimal round complexity of \(O(D + \sqrt{n})\) for exact computation of minimum cut in unweighted graphs.

**Beyond worst-case.** When can we call a distributed algorithm “optimal” or “near-optimal” and what exactly do we mean by that? The \(O(D + \sqrt{n})\) complexity achieved above is near-optimal, in a worst-case sense, as follows: there is a weighted graph with diameter \(D = O(\log n)\) in which any min-cut algorithm would need \(\Omega(D + \sqrt{n})\) rounds. This optimality is stronger than another worst-case optimality, where we would consider \(O(n)\)-round algorithms near-optimal. Notice that the latter is also a correct statement, as there is a graph in which any algorithm needs \(\Omega(n)\) rounds (namely, a simple \(n\)-node cycle). However, the former gives a sharper bound for a wide range of graphs of interest, particularly, graphs where the diameter \(D\) is small. Is there an even stronger notion of optimality?

One could think about focusing on particular graph parameters that capture “usual” network graphs and aim for faster algorithms when these parameters are small. Even then, we are essentially justifying the performance of the algorithm on any network \(G\) of the family, because of the mere existence of one (concocted) network \(G'\) in the family where the algorithm cannot perform faster. Plausibly, in most usages of the algorithm, the network \(G\) is much more well-behaved than that tailored worst-case graph \(G'\), and thus we could desire much faster algorithms.

**Universal Optimality.** A far more ambitious goal is to seek universal optimality. That is, to seek a single (i.e., uniform) algorithm which, when run on any network \(G\), has the time-complexity that is competitive with the fastest (correct) algorithm’s round complexity on that particular network \(G\) itself. This paper’s objective is to develop such a universally near-optimal algorithm for exact computation of minimum cut. Toward this goal, let us briefly take a detour and recall the concept of low-congestion shortcuts.

**Detour to low-congestion shortcuts and shortcut quality.** Given a network \(G = (V, E)\), Ghaffari and Haeupler [9] defined the shortcut quality \(SQ(G)\) as the smallest value \(Q\) such that we have the following: for any (adversarial) partition of vertices \(V\) into disjoint parts \(V_1, V_2, \ldots, V_N\), each of which induces a connected subgraph \(G[V_i]\), there exists a collection of subgraphs \(H_1, H_2, \ldots, H_N\), such that (1) for each \(i \in [1, N]\) the diameter of \(G[V_i] \cup H_i\) is at most \(Q\), and (2) each edge \(e \in E\) appears in at most \(Q\) many of the subgraphs \(H_i\). The graph \(H_i\) is called the shortcut for part \(V_i\).

Ghaffari and Haeupler [9] showed that any \(D\)-diameter \(n\)-node graph admits a shortcut with quality \(O(D + \sqrt{n})\), and they showed algorithms with round complexity \(O(SQ(G))\) for exact computation of minimum spanning tree and \(1 + \epsilon\) approximation of minimum cut in weighted graphs. These assume that shortcuts can be computed in \(O(SQ(G))\), and otherwise, the construction time should be added to the complexity. This result immediately recovers the \(O(D + \sqrt{n})\) complexity of the minimum spanning tree and \(1 + \epsilon\) approximation of minimum cut in general weighted graphs with hop-diameter \(D\) and \(n\) nodes. But it also leads to significantly faster algorithms in more well-behaved graphs.

In particular, Ghaffari and Haeupler [9] showed that the shortcut quality \(SQ(G)\) is smaller for many other graph families. For instance, \(SQ(G) = O(D)\) for any planar graph or constant-genus family. This was later sharpened and extended for graphs with bounded genus, bounded treewidth, and bounded pathwidth [18]. Haeupler, Li, and Zuzic [19] gave shortcuts for excluded-minor graphs, with quality and construction-time \(O(D^2)\). Finally, Ghaffari and Haeupler [10] improved and strengthened all these results and showed that excluded-minor graphs, which contain all previously mentioned graphs, admit shortcuts with quality \(O(D)\). For all of the aforementioned results, it is known how to construct shortcuts of quality \(O(SQ(G))\) (with an efficient \(O(SQ(G))\)-round and deterministic distributed algorithm [10, 16, 17]). Hence, these imply an \(O(D)\)-round algorithm for \((1 + \epsilon)\)-approximation of weighted min-cut in any \(D\)-diameter excluded-minor graph network.

These results focus mostly on sparse graphs, in a vague sense. On the opposite side, for well-connected graphs, the results of Ghaffari, Kuhn, and Su [12], which were sharpened by Ghaffari and Li [13] showed that any graph with \(1/poly(\log n)\) mixing time admits a shortcut with quality \(poly(\log n)\) and one can compute shortcut of quality \(2^\Theta(\sqrt{\log n})\) in \(2^\Theta(\sqrt{\log n})\) rounds (indeed in any graph with mixing time \(2^{-\Theta(\sqrt{\log n})}\)). These imply an \(2^\Theta(\sqrt{\log n})\)-round algorithm for \((1 + \epsilon)\)-approximation of weighted min-cut in well-connected graphs, with mixing-time \(1/poly(\log n)\) or even \(2^{-\Theta(\sqrt{\log n})}\). This in particular includes Erdos-Renyi random graphs above the connectivity threshold.

**Back to Universal Optimality.** Haeupler, Wajc, and Zuzic [21] showed that the shortcut quality is not only an upper bound for the round complexity of computing a minimum spanning tree or approximation of min-cut, as shown in [9] but also a universal lower bound for it. That is, roughly speaking, for any network graph \(G\) with shortcut quality \(SQ(G)\), one can show that any distributed algorithm that works correctly on all graphs needs \(\tilde{\Omega}(SQ(G))\) rounds to solve the (both approximate or exact) minimum cut problem on the network \(G\) itself. Notice that in this statement, while the topology network \(G\) itself is fixed, and can be even known to all the nodes, the weights on the edges of \(G\) are the input to the problem.

In light of this, we can say that the \((1 + \epsilon)\)-min-cut approximation algorithm of Ghaffari and Haeupler [9], which runs in \(O(SQ(G))\) rounds once given the shortcuts, is a universally-optimal algorithm for \((1 + \epsilon)\)-approximation min-cut since any correct algorithm requires \(\tilde{\Omega}(SQ(G))\) rounds. This is modulo one small but important issue: the time for computing shortcuts has not been taken into account. However, arguably, that is an orthogonal topic within the ambitious path toward the holy grail of obtaining universally-optimal distributed algorithms for all global graph problems: we can separate the issue of efficient shortcut computation in different graphs from the issue of how to design algorithms for various problems whose complexity is proportional to the shortcut quality once efficient computation is assumed.

Only the latter part is within the scope of this paper. A universally-optimal min-cut algorithm that assumes efficient construction still
implies an unconditional universally-optimal algorithm when the network is guaranteed to not contain a fixed minor, and for the setting with known topology (where the weights are still unknown and a part of the input, also known as the supported CONGEST) as studied by Haeupler, Wajc, and Zuzic [21]. Furthermore, for the more standard unknown-topology setting, there has been significant recent progress on the former issue of fast construction of shortcuts; in particular, Haeupler et al. [20] showed that one can obtain shortcuts with quality $O(SQ(G))n^{o(1)}$ in the same number of rounds. As such, combining this with [9], we can now compute a $(1+\epsilon)$-approximation of min-cut in poly($SQ(G)$)$n^{o(1)}$ in any network $G$, and this is within a polynomial of the best-possible bound for the network $G$ itself, modulo an $n^{o(1)}$ factor. Moreover, any future $O(SQ(G))$-quality construction in $O(SQ(G))$ rounds would retroactively turn these conditional universally-optimal algorithms into unconditional ones.

