Fast Distributed Computation of Cuts via Random Circulations

David Pritchard

University of Waterloo

Abstract. We describe a new *circulation*-based method to determine cuts in an undirected graph. A circulation is an oriented labeling of edges with integers so that at each vertex, the sum of the in-labels equals the sum of out-labels. For an integer k, our approach is based on simple algorithms for sampling a circulation (mod k) uniformly at random. We prove that with high probability, certain dependencies in the random circulation correspond to cuts in the graph. This leads to simple new linear-time sequential algorithms for finding all cut edges and *cut pairs* (a set of 2 edges that form a cut) of a graph, and hence 2-edge-connected and 3-edge-connected components.
In the model of *distributed computing* in a graph $G = (V, E)$ with $O(\log |V|)$-bit messages, our approach yields faster algorithms for several problems. The diameter of G is denoted by $\mathcal{D}$. Previously, Thurimella [J. Algorithms, 1997] gave a $O(\mathcal{D} + \sqrt{|V|}\log^* |V|)$-time algorithm to identify all cut vertices, 2-edge-connected components, and cut edges, and Tsin [Int. J. Found. Comput. Sci., 2006] gave a $O(|V| + \mathcal{D}^2)$-time algorithm to identify all cut pairs and 3-edge-connected components.
We obtain simple $O(\mathcal{D})$-time distributed algorithms to find all cut edges, 2-edge-connected components, and cut pairs, matching or improving previous time bounds on all graphs. Under certain assumptions these new algorithms are *universally optimal*, due to a $\Omega(\mathcal{D})$-time lower bound on every graph. These results yield the first distributed algorithms with *sub-linear* time for cut pairs and 3-edge-connected components. Let Δ denote the maximum degree. We obtain a $O(\mathcal{D} + \Delta/\log |V|)$-time distributed algorithm for finding cut vertices; this is faster than Thurimella's algorithm on all graphs with $\Delta, \mathcal{D} = O(\sqrt{|V|})$. The basic distributed algorithms are Monte Carlo, but can be made Las Vegas without increasing the asymptotic complexity.

1 Introduction

Let $G = (V, E)$ be a connected undirected graph. A part of G is said to be a *cut* if, after deleting it from G, the remaining graph is disconnected. We use the following terminology:

- A *cut vertex* is a vertex v such that $\{v\}$ is a cut.
- A *cut edge* is an edge e such that $\{e\}$ is a cut (i.e., a bridge).
- A *cut pair* is a cut consisting of two edges e, f such that neither e nor f is a cut edge.

For brevity we call all of these objects *small cuts*. In a network (e.g., for communication or transportation), the small cuts are relevant because they represent the critical points where local failures can cause global disruption. Our primary motivation is to efficiently find all small cuts of an undirected graph. We study this problem in the sequential, distributed, and parallel models of computation.

The fundamentally new idea in this paper is to use *random circulations* to find small cuts. Informally, in a circulation we transport quantities of a commodity along the edges of a graph, so that the net accumulation at each vertex is zero (i.e., it is like a *flow* without a source or sink). When the shipment quantities are taken modulo some integer k, there are only finitely many possible circulations, and our first contribution is the observation that it is easy to sample *uniformly* from the family of all circulations on a fixed graph.

For $S \subset V$, let $\delta(S)$ denote the edges with exactly one end in S. An *induced edge cut* is a set of the form $\delta(S)$ for some S; we observe that cut edges and cut pairs are induced edge cuts. A

well-known principle behind our method, made precise in Proposition 1, is that the net flow of any circulation across any induced edge cut is 0. At a high level, our algorithms depend on the near-converse: for certain edge sets F that are *not* induced edge cuts, the net flow of a uniformly random circulation on F is uniformly random, hence nonzero with high probability.

The Distributed Model. Our approach improves known time bounds in the *distributed* computing model with *congestion*. This model, denoted $\mathcal{CONGEST}$ (e.g. by Peleg [20, §2.3]), works as follows. The computation takes place in the graph $G = (V, E)$ where each vertex is a computer and each edge is a bidirectional communication link; i.e., we study the problem of having a network compute the small cuts of its own topology. There is no globally shared memory, only local memory at each vertex. Initially only local topology is known: each vertex knows its ID value, which is unique, and its neighbours' IDs. Time elapses in discrete *rounds*. In each round, every vertex performs local computations and may send one message to each of its neighbors, to be received at the start of the next round. The *time complexity* of a distributed algorithm is the number of rounds that elapse, and the *message complexity* is the total number of messages that are sent.

In the $\mathcal{CONGEST}$ model, every message must be at most $O(\log V)$ bits long. The model does not bound the memory capacity or computational power of the vertices, although our algorithms use time and space polynomial in $|V|$ at each vertex. Let $\mathcal{D}$ denote the diameter of (V, E), i.e. $\mathcal{D} := \max_{u,v \in V} \text{dist}_G(u, v)$. The message size bound, in addition to making the algorithms more practical, affects what is possible in the model, as the following example from Lotker, Patt-Shamir & Peleg [18] shows. On the one hand, if messages are allowed to be arbitrarily long, any graph property whatsoever can be trivially computed in $\mathcal{D}$ time[1]. On the other hand, Lotker et al. gave a family of graphs with $\mathcal{D} = 3$, such that in $\mathcal{CONGEST}$ on this family, a $\Omega(\sqrt[4]{|V|}/\sqrt{\log |V|})$-time lower bound holds to find the minimum spanning tree (MST).

Determining whether a task in this model can be accomplished in $O(\mathcal{D}) + o(|V|)$ time, or better yet $O(\mathcal{D})$ time, is a fundamental problem. For finding all cut edges and cut pairs of a graph, we give new affirmative answers by providing $O(\mathcal{D})$-time algorithms.

1.1 Existing Results

Our results apply to three common models of computation: sequential, distributed, and parallel. Abusing notation for readability, we sometimes abbreviate $|V|$ to V and $|E|$ to E.

Sequential. In the usual sequential (RAM) model of computing, Tarjan in the 1970s was the first to obtain linear-time ($O(V + E)$-time) algorithms to find all cut vertices [21], cut edges [21], and cut vertex-pairs (cuts $C \subset V$ with $|C| = 2$) [15]. These algorithms are based on depth-first search (DFS). Galil & Italiano, in 1991, gave the first linear-time algorithm to compute all cut pairs, by reducing to the cut vertex-pair problem.