The Minor-Aggregation model. State-of-the-art distributed algorithms have become increasingly more complex, due to the influx of new ideas and their increasing complexity. To address this issue, Zuzic et al. [31] introduced the Minor-Aggregation model: a simple and powerful interface for designing ultra-fast distributed algorithms in the standard message-passing (i.e., CONGEST) model. The interface provides high-level primitives that simplify algorithm design; the primitives are then, in turn, efficiently implemented in CONGEST using low-congestion shortcuts and the multitude of tools developed around them. For example, [31] used the interface to simplify the design of their universally-optimal $(1+\epsilon)$-approximate distributed shortest path algorithm. On a technical level, the Minor-Aggregation model restricts the algorithm to operate only on aggregates: each node receives only an aggregate value (e.g., sum, max, logical-OR, etc.) of all messages sent to it. However, this restriction, combined with low-congestion shortcuts, enables efficient edge contractions which are difficult to efficiently implement in a distributed setting. In other words, this allows a black-box algorithm to be run on an arbitrary minor. As an instructive example, consider the classic Boruvka’s MST algorithm which works by computing the minimum-weight outgoing edge from each node and then contracting all such edges; this iteration is repeated for $O(\log n)$ steps until the graph is trivial. Boruvka’s algorithm naturally operates on aggregates, hence it can be immediately performed on the minor resulting from contracting minimum-weighted edges, giving us an $O(\log n)$-round Minor-Aggregation algorithm. Using prior work, this can be turned into, say, an $O(D)$-round algorithm for (weighted) planar networks. More generally, a $t_1$-round Minor-Aggregation algorithm can be turned into an $O(t_1 \cdot t_2)$-round CONGEST algorithm, where $t_2$ is the time to construct shortcuts of quality $t_2$ (see below for a list of implications).

Our contributions. Our first contribution of this paper is to develop a poly($\log n$)-round Minor-Aggregation algorithm that computes the exact minimum cut in weighted graphs. This unconditionally recovers the breakthrough $O(D + \sqrt{n})$-round CONGEST algorithm for general graphs of Dory et al. [5]. Moreover, it unconditionally implies the following set of novel results.

**Theorem 1.1.** Suppose $G$ is an $n$-node graph with hop-diameter $D$. There are randomized distributed CONGEST algorithms $A_1$, $A_2$, $A_3$, $A_4$ over $G$ for the exact weighted min-cut problem with the following guarantees:

- When $G$ is an excluded-minor graph (e.g., a planar network), $A_1$ terminates in universally-optimal $O(D)$ rounds.
- When the graph topology $G$ is known to all nodes, $A_2$ terminates in universally-optimal $O(SQ(G))$ rounds. (Note: this bullet, along with known implications, implies all other bullets.)
- When $G$ well-connected graph with mixing-time $2^{O(\sqrt{\log n})}$, $A_3$ terminates in almost-universally-optimal $2^{O(\sqrt{\log n})}$ rounds.
- When $SQ(G) \leq n^{o(1)}$, $A_4$ terminates in (almost-universally-optimal) $n^{o(1)}$ rounds.

Our second contribution is a simple but powerful extension of the Minor-Aggregation model. Specifically, we show that any black-box Minor-Aggregation algorithm can be logically executed on a network graph $G$ joined with (a small number of) arbitrarily-connected virtual nodes which do not need to exist in $G$; this is compiled down to an algorithm that only communicates using the existing links in the underlying network graph $G$ while suffering only a small overhead. Any such property fails for general CONGEST (i.e., non-aggregation based) algorithms, as adding a single fully-connected virtual node greatly increases the computational power of the model. Virtual nodes allow us to import various techniques like divide-and-conquer from the centralized and parallel settings into the distributed world, which is the reason why this paper can simplify and speed up the arguably-complicated exact min-cut algorithm of [5]. Moreover, our virtual-node extension has already found prolific use in Rozhon et al. [30], which gives a unified algorithm for the deterministic $(1+\epsilon)$-shortest path that is both the first near-optimal in the parallel setting and the first universally optimal in the distributed setting.

Our third contribution is a deterministic Minor-Aggregation algorithm for the 2-respecting min-cut problem, which often implies a deterministic CONGEST algorithm. To give context, our exact min-cut algorithm follows the strategy outlined by Karger [22]—and frequently used later, e.g., [3, 5]—which is comprised of exactly two self-contained pieces: the tree packing and the 2-respecting min-cut. The former piece, tree packing, is about finding a collection of poly($\log n$) spanning trees such that every min-cut 2-respects one tree $T$ in the collection, in the sense that the cut includes at most 2 edges of $T$. The latter piece, 2-respecting min-cut, is when we are given a tree $T$ and we should compute the minimum cut in graph $G$ among those that $2$-respect $T$. We note that obtaining “efficient” deterministic tree packing is still an active area of research even in the centralized setting (with some exciting recent progress by Li [23]). In contrast, computing 2-respecting min-cut has been successfully derandomized in the centralized [8] and parallel settings [24]. We contribute the analogous distributed result and obtain a deterministic CONGEST 2-respecting min-cut that terminates in $O(D)$ rounds for weighted excluded-minors graphs (e.g., weighted planar graphs), and $O(D + \sqrt{n})$ rounds for general graphs. The latter result resolves an open question of [5], who asked for a $O(D + \sqrt{n})$-round deterministic algorithm for this 2-respecting min-cut problem. To achieve this result, we contribute to the low-congestion shortcut ecosystem of tools by derandomizing several important primitives.
like heavy-light decompositions, subtree sums, and ancestor sums of trees.

Other related work on exact min-cut. Algorithms for min-cut have seen a flurry of recent progress. Mukhopadhyay and Nanongkai [25] observed several structural properties of min-cut that enable the min-cut to be computed more efficiently and in different models as compared to the celebrated near-linear-time algorithm of Karger [22]. Subsequent result include work-optimal parallel algorithms for non-sparse graphs [24], near-existentially-optimal distributed algorithms [5], faster directed algorithm for directed min-cut [1], etc. At the same time and independently of [25], Gawrychowski, Mozes, and Oren [7] proposed an algorithm for the 2-respecting min-cut in the centralized setting that is deterministic and faster than that of Karger [22]. Building on top of concurrent work, they ultimately also proposed a centralized $O(m \log^2 n + n \log^2 n)$-time algorithm [8].

2 AN OVERVIEW OF OUR METHODS

We give an extensive overview of our methods which could be sufficient to reconstruct the major parts of our approach.

Minor-Aggregation with virtual nodes. We start by giving a short and informal preliminary on the Minor-Aggregation model, as introduced in [31] (see the Preliminaries section in the full version for a formal discussion). A distributed algorithm in this model performs computations in synchronous rounds. In each round, the algorithm first contracts an arbitrary subset of edges. Then, each super-node (which is created from contracting a connected component of the contracted edges) sends a message to all of its neighbors. On the receiving end, each node $v$, instead of receiving each of the messages $m_1, \ldots, m_k$ sent to it, receives only an aggregate value $\bigoplus_{i=1}^k m_i$, where $\bigoplus$ is some aggregate function like the sum or the max, but can also be as complicated as an arbitrary mergeable sketch. Several observations are immediate: we can run black-box algorithms on minors (due to contractions), and we can run simultaneous algorithms on node-disjoint connected subgraphs (we add the warning that edge-disjointness would not suffice). The goal is to find a poly($\log n$)-round min-cut algorithm in this model, which corresponds to a universally-optimal algorithm (under certain conditions orthogonal to this paper). We contribute to the model in the following ways:

Virtual nodes. (Section 4.1) We observe the simple but powerful property that aggregation-based algorithms behave remarkably well under the addition of virtual nodes. Specifically, we allow to add poly($\log n$) virtual nodes to the underlying network and arbitrarily connect them with virtual edges, either among themselves or between virtual nodes and nodes of $G$. Any algorithm on the resulting virtual graph can be simulated on $G$ with a poly($\log n$) multiplicative blowup in the number of rounds. Note that no such property exists for CONGEST without introducing polynomial blowup factors in the computation.