Distributed. Here we only mention results valid in $\mathcal{CONGEST}$, ignoring results with $\Omega(n)$ message size such as one of Chang [6]. **Cut Edges/Vertices.** Two early distributed algorithms for cut edges and vertices, by Ahuja & Zhu [1] and Hohberg [14], use DFS. The smallest time complexity of any known distributed DFS algorithm is $\Theta(V)$; as such, the algorithms of Ahuja & Zhu and Hohberg have $\Omega(V)$ time complexity. Huang [16] gave a non-DFS-based algorithm with $\Theta(V)$ time complexity. A breakthrough by Thurimella [23] was an algorithm that is asymptotically faster than $\Theta(V)$ on some graphs (a so-called *sub-linear* algorithm). Precisely, Thurimella obtained time

[1] In $\mathcal{D}$ rounds each vertex broadcasts its local topology to all other vertices, then each vertex deduces the global topology and solves the problem with a local computation.

complexity[2] $O(\mathcal{D} + \sqrt{V}\log^* V)$ for cut edges and cut vertices, using a sub-linear MST subroutine. **Cut Pairs.** For cut pairs, Jennings and Motyckova [17] gave a distributed algorithm with worst-case time and message complexity $\Theta(n^3)$, and Tsin [25] recently obtained a DFS-based algorithm with improved time complexity $O(\mathcal{D}^2 + V)$.

Distributed Optimality. Distributed $\Theta(V)$-time algorithms for cut edges are optimal (up to a constant factor) on some graphs: e.g. it is straightforward to see, even guaranteed that G is either a $|V|$-cycle or a $|V|$-path, not all edges can determine if they are cut edges in less than $|V|/2 - 2$ rounds. One term for this property is *existentially optimal*, due to Garay, Kutten and Peleg [11]. However, as Thurimella's algorithm [23] showed, there are some graphs on which $\Theta(V)$ time is not asymptotically optimal. The stronger term *universally optimal* [11] applies to algorithms which, on *every* graph, have running time within a constant factor of the minimum possible.

Parallel. In the PRAM model, optimal $O(\log V)$-time and $O(V+E)$-work Las Vegas algorithms have been given by Tarjan & Vishkin [22] for cut edges and cut vertices, and Fussell, Ramachandran & Thurimella [9] (using the reduction of Galil & Italiano [10]) for cut pairs. These algorithms require optimal spanning forest subroutines of Halperin & Zwick [12].

1.2 Our Contributions

Since our algorithms are randomized, we differentiate between two types of algorithms: *Monte Carlo* ones have deterministically bounded running time but may be incorrect with probability $1/V$ and *Las Vegas* ones are always correct and have bounded *expected* running time[3]. (Note, a Las Vegas algorithm can always be converted to Monte Carlo, so Las Vegas is generally better).

Sequential. The random circulation approach yields *new linear-time algorithms to compute all cut edges and cut pairs* of the Las Vegas type. As far as we are aware, our linear-time cut pair algorithm is the first one that does not rely on either DFS (e.g., see references in Tsin [24]) or open ear decomposition (e.g., see references in Fussell et al. [9]).

Distributed. We remark that all existing distributed algorithms mentioned for finding small cuts are deterministic. The random circulation approach yields *faster distributed algorithms for small cuts* of the Las Vegas type. For cut edges and pairs, we obtain $O(\mathcal{D})$-time algorithms. Compared to the previous best time of $O(\mathcal{D} + \sqrt{V}\log^* V)$ for cut edges, we remove the dependence on $|V|$. Compared to the previous best time of $O(\mathcal{D}^2 + V)$ for cut pairs, we obtain a quadratic speedup on every graph. For cut vertices, we obtain a $O(\mathcal{D} + \Delta/\log V)$-time algorithm where Δ is the maximum degree. Compared to the previous best time of $O(\mathcal{D}+\sqrt{V}\log^* V)$ for cut vertices, this is faster on graphs with $\Delta, \mathcal{D} = O(\sqrt{V})$. We also obtain the first sub-linear distributed algorithm for 3-edge-connected components, using a connected components subroutine of Thurimella [23]. In Table 1 we depict our main results and earlier work, showing both time and message complexity.

Universal Optimality. If we assume distributed algorithms must act globally in a natural sense — either by initiating at a single vertex, or by reporting termination — then a $\Omega(\mathcal{D})$-time lower bound holds for the problems of finding cut edges or cut pairs, on any graph. Hence under natural conditions, our $O(\mathcal{D})$-time algorithms for cut edges and cut pairs are universally optimal.

Parallel. In the PRAM model, we obtain new optimal $O(\log V)$-time and $O(V + E)$-work Las Vegas algorithms for finding cut pairs and cut edges. Our algorithms require spanning forest subroutines of Halperin & Zwick [12]. These results are deferred to the full version of the paper.

[2] $\log^* x$ is the number of times which log must be iteratively applied to x before obtaining a number less than 1.

[3] More generally, our algorithms can obtain error probability $\leq 1/V^c$ for any constant c without changing the asymptotic complexity.

		Cuts Found	Time	Messages
[1]	'89	Vertices & Edges	$O(V)$	$O(E)$
[23]	'95	Vertices & Edges	$O(\mathcal{D} + \sqrt{V}\log^* V)$	$O(E \cdot (\mathcal{D} + \sqrt{V}\log^* V))$
[25]	'06	Pairs	$O(V + \mathcal{D}^2)$	$O(E + V \cdot \mathcal{D})$
Theorem 7†		Edges	$O(\mathcal{D})$	$O(E)$
Theorem 10†		Pairs	$O(\mathcal{D})$	$O(\min\{V^2, E \cdot \mathcal{D}\})$
Theorem 8†		Vertices	$O(\mathcal{D} + \Delta/\log V)$	$O(E(1 + \Delta/\log V))$

Table 1. Comparison of our three main distributed results (denoted by †) to the best previously known algorithms.

1.3 Other Related Work

Circulations have several applications in diverse fields. Hoffman's Circulation Theorem [13] is a min-max relation for circulations from which many other min-max relations can be derived. In planar graphs, *nowhere-zero* circulations modulo k correspond to vertex k-colourings of the dual graph (e.g. see the book of Zhang [26] for related work). Circulations also appear in the most efficient flow algorithm for planar directed graphs, obtained by Borradaile & Klein [5]. We defer a more extended discussion to the full version.

As far as we know, our usage of uniformly random circulations is novel. The closest related work we are aware of is by Benjamini & Lovász [3]: they give a method to compute the genus of an embedded graph G while only "observing" part of it. The similarity is that they use random perturbation and balancing steps to compute a "near-circulation" on G and the *dual graph* of G. Their computational model is significantly different, e.g. they allow a face to modify the values of all its incident edges in a single time step.

1.4 Organization of the Paper

Section 2 contains definitions and basic results pertaining to circulations. In Section 3 we define random circulations and show how to construct them efficiently. In Section 4 we show how random circulations yield algorithms for small cuts and give sequential implementations. In Section 5 we give our distributed results, starting with a precise discussion of input/output format and technical assumptions. In Section 5.2 we introduce a new technique, *fundamental cycle-cast*, which may be of independent interest. We defer the lower bounds leading to universal optimality to Appendix A and discussion of $\{2,3\}$-edge-connected components to Appendix B. We defer the Las Vegas versions of our algorithms to the full version of the paper; we only discuss Monte Carlo versions here.

2 Preliminaries

In this paper the set notation $\{u, v\}$ denotes an undirected edge of $G = (V, E)$, which can be oriented in two ways, denoted (u, v) and (v, u). For a set F of edges let $\overrightarrow{F}$ denote the $2|F|$ orientations of edges in F. An F-*orientation* is a subset of $\overrightarrow{F}$ consisting of exactly 1 orientation of each $e \in F$. Let $\mathbb{Z}_k$ denote the integers modulo k. For $v \in V$ the notation $\Gamma(v)$ denotes the set of neighbours of v.

Definition 1. *A* circulation on G *is a function* $\phi : \overrightarrow{E} \to \mathbb{R}$ *with the following two properties.*

Antisymmetry: $\phi(u, v) = -\phi(v, u)$ *for all* $\{u, v\} \in E$.
Conservation: $\sum_{v \in \Gamma(u)} \phi(u, v) = 0$ *holds for all vertices* u.

A k-circulation *is a function* $\phi : \overrightarrow{E} \to \mathbb{Z}_k$ *that satisfies* **Antisymmetry** *and* **Conservation** *when equality is replaced by equivalence modulo* k.

We illustrate a circulation in Figure 1. Although our notation is chosen for brevity, there are alternatives, e.g. we could equivalently define, for an E-orientation E', a circulation as a function $\phi : E' \to \mathbb{R}$ so that for each $u \in V$, $\sum_{v:(v,u)\in E'} \phi(v,u) = \sum_{v:(u,v)\in E'} \phi(u,v)$ is satisfied.

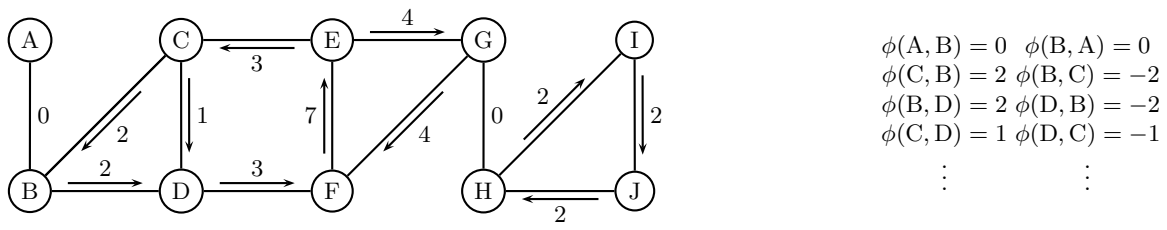

Fig. 1. A circulation ϕ. We have labeled edges in the direction of positive flow. Note that the net flow at each vertex is 0, i.e. conservation holds.

For $U \subsetneq V$, the *induced directed edge cut* $\delta^+(U)$ is the set of directed edges (u,v) with $u \in U, v \notin U$. We prove the following folklore result in Appendix C (see also [26, p. 7]).

Proposition 1 (Circulations vanish across induced cuts) *Let ϕ be a k-circulation on G, and let $U \subset V$. Then*

$$\sum_{(u,v)\in\delta^+(U)} \phi(u,v) = 0.$$

We immediately obtain the following corollary (see also [26, p. 8]).

Corollary 2 *If $\{u,v\}$ is a cut edge of G and ϕ is a circulation on G then $\phi(u,v) = \phi(v,u) = 0$.*

Proof. Let U be the connected component of $G\backslash\{\{u,v\}\}$ containing u. Now apply Proposition 1; the only member of $\delta^+(U)$ is (u,v) and so we obtain $\phi(u,v) = 0$. By antisymmetry, $\phi(v,u) = 0$. □

We now explain a tool which allows one to construct circulations; it appears implicitly in the book of Bondy & Murty [4, Ex. 12.1.1]. The idea is that for any spanning tree, we can choose any circulation values on the non-tree edges, and then there is a unique extension to a circulation on the whole graph.

Proposition 3 *Let T be any spanning tree of G and let $\phi_0 : \overrightarrow{E\backslash E(T)} \to \mathbb{Z}_k$ be antisymmetric. There is a unique circulation ϕ on G such that $\phi(u,v) = \phi_0(u,v)$ for all $\{u,v\} \in E\backslash E(T)$.*

Proof (Sketch). For a leaf node v incident to $\{u,v\} \in E(T)$, the value of $\phi(u,v)$ must equal $-\sum_t \phi_0(t,v)$ to satisfy conservation at v. The idea is to then delete $\{u,v\}$ from T and repeat. We give pseudocode in Algorithm 1 but the formal proof is deferred to Appendix C. □

3 Random Circulations

We begin this section by showing that it is easy to uniformly sample from the set of all k-circulations. The basic idea is to feed a "random" ϕ_0 in to Algorithm 1. More precisely, we pick the values of ϕ_0 randomly and independently (up to antisymmetry) from $\mathbb{Z}_k$. We denote this algorithm by Rand-k-Circ(T) and illustrate it in Algorithm 2.

Algorithm 1 Input: tree T, antisymmetric ϕ_0 on $E\backslash E(T)$. Output: circulation ϕ extending ϕ_0.

1: Initialize $\phi := \phi_0, S := T$. ▷ S is the subtree of T where ϕ is not yet defined
2: Root T at an arbitrary vertex r.
3: **while** S has any edges **do**
4: Let v be any leaf of S with $v \neq r$ and let u be the unique neighbor of v in S.
5: Define $\phi(v,u) := -\sum_{w \in \Gamma(v)\backslash\{u\}} \phi(v,w)$. ▷ Satisfy conservation at v
6: Define $\phi(u,v) := -\phi(v,u)$. ▷ Satisfy antisymmetry
7: Delete $\{u,v\}$ from S.
8: Output ϕ.

Algorithm 2 Input: a connected graph G. Output: the cut edges of G.

1: **procedure** RAND-k-CIRC(T)
2: **for** each edge $\{u,v\} \in E\backslash E(T)$ **do**
3: Pick $x \in \mathbb{Z}_k$ uniformly and independently at random and set $\phi_0(u,v) = x, \phi_0(v,u) = -x$.
4: Return the unique circulation that extends ϕ_0 by calling Algorithm 1.

Theorem 1 *Let ϕ^* be a circulation on G and T be a spanning tree of G. Let ϕ be the output of* RAND-k-CIRC(T). *Then*

$$\Pr[\phi = \phi^*] = 1/k^{|E|-|V|+1}, \tag{1}$$

and the distribution produced by RAND-k-CIRC(T) *over all k-circulations is uniform.*

Proof. The tree T has $|V|-1$ edges, so $|E\backslash E(T)| = |E| - |V| + 1$. Clearly, the probability that ϕ_0 agrees with ϕ^* on $E\backslash E(T)$ is exactly $1/k^{|E|-|V|+1}$. But furthermore, by Proposition 3, $\phi = \phi^*$ if and only if ϕ_0 and ϕ^* agree on $E\backslash E(T)$, hence we obtain Equation (1). Since $1/k^{|E|-|V|+1}$ does not depend on ϕ^*, uniformity follows. □

Note, Theorem 1 implies that different choices of T have no effect on the output of RAND-k-CIRC(T). Because of this we later refer to a "random k-circulation," meaning to sample a k-circulation uniformly at random by calling RAND-k-CIRC with any spanning tree. We will make repeated use of the following corollary to show that ϕ "behaves randomly" on certain edge sets.