Modeling power and caveats when using virtual nodes. This possibility of adding virtual nodes greatly enhances the modeling power of the Minor-Aggregation model. For example, one can turn a black-box single-source shortest path algorithm in the Minor-Aggregation model into a multi-source shortest path algorithm by creating a virtual super-source and connecting it to a set of source nodes. Moreover, when combined with recursions, virtual nodes can be used to bring many recursive graph algorithms from the centralized and parallel settings into the distributed world. To see why, when doing a recursive call, one often needs to change the subgraphs before passing them to the recursive calls. Virtual nodes provide a very simple way of achieving this. However, we should add a warning about an issue we call simulation cascade that can arise when combining virtual nodes and recursions: when issuing a recursive call on a virtual graph, that call has to eventually run on the underlying communication network. A naive solution would be to simply remove the virtual nodes from the recursive call via simulation, thereby causing a (small) multiplicative blowup. However, this multiplicative blowup happens on every level of the recursion, preventing the final algorithm from having a polylogarithmic running time. In this paper, we develop several different solutions for this issue (explained later).

Deterministic primitives. (Section 4.2) Important primitives like heavy-light decompositions, ancestor sums, and subtree sums of trees are often ubiquitously-used primitives within the low-congestion shortcut framework. Within the Minor-Aggregation model, consider combining subtree sums with the approximate heavy-slicer sketch (which is a mergeable sketch, hence is a valid aggregation operator): given inputs $x_u$ for each node $u$, each node $v$ can compute the heavy hitters among $\{x_u : u \text{ is in the subtree of } v\}$. However, prior work has typically resorted to a randomized implementation of these primitives [6, 15, 21]. We address this issue by providing deterministic poly($\log n$)-round Minor-Aggregation algorithms for all the aforementioned primitives. This yields fully deterministic $O(D)$-round CONGEST algorithms for these primitives in excluded-minor networks, and $O(D + \sqrt{n})$-round CONGEST algorithms for general graphs. We achieve this result by replacing the randomized star-merging technique used throughout the low-congestion shortcut framework with a deterministic version by leveraging the deterministic 3-coloring of out-degree-one graphs developed by Cole and Vishkin [2].

Minimum cut via tree packing, and 2-respecting min-cuts. To solve the minimum cut problem, thanks to the known tree packing results [3, 5, 22], it suffices to find the minimum cut among the cuts that 2-respects a given tree. We note that the algorithm from prior work for this tree-packing part easily extends to our setting. Our focus will be on computing the minimum 2-respecting cut, for a given tree $T$. That is, given a fixed spanning tree $T$ of a graph $G$, compute the minimum cut in $G$ among those that cut at most 2 edges of $T$.

To treat the minimum 2-respecting cut problem for the given tree $T$, we break it into simpler special cases. The general algorithm will be achieved by a clean and modular combination of these cases.

Path-to-path 2-respecting min-cut. (Section 6) First, we consider an important sub-case of the path-to-path 2-respecting min-cut: the case when the tree $T \subseteq G$ happens to be composed of exactly two paths $P$ and $Q$ along with a common root connecting them (see Figure 1). Our goal is to find the min Cut($e, f$) over all pairs $e, f \in E(P) \times E(Q)$ (i.e., the edges are on different paths), where
Cut$(e, f)$ is the sum of weights of edges of $G$ which cross the cut determined by $(e, f)$ (i.e., all edges with endpoints $u, v$ such that the unique $T$-path between $u, v$ crosses exactly one of $(e, f)$). Several notable ideas go into designing an algorithm for this problem:

**General recursive idea and the Monge property.** (Observed by Mukhopadhyay and Nanongkai [25].) The main idea is to import the state-of-the-art centralized techniques into the distributed setting with the help of the Minor-Aggregation model. Specifically, we first fix $e_a$ to be the midpoint edge of $P$ (i.e., $a := \lfloor |P|/2 \rfloor$) and let $f_b$ be the best response to $e_a$, meaning the edge $f_b \in E(Q)$ that minimizes $\text{Cut}(e_a, f_b)$. Then, either $(e_a, f_b)$ is the pair that minimizes the 2-respecting cut, or the minimizing pair can be found on either $P_{\text{up}} := \{e_1, \ldots, e_{a-1}\} \times Q_{\text{up}} := \{f_1, \ldots, f_{b-1}\}$ ($e_1, f_1$ are connected to the root) or $P_{\text{down}} := \{e_{a+1}, \ldots, e_{|P|}\} \times Q_{\text{down}} := \{f_{b+1}, \ldots, f_{|Q|}\}$. This property, i.e., that the minimizing pair is either completely on one side or completely on the other side of $(e_a, f_b)$, is the so-called Monge property. Due to this property, we can issue two simultaneous recursive calls on $P_{\text{up}} \times Q_{\text{up}}$ and $P_{\text{down}} \times Q_{\text{down}}$ and return the best result found. Note that a parallel implementation of this idea has $O(1)$ recursion depth and can be implemented in near-linear work.

**Private cut-equivalent graphs.** One issue afflicting the above idea in the distributed setting is that the recursive call on, say, $P_{\text{up}} \times Q_{\text{up}}$ requires information private to $P_{\text{down}} \times Q_{\text{down}}$: an edge strictly between the latter affects the answer of the former. To prevent this, we construct private and cut-equivalent graphs $G_{\text{up}}$ and $G_{\text{down}}$ that are (1) private, in the sense that the recursions can freely use them as well as guaranteeing that (2) Cut$(e \in E(P_{\text{up}}), f \in E(Q_{\text{up}}))$ is the same with respect to $G$ and $G_{\text{up}}$. This is achieved by replacing the top-most and bottom-most edges of $P_{\text{up}}$ and $Q_{\text{up}}$ with virtual nodes (as well as the root), which are both private to the recursion and allow us to insert additional edges to achieve cut equivalency. For example, an edge $\{a \in V(P_{\text{down}}), b \in V(Q_{\text{up}})\}$ is replaced in $G_{\text{up}}$ with an edge between the bottom (virtual) node of $P_{\text{down}}$ and $b$, making it private and making its contribution to all 2-respecting cuts equivalent in the recursive call on $G_{\text{up}}$ as it would have been if considering $G$. Other types of edges and the recursive call on $G_{\text{down}}$ are analogous.

**Avoiding simulation cascade (using separability).** Another issue with the above idea of private-but-virtual graphs is that each recursive call is performed on a virtual graph (albeit, with a small number of virtual nodes). This has to be ultimately converted to an algorithm without virtual nodes. For instance, one idea is to naively call the recursive algorithm on, say, the virtual graph $G_{\text{up}}$ and then remove the virtual nodes using simulation (which introduces a small multiplicative overhead). However, this would yield a runtime explosion as every level of the recursion would introduce a cascading multiplicative overhead to the computation, making the final runtime polynomial (the desired runtime is polylogarithmic). The solution, however, might seem simple but is essential. Consider, say, the sub-instance $P_{\text{up}} \times Q_{\text{up}}$ on $G_{\text{up}}$. We want to remove the virtual nodes before the recursive call returns so that the returned call only performs work on the underlying graph, removing any need for cascading simulation of virtual nodes. This “de-virtualization,” however, can only be performed in $G_{\text{up}}$ if $G_{\text{up}} = \text{Virt}$ (minus its virtual nodes) is connected. If this is the case, we can resolve the issue as explained. If it is not connected, however, this forces a trivial structure called separability on the sub-instance which can be solved without recursing. Specifically, we show that Cut$(e, f)$ can be separated, i.e., written as Cut$(e, f) = F_P(e) + F_Q(f)$ for some functions $F_P, F_Q$. In this case, separate minimizations of both sides lead to the correct result.