Corollary 4 *Let $D \subset E$ be such that $G\backslash D$ is connected, and D' be a D-orientation. Let ϕ be a random k-circulation. The values of ϕ on D' are uniformly and independently distributed over $\mathbb{Z}_k$.*

Proof. Since $G\backslash D$ is connected, it contains a spanning tree T of G. By Theorem 1, the distribution of ϕ is the same as if ϕ were generated by running RAND-k-CIRC with this choice of T. When calling RAND-k-CIRC on this T, each $e \in E\backslash E(T) \supset D$ incurs a uniform, independent sample $x \in \mathbb{Z}_k$ on line 1. (Notice that if x is uniformly distributed over $\mathbb{Z}_k$, so is $-x$). The result then follows. □

Note that Corollary 4 holds regardless of what spanning tree was actually used to generate ϕ.

4 Sequential Algorithms

In this section we show how to use random circulations to probabilistically determine the cut edges, cut pairs, and cut vertices of a graph. These are the Monte Carlo versions of the algorithms.

4.1 Finding All Cut Edges

Proposition 5 *Let $\{u, v\} \in E$ and ϕ be a random k-circulation on G. Then $\Pr[\phi(u, v) = 0]$ is 1 if $\{u, v\}$ is a cut edge and $1/k$ otherwise.*

Proof. If $\{u, v\}$ is a cut edge then Corollary 2 applies. Otherwise by Corollary 4 the value $\phi(u, v)$ is a uniformly random element of $\mathbb{Z}_k$, since $G \backslash \{\{u, v\}\}$ is connected. □

Thus, provided we pick k large enough, it is likely that the cut edges are exactly $\{\{u, v\} \mid \phi(u, v) = 0\}$. We provide pseudocode in Algorithm 3 and prove its correctness.

Algorithm 3 Input: a connected graph G. Output: the cut edges of G.

1: Let $k = |V||E|$ and let ϕ be a random k-circulation on G.
2: Output all edges $\{u, v\}$ for which $\phi(u, v) = 0$.

Theorem 2 *Algorithm 3 correctly determines the cut edges with probability $1 - 1/V$ and can be implemented in $O(E)$ sequential time.*

Proof. The algorithm chooses $k = |V||E|$. A union bound, in conjunction with Proposition 5, shows that the probability of error is at most $|E|/k = 1/|V|$. As is standard, we assume the machine word size is $\Omega(\log V)$. The subroutine RAND-k-CIRC performs $O(E)$ random choices and arithmetic operations, each of which take $O(1)$ time since k is $O(\log V)$ bits long. □

4.2 Finding All Cut Pairs and Cut Classes

For cut pairs and cut vertices we work with circulations only modulo $k = 2$. This is convenient because $x = -x$ for all $x \in \mathbb{Z}_2$, and hence we can unambiguously refer to $\phi(e)$ for an edge e without specifying an orientation. The *cycle space* of an undirected graph is the family of subsets of E with even degree at each vertex, see e.g. Bondy & Murty [4, §12.1]. We remark that 2-circulations are the same as characteristic vectors of members of the cycle space.

Proposition 6, which we prove in Appendix D, leads to our approach for finding cut pairs.

Proposition 6 (Cut pairs are induced) *If $\{e, f\}$ is a cut pair then $\{e, f\} = \delta(U)$ for some $U \subset V$.*

Proposition 7 *Let e, f be two distinct edges that are not cut edges. If ϕ is a random 2-circulation on G, then $\Pr[\phi(e) = \phi(f)] = 1$ if $\{e, f\}$ is a cut pair, and $1/2$ otherwise.*

Proof. If these two edges form a cut pair, using Proposition 6 and Proposition 1, we know that $\phi(e) + \phi(f) = 0$ and so $\phi(e) = \phi(f)$. Now suppose otherwise, that $\{e, f\}$ is not a cut pair. Then $G \backslash \{e, f\}$ is connected, and by Corollary 4 the values of ϕ on e and f are independent and uniform over $\mathbb{Z}_2$ whence $\Pr[\phi(e) = \phi(f)] = 1/2$. □

Proposition 7 gives us a probabilistic proof of the following fact.

Corollary 8 (Transitivity of cut pairs) *If $\{e, f\}$ and $\{f, g\}$ are cut pairs, then so is $\{e, g\}$.*

Proof. Note that e, f, g are not cut edges. Let ϕ be a random 2-circulation on G. By Proposition 7, $\phi(e) = \phi(f)$ and $\phi(f) = \phi(g)$. So $\phi(e) = \phi(g)$ with probability 1. By Proposition 7, $\{e, g\}$ must be a cut pair. □

Definition 2. *A* cut class *is an inclusion-maximal subset K of E such that $|K| > 1$ and every pair $\{e, f\} \subseteq K$ is a cut pair.*

Corollary 8 implies that any two distinct cut classes are disjoint. Hence, even though there may be many cut pairs, we can describe them all compactly — e.g. in $O(E)$ space in the sequential model — by listing all cut classes of the graph.

Let $\mathbb{Z}_2^b$ denote the set of b-bit binary strings. For $\phi : E \to \mathbb{Z}_2^b$, let $\phi_i(e)$ denote the ith bit of $\phi(e)$.

Definition 3. *A* b-bit circulation *is obtained by concatenating b 2-circulations $\{\phi_i\}_{i=1}^b$. A* (uniformly) random b-bit circulation *is obtained by concatenating b independent uniformly random 2-circulations $\{\phi_i\}_{i=1}^b$.*

Let $\oplus$ denote the bitwise xor operation. Notice that $\phi : E \to \mathbb{Z}_2^b$ is a b-bit circulation if and only if $\bigoplus_{e \in \delta(u)} \phi(e) = \mathbf{0}$ holds for each $u \in V$. The results of Sections 2 and 3 apply to b-bit circulations; for example, we can obtain Rand-b-Bit-Circ, a modified version of Rand-k-Circ that generates a uniformly random b-bit circulation, by replacing $\sum$ in Line 5 of Algorithm 1 by $\bigoplus$, replacing $\mathbb{Z}_k$ in Line 3 of Algorithm 2 by $\mathbb{Z}_2^b$, and ignoring occurrences of the unary $-$ operator. Propositions 5 and 7 give probability bounds of $1/2^b$ in place of $1/2$ when use random b-bit circulations instead of random 2-circulations, due to the independence of each bit.

We now give our simple linear-time algorithm to find all cut classes. The idea is to compute a random b-bit circulation for large enough b that $\phi(e) = \mathbf{0}$ only for cut edges, and so that ϕ labels the cut classes of other edges. Pseudocode is given in Algorithm 4.

Algorithm 4 Input: a connected graph G. Output: the cut classes of G.

1: Let $b = \lceil \log_2(|V||E|^2) \rceil$ and let ϕ be a random b-bit circulation on G.
2: For each $x \in \mathbb{Z}_2^b \backslash \{\mathbf{0}\}$, if $|\{e \in E \mid \phi(e) = x\}| \geq 2$, then output the cut class $\{e \in E \mid \phi(e) = x\}$.

Theorem 3 *Algorithm 4 correctly determines the cut pairs with probability $1 - 1/V$ and can be implemented in $O(E)$ sequential time.*

Proof. There are $|E|$ edges and Proposition 5 shows that $\Pr[\phi(e) = \mathbf{0}] \leq 1/2^b$ for each non-cut edge e. There are at most $\binom{E}{2}$ pairs $\{e, f\}$ of non-cut edges that are not cut pairs and Proposition 7 shows that $\Pr[\phi(e) = \phi(f)] \leq 1/2^b$ for each such pair. Hence, by a union bound and our choice $b = \lceil \log_2(|V||E|^2) \rceil$, the total probability of error is at most $|E|/2^b + \binom{E}{2}/2^b \leq 1/V$.

The subroutine Rand-b-Bit-Circ performs $O(E)$ random choices and xor operations on b-bit binary strings, each of which take $O(1)$ time since $b = O(\log V)$. To implement Line 2 in Algorithm 4 we sort all edges e according to the key $\phi(e)$. In particular, we use a three-pass *radix sort* (i.e., we consider each value in $\mathbb{Z}_2^b$ as a three-digit number in base $2^{b/3} = O(E)$ — see Cormen, Leiserson & Rivest [7, §9.3]), which runs in time $O(E)$. □

4.3 Finding All Cut Vertices

As we show in this section, the cut $\delta(v)$ properly contains smaller induced edge cuts iff v is a cut vertex. The essential idea behind our approach is to detect these induced edge cuts, in order to determine the cut vertices. We detect induced edge cuts via Proposition 1, which says that circulations vanish across induced edge cuts. To do so efficiently, we rephrase the detection problem as one of finding linearly dependent rows of a binary matrix. Hence we need the following fact, when $\mathbb{Z}_2$ is viewed as a field.

Fact 9 *In a matrix over $\mathbb{Z}_2$, a set C of columns is linearly dependent if and only if some nonempty subset of C sums to the zero column vector* $(\mathrm{mod}\ 2)$.

Our approach works as follows. We generate a random b-bit circulation ϕ for some suitably large b; denote the ith bit of $\phi(e)$ by $\phi_i(e)$. Let $d(v) := |\delta(v)|$, the *degree* of v. Let Δ denote the maximum degree. For each vertex v, let $M^{[v]}$ be a matrix with b rows indexed $1, \ldots, b$, and $d(v)$ columns indexed by $\delta(v)$; then fill the entries of $M^{[v]}$ according to $M^{[v]}_{ie} = \phi_i(e)$. The following two complementary claims validate our approach; their proofs are deferred to Appendix E.

Claim 10 *If v is a cut vertex then* $\mathrm{rank}(M^{[v]}) \leq d(v) - 2$.

Claim 11 *Let $v \in V$ and assume that v is not a cut vertex. Let $\emptyset \subsetneq D \subsetneq \delta(v)$. The probability that the columns of $M^{[v]}$ indexed by D sum to the zero vector* $(\mathrm{mod}\ 2)$ *is 2^{-b}.*

Next we show that for $b = \lceil \Delta + 2\log_2 |V| \rceil$, it is very likely that $\mathrm{rank}(M^{[v]}) < d(v) - 1$ iff v is a cut vertex. Thus our approach, with pseudocode given in Algorithm 5, is correct with high probability. It is not very efficient in the sequential model, but still runs in $\mathrm{poly}(V)$ time.

Algorithm 5 Input: a connected graph G. Output: the cut vertices of G.

1: Let $b = \lceil \Delta + 2\log_2 |V| \rceil$ and let ϕ be a random b-bit circulation on G.
2: for each vertex v of G, if $\mathrm{rank}(M^{[v]}) < d(v) - 1$ then output v.

Theorem 4 *Algorithm 5 correctly determines the cut vertices with probability $1 - 1/V$.*

Proof. Claim 10 shows that all cut vertices are output. Consider a vertex v that is not a cut vertex and let D be a subset of $\delta(v)$ of size $d(v) - 1$. By Claim 11, Fact 9, and a union bound, the probability that the columns of $M^{[v]}$ corresponding to D are linearly dependent is at most $2^{d(v)-1}2^{-b} \leq 1/|V|^2$; so with probability $1 - |V|^{-2}$, we have $\mathrm{rank}(M^{[v]}) \geq |D| = d(v) - 1$ and v is not output. By another union bound, the probability that any vertex is misclassified by Algorithm 5 is at most $|V|/|V|^2 = 1/|V|$. □

5 Distributed Implementation

Our algorithms make the following three assumptions: first, the network is synchronous; second, there is a distinguished *leader* vertex at the start of computation; third, every node begins with a unique $O(\log V)$-bit ID. These assumptions are standard in the sense that they are made by the best previous distributed algorithms [1, 23, 25] for small cuts. Nonetheless, these assumptions can be removed at a cost if desired, e.g. using the synchronizer of Awerbuch and Peleg [2] at a polylog(V) factor increase in complexity, Peleg's [19] $O(\mathcal{D})$-time leader election algorithm, or by randomly assigning IDs in the range $\{1, \ldots, |V|^3\}$ (resulting in additional failure probability at most $\binom{V}{2}/|V|^3$ due to ID collisions).

Although only vertices can store data in the distributed model, we maintain data for each edge e (e.g., to represent a tree) by having both endpoints of e store the data. At the end of the algorithm, we require that the correct result is known locally, so each node stores a boolean variable indicating whether it is a cut node, and similarly for edges. To indicate cut pairs, each edge must know whether it is in any cut pair, and in addition we must give every cut class a distinct label. Previous work also essentially uses these representations.

When stating distributed algorithms, the assumptions of a leader, synchrony, unique IDs, and $O(\log V)$-bit messages are implicit. Our algorithms use a breadth-first search (BFS) tree with a root r as the basis for communication. One reason that BFS trees are useful is that they can be constructed quickly (e.g., see Peleg [20, §5.1]), as follows.

Proposition 12 *There is a distributed algorithm to construct a BFS tree in $O(\mathcal{D})$ time and $O(E)$ messages.*

For a tree T, the *level* $l(v)$ of $v \in V$ is the distance in T between v and r. The *height* $h(T)$ of tree T is the maximum vertex level in T. Any BFS tree T has $h(T) \leq \mathcal{D}$ and this is important because several fundamental algorithms based on passing information up or down the tree take $O(h(T))$ time. The *parent* of u is denoted $p(u)$. The *level of tree edge* $\{u, p(u)\}$ is the level of u.