**Star 2-respecting min-cut.** (Section 7) Next, we use the path-to-path algorithm to build an algorithm in which the tree $T \subseteq G$ is exactly composed of $k$ paths $P_1, \ldots, P_k$ and a common root that connects to the top of each path (see Figure 2). The goal is to find the minimum 2-respecting cut Cut$(e, f)$ where $e \in E(P_i)$ and $f \in E(P_j)$ are two edges on different paths $i \neq j$. Several notable ideas go into designing an algorithm for this problem:

**Path interest.** (Introduced by Mukhopadhyay and Nanongkai [25].) We say a non-tree edge $\{u, v\} \in E(G)$ covers a tree-edge $e$ if the unique path in $T$ between $u$ and $v$ contains $e$. We say that a path $P_i$ is interested in a path $P_j$ if there exist edges $e \in E(P_i)$, $f \in E(P_j)$ such that at least half of the edges covering $e$ also cover both $e$ and $f$ (counting weight as the multiplicity). If the pair of edges that determine the optimum 2-respecting min-cut lie on paths $P_i$ and $P_j$, then $P_i$ and $P_j$ must be mutually interested in each other. Therefore, the general idea for the star algorithm will be to examine all mutually-interested pairs of paths using the path-to-path oracle. An important property that enables solving the star instance is that each path is interested in at most $O(\log n)$ other paths.

**Interest lists and cross-edges.** We now describe how to efficiently compute for each path $P_i$ a list of paths that $P_i$ is interested in. On a technical level, each path-edge $e \in E(P_i)$ needs to find the set of edges $f \in E(P_j)$ such that the majority of non-tree edges covering $e$ also cover $f$. It is immediate that, for each fixed $e$, all edges $f$
(it any) lie on a single path $P_j$ in which case $P_i$ is interested in $P_j$. For a fixed edge $e \in E(P_i)$, this corresponds to picking a majority element of a sequence, where each (non-tree) edge $f \in E(G)$ between $P_i$ and $P_j$ contributes $\omega(f)$ weight to $P_j$. This majority operation, however, can be performed using deterministic heavy-hitter sketches, which gracefully fit within the framework of aggregation operations. Therefore, we can use the newly developed deterministic subtree sum operation with the heavy-hitter aggregator to find the majority element for each edge $e$, indicating path interest. Furthermore, as $e \in E(P_i)$ is “moved across” the path $P_i$, there can be at most $O(\log n)$ other paths that $P_j$ is interested in. Therefore, we find the union of all found (almost) majority elements in each path, as there can be at most $\tilde{O}(1)$ of them.

However, there is an issue plaguing this approach: if one simply considers all edges $f \in E(G)$ covering a path-edge $e \in E(P_i)$ and is looking for the majority element using the subtree sum operation, they would also need to support the “remove” operation in the heavy-hitter sketch since some edges considered throughout the subtree of a node $v$ should not be considered at its parent node. However, the heavy-hitter sketch does not support this. We get around this by slightly changing the definition of interest to only consider cross-edges (edges going from one path to another), which do not require removals. We show that all the important results hold even if one ignores all other types of edges.

**Interest graph.** Consider the logical graph where each node represents a different path $P_i$ and there is an edge $(P_i, P_j)$ if and only if $P_i$ and $P_j$ are mutually interested in each other. Moreover, we can simulate an arbitrary Minor-Aggregation algorithm on the interest graph by contracting away all path edges since any two mutually-interested paths must have an edge between them. Moreover, since each path is interested in at most $O(\log n)$ other paths, the maximum degree of the interest graph is $\Delta = O(\log n)$. This implies that we can also simulate arbitrary CONGEST algorithms on the interest graph (i.e., non-aggregation based) with a multiplicative $O(\Delta) = \tilde{O}(1)$ blowup.

**Edge coloring of the interested graph.** We find the smallest 2-respecting cut among all pairs of mutually-interest paths by first computing an edge coloring of the interest graph. To this end, we can simulate the deterministic CONGEST algorithm of Panconesi and Rizzi [27] on the interest graph that colors the interest graph into $O(\Delta) = \tilde{O}(1)$ colors. Then, we iteratively consider each color class in isolation. Within each class, all pairs of matched paths are node disjoint, hence we can use the previously-developed path-to-path 2-respecting min-cut algorithm to find the optimum solution.

**Between-subtree 2-respecting min-cut.** (Section 8) We now use the star algorithm to build a between-subtree 2-respecting cut algorithm, in which the tree $T \subseteq G$ is exactly composed of $k$ subtrees $T_1, \ldots, T_k$ and a common root that connects to the top of each subtree (see Figure 3). The goal is to find the minimum 2-respecting cut $\text{Cut}(e, f)$ where $e$ and $f$ are two edges in different subtrees. Several notable ideas go into designing an algorithm for this problem:

**Pairwise coloring.** Our first idea is to reduce the problem for general $k$ to the case when $k = 2$. Suppose the optimum 2-respecting cut $(e^*, f^*)$ is contained in subtrees $e^* \in E(T_j)$ and $f^* \in E(T_j^*)$. We will construct a pairwise coloring $\{f_1, \ldots, f_k\}$, i.e., a small collection of color assignments $f_i : [k] \rightarrow \{\text{red}, \text{blue}\}$ such that each pair of subtrees $T_i, T_j$ is assigned a different color in at least one color assignment. It is a folklore result that there exists such an assignment with $\chi = O(\log n)$ (e.g., consider the $O(\log n)$ different bits of the subtree IDs). After constructing such a collection of colorings, we iterate over each color assignment, and for each assignment, merge all the roots of all subtrees colored red and all subtrees colored blue. This reduces the problem to the $k = 2$ case.

**Heavy-light decomposition.** We now reduce the $k = 2$ problem to the (solved) star case. First, we construct a heavy-light decomposition of both subtrees, in which each edge is assigned a label “heavy” or “light” such that each root-to-leaf path has at most $O(\log n)$ light edges. We define an HL-depth of an edge $e$ to be the number of light edges on the root-to-$e$ path. Now, suppose the optimum 2-respecting cut $(e^*, f^*)$ has $d_1^* = \text{HL-depth}(e^*)$ and $d_2^* = \text{HL-depth}(f^*)$. Since $d_1, d_2 = O(\log n)$, we guess the correct $d_1^*$ and $d_2^*$ by testing all possible combinations. For each guess, contract all edges $e$ in $T_j$ with HL-depth$(e) \neq d_1^*$ and all edges $e$ in $T_i$ with HL-depth$(e) \neq d_2^*$. This reduces the question to exactly the star case (see Figure 4).