5.1 Random Circulations, Cut Edges, and Cut Vertices

When we construct a random circulation, we require at termination that each v knows $\phi(v, u)$ for each $u \in \Gamma(v)$.

Theorem 5 *There is a distributed algorithm to sample a uniformly random k-circulation in $O(\mathcal{D})$ time and $O(E)$ messages, when $k = \text{poly}(V)$.*

Proof. We implement Rand-k-Circ distributively. Since $k = \text{poly}(V)$, any value in $\mathbb{Z}_k$ can be sent in a single $O(\log V)$-bit message. We compute a BFS tree T, using Proposition 12. Then for each non-tree edge $\{u, v\}$ in parallel, the endpoint with the higher ID (say, u) sets $\phi(u, v)$ to a random value in $\mathbb{Z}_k$, sends the value $\phi(u, v)$ to v, and then v sets $\phi(v, u) := -\phi(u, v)$. In the following $h(T)$ rounds, for $i = h(T)$ down to 1, for all level-i tree edges $\{v, p(v)\}$ in parallel, vertex v assigns $\phi(v, p(v))$ a value so that conservation is satisfied at v (like in Algorithm 1), notifies $p(v)$ of this value with a message, and then $p(v)$ sets $\phi(p(v), v) := -\phi(v, p(v))$. Termination occurs after r computes its incident ϕ values. This takes $O(\mathcal{D} + h(T)) = O(\mathcal{D})$ time and $O(E)$ messages, as claimed. □

Theorem 6 *There is a distributed algorithm to sample a uniformly random b-bit circulation in $O(\mathcal{D})$ time and $O(E)$ messages, when $b = O(\log V)$.*

Proof. Any value in $\mathbb{Z}_2^b$ can be sent in a single $O(\log V)$-bit message. Thus, analogous to the proof of Theorem 5, we implement Rand-b-Bit-Circ distributively. □

Theorem 5 yields our distributed cut edge algorithm.

Theorem 7 *There is a distributed algorithm to compute all cut edges with probability $1 - 1/V$ in $O(\mathcal{D})$ time and using $O(E)$ messages.*

Proof. We implement Algorithm 3 distributively, obtaining the required correctness probability by Theorem 2. For $k = |V||E|$, we use Theorem 5 to compute a random k-circulation in the required complexity bounds. Then we identify $\{u, v\}$ as a cut edge if $\phi(u, v) = 0$. □

A straightforward implementation of Algorithm 5 results in our cut vertex algorithm, as follows.

Theorem 8 *There is a distributed algorithm to compute all cut vertices with probability $1 - 1/V$ in $O(\mathcal{D} + \Delta/\log V)$ time and using $O(E(1 + \Delta/\log V))$ messages.*

Proof (Sketch). In Appendix F we give a full proof, and define a technique called *pipelining*. Using pipelining and Theorem 6, we efficiently sample a random b-bit circulation for $b = \lceil \Delta + 2\log_2 |V| \rceil$. Then, since local computations are free in the distributed model, each vertex v can immediately compute $\text{rank}(M^{[v]})$. □

5.2 Fundamental Cycle-Cast (fc-cast)

We now define a new distributed technique. A non-tree edge is an edge $e \in E \backslash E(T)$. For a spanning tree T and non-tree edge e, the unique cycle in $T \cup \{e\}$ is called *the fundamental cycle of T and e*, and we denote it by C_e. We call our new technique *fundamental cycle-cast*, or *fc-cast* for short, and informally it allows simultaneous processing on all fundamental cycles. Let each vertex v store some data $\mathtt{d}[v]$ of length $O(\log V)$ bits. WOLOG $\mathtt{d}[v]$ includes the ID, level, and parent ID of v. At the end of the fc-cast, each non-tree edge e will know $\mathtt{d}[u]$ for every vertex u in the fundamental cycle of T and e. We defer the proof of Theorem 9 to Appendix G.

Theorem 9 *There is a distributed algorithm* Fc-Cast *using $O(h(T))$ time and $O(\min\{E \cdot h(T), V^2\})$ messages that, for each non-tree edge e, for each $v \in C_e$, sends $\mathtt{d}[v]$ to both endpoints of e.*

5.3 Distributed Cut Pair Algorithm

It is not obvious how to implement our sequential cut pair algorithm (Algorithm 4) distributively: although the cut classes are properly labeled with high probability by ϕ, in order for edge e to know whether it belongs to any cut pair, it needs to determine if any other f has $\phi(e) = \phi(f)$, and this cannot be done using local information (i.e., in $O(1)$ rounds). We use fc-cast to overcome this obstacle. The following claim relates fundamental cycles to cut classes; it is proved in Appendix D.

Claim 13 *Let K be a cut class. Then $K \subset C_e$ for some $e \in E \backslash E(T)$.*

To describe our cut pair algorithm we introduce a variant of a standard technique, the *convergecast* (e.g., see Peleg [20, §4.2]). Informally, it allows each node to independently query its descendants. Let $desc(v)$ denote the set of v's descendants, including v itself. For each $v \in V$, and each $u \in desc(v)$, let $\mathtt{w}[u, v]$ be a boolean variable stored at u. We defer the proof of Proposition 14 to Appendix G. Then we give our distributed cut pair algorithm.

Proposition 14 *There is a distributed algorithm* Converge-Cast *using $O(h(T))$ time and $O(V \cdot h(T))$ messages so that each $v \in V$ determines whether any $u \in desc(v)$ has $\mathtt{w}[u, v] =$ TRUE.*

Theorem 10 *There is a distributed algorithm to compute all cut classes with probability $1 - 1/V$ in $O(\mathcal{D})$ time and using $O(\min\{E \cdot \mathcal{D}, V^2\})$ messages.*

Proof. We will use two claims below; their proofs are deferred to Appendix H. As in Algorithm 4, for $b = \lceil \log_2(|V||E|^2) \rceil$ we compute a random b-bit circulation ϕ on G, using Theorem 6. Denote the following assumption by $(\star)$.

$$\text{For all edges } e, f \text{ that are not cut edges, } \phi(e) = \phi(f) \text{ if and only if } \{e, f\} \text{ is a cut pair.} \qquad (\star)$$

By the analysis in Theorem 3, we may assume that $(\star)$ holds without violating the required bound of $1/V$ on the probability of error.

It remains only for each edge to determine whether it is a member of any cut pair, since then ϕ labels the cut classes. For each vertex $v \neq r$ let $\mathtt{d}[v] := \phi(v, p(v))$. We run Fc-Cast, and as a result, the endpoints of each non-tree edge e can compute the multiset $\Phi_e := \{\phi(f) \mid f \in C_e\}$. The following claim lets each non-tree edge determine if it is a member of any cut pair.

Claim 15 *A non-tree edge e is in a cut pair if and only if $\phi(e)$ occurs multiple times in Φ_e.*

To deal with tree edges, for each $v \in V$ and each $u \in desc(v)$, define

$$\mathtt{w}[u,v] := (\exists e \in \delta(u) \backslash E(T) \text{ such that } \{v, p(v)\} \in C_e \text{ and } \phi(v, p(v)) \text{ occurs multiple times in } \Phi_e).$$

and note that $\mathtt{w}[u,v]$ can be determined by u after the fc-cast. We run Converge-Cast.

Claim 16 *Tree edge $\{v, p(v)\}$ is in a cut pair if and only if $\exists u \in desc(v)$ such that $\mathtt{w}[u,v] =$ TRUE.*

By Proposition 14, after the convergecast, each tree edge can use Claim 16 to determine if it is a member of any cut pair. Adding up the complexity associated with constructing a BFS tree and a random circulation, the fc-cast, and the converge-cast, we obtain $O(\mathcal{D} + \mathcal{D} + \mathcal{D} + \mathcal{D})$ time and $O(E + E + \min\{E\mathcal{D}, V^2\} + V\mathcal{D}) = O(\min\{E\mathcal{D}, V^2\})$ messages, as claimed. □

References

1. M. Ahuja and Y. Zhu. An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks. In *Proc. 9th FSTTCS*, pages 99–108, 1989.
2. B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st FOCS*, pages 514–522, 1990.
3. I. Benjamini and L. Lovász. Harmonic and analytic functions on graphs. *J. Geom.*, 76(1):3–15, 2003. Preliminary version appeared in *Proc. 43rd FOCS*, pages 701–710, 2002.
4. A. Bondy and U. Murty. *Graph Theory with Applications*. North-Holland, 1976.
5. G. Borradaile and P. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. In *Proc. 17th SODA*, pages 524–533, 2006.
6. E. J.-H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Trans. Softw. Eng.*, SE-8:391–401, 1982.
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
8. M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006. Preliminary version appeared in *Proc. 15th SODA*, pages 359–368, 2004.
9. D. S. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacement. *SIAM J. Comput.*, 22:587–616, 1993.
10. Z. Galil and G. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22:57–61, 1991.
11. J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998. Preliminary version appeared in *Proc. 34th FOCS*, pages 659–668, 1993.
12. S. Halperin and U. Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests. *J. Algorithms*, 39(1):1–46, 2001. Preliminary version appeared in *Proc. 7th SODA*, pages 438–447, 1996.
13. A. Hoffman. Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In *Proc. 10th AMS Symp. on Appl. Math.*, pages 113–127, 1960.
14. W. Hohberg. How to find biconnected components in distributed networks. *J. Parallel Distrib. Comput.*, 9(4):374–386, 1990.
15. J. Hopcroft and R. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comp.*, 2(3):135–158, 1973.
16. S. T. Huang. A new distributed algorithm for the biconnectivity problem. In *Proc. 1989 International Conf. Parallel Processing*, pages 106–113, 1989.
17. E. Jennings and L. Motyckova. Distributed computation and incremental maintainance of 3-edge-connected components. In *Proc. 3rd SIROCCO*, pages 224–240, 1996.
18. Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006. Preliminary version appeared in *Proc. 20th PODC*, pages 63–71, 2001.
19. D. Peleg. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.*, 8(1):96–99, 1990.
20. D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.
21. R. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.
22. R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. Comput.*, 14(4):862–874, 1985. Preliminary version appeared in *Proc. 25th FOCS*, pages 12–20, 1984.
23. R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *J. Algorithms*, 23(1):160–179, 1997. Preliminary version appeared in *Proc. 14th PODC*, pages 28–37, 1995.
24. Y. H. Tsin. A simple 3-edge-connected component algorithm. *Theory Comput. Systems*, 40(2):125–142, 2005.
25. Y. H. Tsin. An efficient distributed algorithm for 3-edge-connectivity. *Int. J. Found. Comput. Sci.*, 17(3):677–702, 2006.
26. C.-Q. Zhang. *Integer flows and cycle covers of graphs*. Marcel Dekker, 1997.

A Time Lower Bounds

Let r denote the unique leader vertex in the graph. A vertex is *quiescent* in a given round if it does not send any messages or modify its local memory in that round. We adopt the following terminology from Peleg [20, §3.4 & Ch. 24].

Definition 4. *A distributed algorithm has* termination detection *if r has a local boolean variable* `done`*, initialized to* FALSE*, so that* `done` *is set to* TRUE *exactly once, in the last round of the algorithm. A distributed algorithm has* a single initiator *if, except for r, every vertex is quiescent until it receives a message.*

For now, assume we only allow deterministic distributed algorithms. The *state* of a vertex means the contents of its memory. We omit the easy inductive proof of the following proposition.

Proposition 17 *Suppose two graphs G and G', both containing a vertex v, agree*[4] *in the distance-t neighbourhood of v. If we instantiate the same distributed algorithm on G and G', the state of v will be the same in both instances for the first t rounds.*

For a graph $v \in V$ and a positive even integer $\ell \geq 4$, let $G_c^{\ell,v}$ denote the graph obtained from G by attaching a ℓ-edge cycle to G at v, and $G_p^{\ell,v}$ the graph obtained from G by attaching a $(\ell-1)$-edge path to G at v, as shown in Figure 2. Give corresponding vertices v_i in the two graphs the same ID. For this construction, Proposition 17 yields the following corollary.

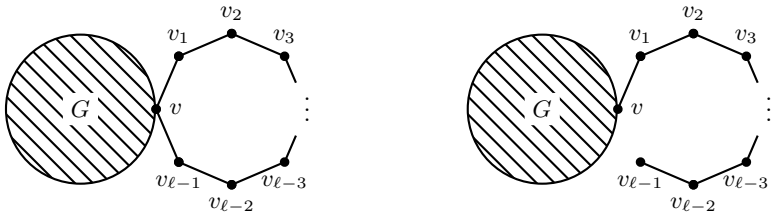

Fig. 2. Left: the graph $G_c^{\ell,v}$. Right: the graph $G_p^{\ell,v}$.

Corollary 18 *If we run a distributed algorithm on $G_p^{\ell,v}$ and $G_c^{\ell,v}$, for the first $\ell/2-1$ rounds, vertex $v_{\ell/2}$ in both instances has the same state.*

Proof. Observe that $v_{\ell/2}$ has the same distance-$(\ell/2-1)$ neighbourhood in both graphs. □

Theorem 11 *Any correct distributed algorithm for finding all cut edges that has termination detection takes at least $\mathcal{D}/2$ rounds on every graph.*

Proof. Suppose otherwise, that the algorithm terminates in less than $\mathcal{D}/2$ rounds on some G. Let v be any vertex of distance at least $\mathcal{D}/2$ away from r, and let $\ell = 2\lceil \mathcal{D}/2 \rceil + 2$. Then r also sets `done`:=TRUE in less than $\mathcal{D}/2$ rounds on $G_c^{\ell,v}$ and $G_p^{\ell,v}$ (applying Proposition 17 once to G, $G_c^{\ell,v}$ and r, and a second time to G, $G_p^{\ell,v}$ and r). The new edges of $G_p^{\ell,v}$ are cut edges, while the new edges of $G_c^{\ell,v}$ are not. But by Corollary 18, when the algorithm terminates, one of the two instances must incorrectly classify the edges incident to $v_{\ell/2}$. □

If we assume that the algorithm has a single initiator instead of assuming termination detection, a similar argument works. We use the following lemma, whose easy inductive proof is omitted.