**Final step: 2-respecting general cut.** (Section 9) Finally, we solve the general 2-respecting min-cut, in which we are given a spanning tree $T$ of a weighted graph $G$ and the goal is to find $\min_{e \in E(T)} \text{Cut}(e, f)$. Several notable ideas go into designing an algorithm for this problem:

**The general recursive idea and the centroid decomposition.** It is a well-known folklore result that each tree $T$ has a centroid node $c \in V(T)$ such that all connected components of $T - c$ have at most $|V(T)|/2$ nodes. We will solve the general case by first finding the centroid $c$ of our tree $T$, which can be performed using the subtree
sum operation. Now, we denote the pair of edges defining some 2-respecting min-cut by \((e^*, f^*) \in E(T) \times E(T)\), and suppose we denote the maximal connected subtrees of \(T - e\) by \(T_1, \ldots, T_k\). Then, the pair \((e^*, f^*)\) can either be (1) in two different subtrees \(T_i, T_j\), or (2) in the same subtree \(T_2\). For case (1) we simply need to call the 2-respecting between-subtree cut algorithm on \(T_1, \ldots, T_k\); for case (2) we will use recursion on each one of the (node disjoint) subtrees \(T_i\), allowing us to schedule all recursive calls simultaneously.

Cut-equivalent subtrees via virtual nodes. One immediate issue breaking a naive implementation of the above recursive idea is that each recursive call on, say, \(T_i\) needs to have a private copy of edges \(E(G)\) which are used to calculate the values of (2-respecting) cuts. The issue seems essential: cut values completely within \(\mathcal{V}_{\text{virt}}\) can either be (1) in two different subtrees \(T_i, T_j\), or (2) in the same subtree \(T_2\). For case (1) we simply need to call the 2-respecting between-subtree cut algorithm on \(T_1, \ldots, T_k\); for case (2) we will use recursion on each one of the (node disjoint) subtrees \(T_i\), allowing us to schedule all recursive calls simultaneously.

Avoiding simulation cascade. Naively recursing on each \(T_i + c_i\) and eliminating the virtual node by simulation leads to a simulation cascade, preventing us from achieving the desired runtime. However, we can mitigate this in the following way. Consider some particular recursive call, which happens to be run on some tree \(T\). In parent recursive calls several virtual nodes \(\mathcal{V}_{\text{virt}} \subseteq V(T)\) were introduced, one per level of the recursion. However, our algorithm has the immediate property that \(T - \mathcal{V}_{\text{virt}}\) (all virtual nodes and their adjacent edges removed) is connected. Furthermore, due to the choice of the centroid as the pivoting node, the depth of the recursion is \(O(\log n)\), giving us a bound that \(|\mathcal{V}_{\text{virt}}| \leq O(\log n)\). Therefore, inside the recursive call before returning, we eliminate all the virtual nodes \(\mathcal{V}_{\text{virt}}\) by simulating the algorithm on \(T - \mathcal{V}_{\text{virt}}\), which is a connected subgraph of the underlying communication network. Hence, no additional (or cascading) virtual node elimination needs to happen after the recursive call returns.

3 FULL TECHNICAL DETAILS

Due to space constraints, we set aside the full technical details of the paper (including the technical preliminaries) to the full version, available on arXiv.

4 EXTENDING THE MINOR-AGGREGATION MODEL

In this section, we extend the Minor-Aggregation model in two ways. First, we show how to do computations when virtual nodes and edges are added to the topology. Second, we show how to derandomize ubiquitous primitives like heavy-light decompositions and subtree sums.

4.1 Virtual nodes

Virtual graphs allow us to create a logical network \(G_{\text{virt}}\) which can be implemented by only performing communication/computations on some underlying graph \(G \subseteq G_{\text{virt}}\). For example, we might want to add a virtual node \(v_{\text{virt}}\) (and its neighboring edges) to a graph \(G\) and simulate computation in \(G + v_{\text{virt}}\) without actually having \(v_{\text{virt}}\) in the underlying network graph itself. Naturally, this simulation will inherently have some overhead; in this section, we show that the (multiplicative) overhead of adding \(\beta \geq 1\) virtual nodes and arbitrarily connecting them (to the rest of the graph or among themselves) is only \(O(\beta)\).

Definition 4.1. A virtual graph \(G_{\text{virt}}\) extending \(G = (V, E)\) is a graph whose node set can be partitioned into \(V\) and a set of so-called virtual nodes \(V_{\text{virt}}\), i.e., \(V(G_{\text{virt}}) = V \cup V_{\text{virt}}\). We say \(G_{\text{virt}}\) has at most \(\beta\) virtual nodes (as an extension of \(G\)) if \(|V_{\text{virt}}| \leq \beta\). Furthermore, each edge of \(E(G_{\text{virt}})\) adjacent to at least one virtual node is called a virtual edge.

Distributed storage of virtual graphs. A virtual graph \(G_{\text{virt}}\) (extending \(G\)) is distributed stored in \(G\) in the following way. All nodes are required to know the list of all (IDs of) virtual nodes. A virtual edge connecting a non-virtual \(u\) and a virtual \(v\) is only stored in \(u\) (other nodes do not need to know about its existence). A virtual edge between two virtual nodes is required to be known by all nodes.

Simulations on virtual graphs. In a nutshell, we can add many virtual (arbitrarily interconnected) nodes to any graph \(G\) and still simulate any Minor-Aggregation algorithm on the virtual graph with a \(O(\beta + 1)\) blowup in the number of rounds. The proof is deferred to the full version (available on arXiv).

Theorem 4.2. Suppose \(A_{\text{virt}}\) is a (deterministic) \(\tau\)-round Minor-Aggregation algorithm on a virtual graph \(G_{\text{virt}}\) extending \(G\), where \(G\) is connected and the extension has at most \(\beta\) virtual nodes. Any such \(A_{\text{virt}}\) in \(G_{\text{virt}}\) can be simulated with a (deterministic) \(\tau \cdot O(\beta + 1)\)-round Minor-Aggregation algorithm in \(G\). Upon termination, each non-virtual node \(v \in V(G)\) learns all information learned by \(v\) and all virtual nodes.

We now show a useful lemma stating that we can always replace a node with its virtual substitute, which can be useful since we can
arbitrarily interconnect them to other (even non-virtual) nodes in $G$. The proof is deferred to the full version.

**Lemma 4.3.** Let $v \in V(G)$ be a node in $G$. In $O(1)$ deterministic Minor-Aggregation rounds, we can distributely store a graph $G_{\text{virt}}$, where the node $v$ is replaced with a virtual node $v_{\text{virt}}$ such that $G_{\text{virt}}$ is a virtual graph extending $G$ with a single virtual node. Specifically, $v$ in $G$ and $v_{\text{virt}}$ in $G_{\text{virt}}$ have the same set of neighbors. If multiple edges connect $v$ with some neighbor, $G_{\text{virt}}$ will contain a single edge with the weight equal to the sum of such edges in $G$.

### 4.2 Deterministic primitives and simulation

In this section, we state several useful deterministic primitives. The proofs are fairly involved as they need to argue about low-level model-specific details and are deferred to the full version.

**Lemma 4.4 (Deterministic primitives).** Let $T$ be a tree and let $r \in V(T)$. Suppose each node $v$ has an $O(1)$-bit private input $x_v$. There is a deterministic $O(1)$-round Minor-Aggregation algorithm computing the following for each node $v$:

- **Heavy-light decomposition:** $v$ learns its HL-info of the heavy-light decomposition rooted at $r$.
- **Ancestor sum:** $v$ learns $p_v := \sum_{w \in \text{anc}(v)} x_w$ where $\text{anc}(v)$ is the set of ancestors of $v$ w.r.t. root $r$.
- **Subtree sum:** $v$ learns $s_v := \sum_{w \in \text{desc}(v)} x_w$ where $\text{desc}(v)$ is the set of descendants of $v$ w.r.t. root $r$.

We now state how to simulate (deterministic) Minor-Aggregation algorithms in (deterministic) CONGEST. The proof is deferred to the full version.