[4] By *agree* we mean that the graph topology and vertex IDs are the same.

Lemma 19 *In an algorithm with a single initiator, every vertex at distance t from r is quiescent for the first t rounds.*

Theorem 12 *Any correct distributed algorithm for finding all cut edges that has a single initiator takes at least $\mathcal{D}/2$ rounds on every graph.*

Proof. Suppose otherwise, that the algorithm terminates in less than $\mathcal{D}/2$ rounds on some G. Let v be any vertex of distance at least $\mathcal{D}/2$ away from r. Then the algorithm also terminates in less than $\mathcal{D}/2$ rounds on $G_c^{4,v}$ and $G_p^{4,v}$. But by Lemma 19 the new vertices in these graphs are quiescent during the entire execution of the algorithm; hence, the edges incident to v_2 must be incorrectly classified in one instance. □

Similar arguments yield the same lower bounds for finding 2-edge-connected components and cut pairs, since the new edges of $G_c^{\ell,v}$ are in cut pairs, while the new edges of $G_p^{\ell,v}$ are not. A slight modification shows that Monte Carlo algorithms cannot terminate in $o(\mathcal{D})$ time with probability greater than $2/V$, hence they have expected running time $\Omega(\mathcal{D})$. It is straightforward to verify that our distributed algorithms can be implemented so as to have a single initiator and termination confirmation; then their universal optimality follows.

We make two remarks. First, we do not believe the techniques in this section are novel, but the results are important because they make precise the assumptions under which we claim to achieve universal optimality. Second, if we do not require a single initiator or termination confirmation, or if we change our input model to allow additional parameters of G to be initially known at each node, *neighbourhood cover* techniques of Elkin [8] can be synthesized with our techniques to yield even faster algorithms for certain graph classes. Elkin used these techniques to obtain distributed MST algorithms faster than $O(\mathcal{D})$ on some graphs.

B Computing {2, 3}-Edge-Connected Components

Let E_C denote the set of all cut edges, and E_{CP} denote the set of all edges in any cut pair.

Definition 5. *The* 2-edge-connected components *are the connected components of $G \backslash E_C$. The* 3-edge-connected components *are the connected components of $G \backslash (E_{CP} \cup E_C)$.*

In the sequential model, connected components of a graph can be computed in linear time. Hence we immediately see that our linear-time sequential cut edge and cut pair algorithms yield linear-time algorithms for 2- and 3-edge-connected components.

In the distributed model, we first discuss 2-edge-connected components. Let T denote a spanning tree and r its root. The desired representation is for each vertex v to store a label $2ecc(v)$ so that $2ecc(u) = 2ecc(v)$ iff u, v are in the same 2-edge-connected component. Observe that $E_C \subset E(T)$, since if $e \notin T$, then $G \backslash e \supset T$ is connected. Furthermore, the following holds.

Claim 20 *If u, v are in the same 2-edge-connected component, there are no cut edges on the unique u-v path in T.*

Proof. Suppose such a cut edge $e = \{u', v'\}$ exists, where u' is the end of e closer to u along the u-v path in T. Then in $G \backslash \{e\}$, the remainder of the tree path connects u to u' and v to v'. Since u, v are in the same 2-edge-connected component, u and v are connected in $G \backslash \{e\}$. Thus u' and v' are connected in $G \backslash \{e\}$, contradicting the fact that $e = \{u', v'\}$ is a cut edge of G. □

Corollary 21 *$T \backslash E_C$ is a spanning forest of the 2-edge-connected-components.*

In particular, for each 2-connected-component H, there is a subtree T_H of $T \backslash E_C$ spanning H. The idea is to label the vertices of H by the ID of the root of T_H.

Theorem 13 *There is a distributed algorithm to compute all 2-edge-connected components with probability $1 - 1/V$ in $O(\mathcal{D})$ time and using $O(E)$ messages.*

Proof. Note for a vertex v, where H denotes its 2-edge-connected component, v is the root of T_H if and only if either $v = r$, or $\{v, p(v)\}$ is a cut edge. Otherwise, v and $p(v)$ are in the same 2-edge-connected component.

First we compute the cut edges, using Theorem 7. Vertex r sets $2ecc(r)$ equal to its ID. In the following $h(T)$ rounds, for $i = 1$ to $h(T)$, for all level-i tree edges $\{v, p(v)\}$ in parallel, vertex $p(v)$ sends $2ecc(p(v))$ to v. Upon receiving this message, v sets $2ecc(v) := ID(v)$ if $\{v, p(v)\}$ is a cut edge, and $2ecc(v) := 2ecc(p(v))$ otherwise.

The labeling takes $O(h(T))$ time and $|V| - 1$ messages, and the result follows. □

Now we discuss 3-edge-connected components. In the distributed model, we can represent a subgraph (V, F) of (V, E) by using a local boolean variable for each edge. For this representation, Thurimella [23] gave a distributed connected components algorithm in $O(\mathcal{D} + \sqrt{V} \log^* V)$ time. Hence we have the following corollary to our cut pair algorithm, Theorem 7.

Corollary 22 *There is a distributed algorithm to compute all 3-edge-connected components with probability $1 - 1/V$ in $O(\mathcal{D} + \sqrt{V} \log^* V)$ time and using $O(E(\mathcal{D} + \sqrt{V} \log^* V))$ messages.*

As far as we are aware, this is the first sub-linear distributed algorithm for 3-edge-connected components.

C Circulations

Proof (of Proposition 1: circulations vanish across induced cuts). By adding up the conservation equation at all nodes in U, we obtain

$$\sum_{u \in U} \sum_{v \in \Gamma(u)} \phi(u, v) = 0. \tag{2}$$

Now notice that for each edge $\{w, w'\}$ with both endpoints in U, the terms $\phi(w, w')$ and $\phi(w', w)$ both appear in Equation (2). By antisymmetry we may cancel them out, and after we do so for all such edges, we are left with

$$\sum_{u \in U} \sum_{v \in \Gamma(u) \backslash U} \phi(u, v) = 0.$$

However this is precisely what we wanted to prove. □

Proof (of Proposition 3: circulation construction). It is easy to see that ϕ is antisymmetric. Just before line 5 of Algorithm 1 executes, all edges incident on the vertex v except for precisely $\{u, v\}$ have had their ϕ-values assigned. Furthermore, after line 5, conservation holds at v.

To show that ϕ is a circulation, it remains only to establish conservation at r. Using conservation at the other nodes, and cancelation as in the proof of Proposition 1,

$$0 = \sum_{u \in V \backslash \{r\}} \sum_{v \in \Gamma(U)} \phi(u, v) = \sum_{u \in \Gamma(r)} \