**Theorem 4.5.** Suppose $A$ is any deterministic $\tau$-round Minor-Aggregation algorithm and suppose $G$ is an $n$-node graph with diameter $D$. We can simulate $A$ with a CONGEST algorithm on $G$ with the following guarantees:

- Unconditionally, the simulation requires $\tau \cdot O(D + \sqrt{n})$ rounds and is deterministic. [9]
- Unconditionally, the simulation requires $\tau \cdot \text{poly}(\text{SQ}(G)) \cdot n^{O(1)}$ randomized rounds. [20]
- When $G$ is a excluded-minor graph (e.g., planar graph), the simulation requires $\tau \cdot O(D)$ rounds and is deterministic. [10, 17]
- When the topology $G$ is known, the simulation requires randomized $\tau \cdot O(\text{SQ}(G))$ rounds. [21]

### 5 WARM-UP: 1-RESPECTING MIN-CUT

In this section we show how to compute all 1-respecting cuts when given a spanning tree $T$ of a graph $G$ with a poly$(\log n)$-round Minor-Aggregation on $G$. The proof is deferred to the full version.

**Theorem 5.1.** Let $T = (V, E_T)$ be a rooted spanning tree of a weighted graph $G = (V, E_G)$. There exists a deterministic $O(1)$-round Minor-Aggregation algorithm that computes $\text{Cut}_{T,G}(e)$ for all $e \in E_G$ (each edge learns its cut value).

### 6 2-RESPECTING PATH-TO-PATH MIN-CUT

In this section we show how to compute the minimum 2-respecting cut between two paths $P$ and $Q$ (adjointed with a root for orientation purposes, see Figure 1). We formalize this notion in the following result, which is the main result of this section.

**Theorem 6.1.** Suppose $G$ is a weighted graph and $T \subseteq G$ is $G$’s (rooted) spanning tree. Moreover, $T$ is composed of exactly a root $r$, and two descending paths $P, Q$. There exists a deterministic $O(1)$-round Minor-Aggregation algorithm on $G$ that computes the minimum of 1-respecting $\min_{e \in E(P) \cup E(Q)} \text{Cut}_{T,G}(e)$ and 2-respecting cuts $\min_{e \in E(P), f \in E(Q)} \text{Cut}_{T,G}(e, f)$.

We number the edges of $P$ as $e_1, e_2, \ldots , e_{|P|}$ in order of increasing depth (with $|P|$ denoting the length of the path). Similarly, we let $f_1, \ldots , f_{|Q|}$ be the edges of $Q$.

The main idea is to import the state-of-the-art centralized techniques into the distributed setting with the help of the Minor-Aggregation model. Specifically, we first fix $e_0$ as the midpoint edge of $P$ (i.e., $a := \lfloor |P|/2 \rfloor$) and let $f_0$ be the best response to $e_0$, meaning the edge $f_0 \in E(Q)$ that minimizes $\text{Cut}(e_0, f_0)$. Then, either $(e_0, f_0)$ are the pair that minimizes the 2-respecting cut, or the minimizing pair can be found on either $(e_1, \ldots , e_{a-1}) \times \{f_0, \ldots , f_{a-1}\}$ or $(e_{a+1}, \ldots , e_{|P|}) \times \{f_{a+1}, \ldots , f_{|Q|}\}$ (i.e., the minimizing pair is either completely on one side or completely on the other side of $(e_a, f_0)$). This last property easily follows from the following so-called Monge property and was recently observed by [25].

**Lemma 6.2 (Claim 7.1 in the full version of [5]; [8]).** For all $i \leq i', j \leq j'$ we have:

$$\text{Cut}_{T,G}(e_i, f_j) + \text{Cut}_{T,G}(e_{i'}, f_{j'}) \leq \text{Cut}_{T,G}(e_i, f_{j'}) + \text{Cut}_{T,G}(e_{i'}, f_j).$$

**Notation.** We introduce some (section-specific) notation. An edge $[u, v] \in E_G$ is said to be cross-path if it has one endpoints on both $V(P)$ and $V(Q)$; otherwise it’s said to be a same-path edge. Given a graph $G$ and a set of nodes $D \subseteq V(G)$, we denote by $G - D$ the subgraph with all nodes in $D$ removed. Furthermore, given an path-to-path instance $T \subseteq G$, we say the instance is separable if $G - (r, \text{top}(P), \text{bottom}(P), \text{top}(Q), \text{bottom}(Q))$ has no cross-path edges, where $\text{top}(\cdot)$ and $\text{bottom}(\cdot)$ of a rooted path $X$ represent the closest- and furthest-away nodes on $X$ from the root. It is important to observe that the instance is not separable if and only if $G - (r, \text{top}(P), \text{bottom}(P), \text{top}(Q), \text{bottom}(Q))$ is connected (or the paths have less than 3 nodes).

In order to calculate the best response of an edge (i.e., given $e \in E(P)$, calculate $\min_{f \in E(Q)} \text{Cut}(e, f)$), we use the following result.

**Lemma 6.3.** Assume the setting of Theorem 6.1 and let $e_{\text{fix}} \in E(P)$ be a fixed edge. There is a deterministic algorithm where each edge $f \in E(Q)$ learns $\text{Cov}(e_{\text{fix}}, f)$ that runs in $O(1)$-round Minor-Aggregation algorithm on $G$.

**Proof.** First, we compute depth for each node on $P, Q$ using a single subtree-sum operation (initialize all private values to 1 and use Lemma 4.4’s subtree sum with the + aggregation on $T$). Then, for each cross-path edge $[u, v], e \in E(G)$ with $u \in V(P), v \in V(Q)$ we perform the following. If $u$ is below bottom($e_{\text{fix}}$) (specifically, depth($v$) $\geq$ depth(bottom($e_{\text{fix}}$))), we add $+w(e)$ to the label of $v$. Note that this operation can be performed in a single Minor-Aggregation round. Finally, the subtree labels at a node $v$
represents the \(\text{Cov}_{T,G}(e_{\text{fix}}, \{v, \text{parent}(v)\})\). Therefore, we compute the subtree sum (Lemma 4.4) and obtain the required result in \(\tilde{O}(1)\) Minor-Aggregation rounds.

Next, we develop an algorithm that solves separable instances without any recursive calls.

**Lemma 6.4.** Assume the setting of Theorem 6.1 and suppose that \(G = \{r, \text{top}(P), \text{bottom}(P), \text{top}(Q), \text{bottom}(Q)\}\) has no cross-path edges. There exists a deterministic \(\tilde{O}(1)\)-round Minor-Aggregation algorithm on \(G\) that computes the minimum 2-respecting cut

\[
\min_{e \in E(P), f \in E(Q)} \text{Cut}_{T,G}(e, f).
\]

**Proof.** We show that, since the instance is separable, our function is separable in the following sense: there exist two functions \(F_P : E(P) \to \mathbb{R}\) and \(F_Q : E(Q) \to \mathbb{R}\) such that \(\text{Cut}(e, f) = F_P(e) + F_Q(f)\) for all \(e, f\). Due to \(\text{Cut}(e, f) = \text{Cov}(e) + \text{Cov}(f) - 2\text{Cov}(e, f)\) and \(\text{Cov}(e)\) being trivially separable, it is sufficient to prove that \(\text{Cov}(e, f)\) is separable. We show this by showing the contribution to \(\text{Cov}(\cdot, \cdot)\) of each type of allowable edges are separable.

First, we note that any edge originating from \(\text{top}(P)\), \(\text{top}(Q)\) or \(r\) does not contribute to \(\text{Cov}(e, f)\), making the contribution of such edges trivially separable. Second, the same-path edges do not contribute to \(\text{Cov}(e, f)\), making them trivially separable. Finally, there might exist edges that are incident to \(\text{bottom}(P)\) or \(\text{bottom}(Q)\). However, this is also separable: consider an edge \(e := \{\text{bottom}(P), x \in V(Q)\}\); the contribution of \(e\) to \(\text{Cov}(e, f)\) is \(\omega(c)\) if \(f\) is deeper than \(x\) and 0 otherwise, making it separable. The \(\{\text{bottom}(Q), x \in V(P)\}\) case is symmetric. This covers all allowable types of edges.

Finally, the functions \(F_P, F_Q\) are easily computable in \(\tilde{O}(1)\) Minor-Aggregation rounds. First, we compute the \(\text{Cov}(e)\) and \(\text{Cov}(f)\) using 1-respecting min-cut algorithm (Theorem 5.1). Following the case analysis from above, it is easy to calculate the contributions of cross-edges adjacent to \(\text{bottom}(P)\) or \(\text{bottom}(Q)\). Therefore, after we computed (and distributedly stored as edge vectors) \(F_P\) and \(F_Q\), we minimize each side separately and broadcast the result to \(G\). This is the minimizing 2-respecting cut since

\[
\min_{e \in E(P), f \in E(Q)} \text{Cut}_{T,G}(e, f) = \min_{e \in E(P), f \in E(Q)} (F_P(e) + F_Q(f)) = \min_{e \in E(P)} F_P(e) + \min_{f \in E(Q)} F_Q(f).
\]

**Proof.** Deferred to the full version.

Finally, with all ingredients in place, we can directly argue Theorem 6.1.

**Proof of Theorem 6.1.** If the instance is separable, or \(|P| \leq 10\), or \(|Q| \leq 10\), we can trivially solve the problem using Lemma 6.3 and Lemma 6.4. Otherwise, we use Lemma 6.5 and conclude.

7 **2-RESPECTING STAR MIN-CUT**

In this section we show how to compute the minimum 2-respecting cut between \(k\) subtrees \(T_1, T_2, \ldots, T_k\) (adjoined with a root for orientation purposes, see Figure 2). We call such an input a "star instance" and formalize it in the following definition.

**Definition 7.1.** A star instance \(\{P_i\}_{i=1}^k \subseteq T \subseteq G\) is composed of the following. Suppose \(G\) is a weighted graph and \(T \subseteq G\) is \(G\)'s (rooted) spanning tree. Moreover, \(T\) is composed of exactly a root \(r\), and \(k\) (disjoint) descending paths \(P_1, P_2, \ldots, P_k\).

The following result formalizes the goal, and is the main result that will be proved later in the section once sufficient tooling is developed. Unfortunately, due to space constraints, we will defer the rest of the section to the full version.

**Theorem 7.2.** Given a star instance \(\{P_i\}_{i=1}^k \subseteq T \subseteq G\), there exists a deterministic \(\tilde{O}(1)\)-round Minor-Aggregation algorithm on \(G\) that computes the minimum 1-respecting cuts and 2-respecting cuts

\[
\min_{e < j} \min_{e \in E(P_i), f \in E(P_j)} \text{Cut}_{T,G}(e, f).
\]

8 **2-RESPECTING BETWEEN-SUBTREE MIN-CUT**

In this section we show how to compute the minimum 2-respecting cut between \(k\) subtrees \(T_1, T_2, \ldots, T_k\) (adjoined with a root for orientation purposes, see Figure 3). We call such an input a "subtree instance" and formalize it in the following definition.

**Definition 8.1.** A subtree instance \(\{T_i\}_{i=1}^k \subseteq T \subseteq G\) is composed of the following. Suppose \(G\) is a weighted graph and \(T \subseteq G\) is \(G\)'s (rooted) spanning tree. Moreover, \(T\) is composed of exactly a root \(r\), and \(k\) (disjoint) trees \(T_1, T_2, \ldots, T_k\).

Our first idea is to reduce the problem for general \(k\) to the case when \(k = 2\). Suppose the optimum 2-respecting cut \(\langle e^*, f^* \rangle\) is contained in subtrees \(e^* \in E(T_i)\) and \(f^* \in E(T_j)\) where \(i^* \neq j^*\). We want to find a way to break symmetry between the subtrees \(i^*\) and \(j^*\), which we can do with the following structure.

**Definition 8.2.** Given a universe of \(k\) elements, a pairwise coloring is a collection \(\{f_1, \ldots, f_k\}\) where each \(f_i : \{k\} \to \{\text{red, blue}\}\) is called a color assignment which assign the color \(f_i(j)\) to element \(j\) such that for all pairs \(j \neq j' \in \{k\}\) there exists \(i \in \{\chi\}\) such that \(f_i(j) \neq f_i(j')\).

It is a folklore result that there exists such an assignment with \(\chi = O(\log n)\). After constructing such a collection of colorings, we can iterate over each color assignment, and for each assignment merge all the roots of all subtrees colored red and all subtrees colored blue. This is, however, done implicitly in the proof of Theorem 8.4.

**Lemma 8.3.** Given a subtree instance \(\{T_i\}_{i=1}^k \subseteq T \subseteq G\), we can compute and distributively store a pairwise coloring of the \(k\) subtrees \(\{T_i\}_{i=1}^k\) in \(\tilde{O}(1)\) deterministic Minor-Aggregation rounds.

**Proof.** Each subtree can compute its (arbitrary, but unique) ID by contracting all edges in the subtree and computing the minimum ID of all nodes within it. This ID has \(\chi := \tilde{O}(1)\) bits. We iterate over each bit \(i\) and assign create a new color assignment \(f_i\) for each bit: \(f_i(j) := \text{blue}\) if the \(i^{th}\) bit of \(j^{th}\) ID is 0 and \(f_i(j) := \text{red}\) otherwise.
The following result formalizes the goal of solving the between-subtree cuts, and is the main result of this section. The proof is illustrated with Figure 4.

**Theorem 8.4.** Given a subtree instance \( \{T_k\}_{k=1}^K \subseteq T \subseteq G \), there exists a deterministic \( O(1) \)-round Minor-Aggregation algorithm on \( G \) that computes the minimum of 1-respecting cuts and 2-respecting cuts \( \min_{e \in E(T(T))} \min_{f \in E(T)} \text{Cut}_{T,G}(e, f) \).

**Proof. Algorithm.** We first compute the 1-respecting cuts and remember the best result (Theorem 5.1). Furthermore, we construct a pairwise coloring \( \{f^k_i\}_{k=1}^K \) of \( \{T_k\}_{k=1}^K \). Next, we iterate over all possibilities for (1) color assignment \( i \in \{1, 2\} \), (2) HL-depth \( d_1 \leq O(\log n) \), (3) HL-depth \( d_2 \leq O(\log n) \). Next, all subtrees \( T_i \) with \( f^i(j) = \text{red} \) will contract all edges \( e \in E(T_i) \) where the HL-depth \( (e) = d_1 \) and all subtrees \( T_j \) with \( f^j(i) = \text{blue} \) will contract all edges \( e \in E(T_j) \) where the HL-depth \( (e) = d_2 \). This transforms the instance to a star instance (see Figure 4). We use the star instance algorithm to find the best 2-respecting cut on this star (Theorem 7.2). After iterating over all possibilities, the best result is returned. Note that, since there is \( O(1) \) color assignments, \( O(\log n) \) HL-depths, and the star algorithm takes \( O(1) \) rounds, the entire algorithm takes \( O(1) \) Minor-Agregation rounds.

**Correctness analysis.** First, we note that the algorithm only checks some number of existing 2-respecting cuts, hence it can never report an answer that is smaller than the optimum solution. We only need to show that is successfully managed to find the optimum. Suppose that \( (e^*, f^*) \in E(T^*) \times E(T^*) \) with \( a^* \neq b^* \) are the pair of edges that minimizes the 2-respecting cut. Let \( d_1^* := \text{HL-depth}(e^*) \) and \( d_2^* := \text{HL-depth}(f^*) \) and let \( P_1^* \) and \( P_2^* \) be the HL-paths which contain \( e^* \) and \( f^* \), respectively. Due to the definition of pairwise coloring, there exists a \( i^* \in \{1, 2\} \) where \( f^{i^*}(a^*) \neq f^{i^*}(b^*) \). Therefore, when the algorithm iterates over \( (i, d_1, d_2) = (i^*, d_1^*, d_2^*) \), it will transform the subtree instance \( \{T_k\}_{k=1}^K \subseteq T \subseteq G \) into a star instance \( \{P_k^\prime\}_{k=1}^K \subseteq T^* \subseteq G^* \) by contracting tree edges that are not on \( P_1^* \) or \( P_2^* \). However, since contraction of edges not on \( P_1^* \) or \( P_2^* \) does not change the 2-respecting cut values, we have that \( \text{Cut}_{T,G}(e^*, f^*) = \text{Cut}_{T^*,G^*}(e^*, f^*) \). Therefore, since \( \text{Cut}_{T,G}(e^*, f^*) \) will be considered in the star algorithm, the returned solution will return the optimum. \( \square \)

9 2-RESPECTING GENERAL MIN-CUT

In this section, we show how to compute the minimum 2-respecting on an arbitrary spanning tree \( T \) of a weighted graph \( G \).

**Theorem 9.1.** Given a spanning tree \( T \) of a weighted graph \( G \), there exists a deterministic \( O(1) \)-round Minor-Aggregation algorithm on \( G \) that computes the minimum 2-respecting cut \( \min_{e \in E(T)} \min_{f \in E(T)} \text{Cut}_{T,G}(e, f) \). This implies a deterministic \( O(\sqrt{n}) \)-round CONGEST algorithm for general graphs, and a deterministic \( O(1) \)-round CONGEST algorithm for excluded-minor graphs.

Due to space constraints, we defer the rest of this section to the full version of this paper (available on arXiv).

**ACKNOWLEDGMENTS**

We would like to thank Michal Dory, Bernhard Haeupler, and Richard Peng for helpful discussions. The authors were supported in part by funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 853109) and the Swiss National Foundation (project grant 200021-184735).

**REFERENCES**

[1] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Kent Quanrud, and Thatchaphol Saranurak. 2021. Minimum Cuts in Directed Graphs via Partial Sparsification. arXiv preprint arXiv:2111.08959 (2021).

[2] Richard Cole and Uzi Vishkin. 1986. Deterministic coin tossing with applications to optimal parallel list ranking. Information and Control 79, 1 (1986), 32–53.

[3] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Distributed edge connectivity in sublinear time. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing. 343–354.

[4] Atish Das Sarma, Stephan Hohre, Lish Kots, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed verification and hardness of distributed approximation. SIAM Journal on Computing (SICOMP) 41, 5 (2012), 1255–1265.

[5] Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2021. Distributed weighted min-cut in nearly-optimal time. In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing. 521–530.

[6] Michal Dory and Mohsen Ghaffari. 2019. Improved distributed approximations for minimum-weight two-edge-connected spanning subgraph. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. 521–530.

[7] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Minimum Cut in O(\log^2 n) Time. In Proceedings of the 47th International Colloquium on Automata, Languages and Programming (ICALP) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 168), Artur Czumaj, Anuj Dawar, and Emanuela Merelli (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 57:1–57:15. https://doi.org/10.4230/LIPIcs.ICALP.2020.57

[8] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2021. A note on a recent algorithm for minimum cut. In Symposium on Simplicity in Algorithms (SOSA). SIAM, 74–79.

[9] Mohsen Ghaffari and Bernhard Haeupler. 2016. Distributed Algorithms for Planar Networks II: Low-congestion Shortcuts, MST, and Min-Cut. In Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 202–219.

[10] Mohsen Ghaffari and Bernhard Haeupler. 2021. Low-congestion shortcuts for graphs excluding dense minors. In Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, 213–221.

[11] Mohsen Ghaffari and Fabian Kuhn. 2013. Distributed minimum cut approximation. In Proceedings of the 27th International Symposium on Distributed Computing (DISC). 1–15.

[12] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. 2017. Distributed MST and routing in almost mixing time. In Proceedings of the ACM Symposium on Principles of Distributed Computing, 131–140.

[13] Mohsen Ghaffari and Jason Li. 2018. New Distributed Algorithms in Almost Matching Time via Transformations from Parallel Algorithms. In Proceedings of the 32nd International Symposium on Distributed Computing (DISC). 31:1–31:16.

[14] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. 2020. Faster algorithms for edge connectivity via random 2-out contractions. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 1260–1279.

[15] Mohsen Ghaffari and Merav Parter. 2016. MST in log-star routes of congested clique. In Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC). 19–28.

[16] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. 2018. Round-and-message-optimal distributed graph algorithms. In Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC). 119–128.

[17] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. 2016. Low-congestion shortcuts without embedding. In Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC). 451–460.

[18] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. 2016. Near-Optimal Low-Congestion Shortcuts on Bounded Parameter Graphs. In Proceedings of the 38th International Symposium on Distributed Computing (DISC). 158–172.

[19] Bernhard Haeupler, Jason Li, and Goran Zuzic. 2018. Minor excluded network families admit fast distributed algorithms. In Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC). ACM. 465–474.

[20] Bernhard Haeupler, Harald Raecke, and Mohsen Ghaffari. 2022. Hop-Constrained Expander Decompositions, Oblivious Routing, and Distributed Universal Optimality. In Annual ACM Symposium on Theory of Computing (STOC) to appear.

[21] Bernhard Haeupler, David Wajc, and Goran Zuzic. 2021. Universally-Optimal Distributed Algorithms for Known Topologies. In Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC). ACM.

[22] David R Karger. 2000. Minimum cuts in near-linear time. Journal of the ACM (JACM) 47, 1 (2000), 46–76.
[23] Jason Li. 2021. Deterministic mincut in almost-linear time. In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 384–395.

[24] Andrés López-Martínez, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2021. Work-Optimal Parallel Minimum Cuts for Non-Sparse Graphs. In Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures, 351–361.

[25] Sagnik Mukhopadhyay and Danupon Nanongkai. 2020. Weighted min-cut: sequential, cut-query, and streaming algorithms. In Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC), 496–509.

[26] Danupon Nanongkai and Hsin-Hao Su. 2014. Almost-tight distributed minimum cut algorithms. In Proceedings of the 28th International Symposium on Distributed Computing (DISC), 439–453.

[27] Alessandro Panconesi and Romeo Rizzi. 2001. Some simple distributed algorithms for sparse networks. Distributed computing 14, 2 (2001), 97–100.

[28] Merav Parter. 2019. Small Cuts and Connectivity Certificates: A Fault Tolerant Approach. In 33rd International Symposium on Distributed Computing.

[29] David Peleg. 2000. Distributed computing: a locality-sensitive approach. SIAM.

[30] Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. 2022. Undirected (1+epsilon)-Shortest Paths via Minor-Aggregates: Near-Optimal Deterministic Parallel & Distributed Algorithms. Annual ACM Symposium on Theory of Computing (STOC) (2022).

[31] Goran Zuzic, Goramos Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. 2022. Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based Li-Oblivious Routing. In Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM.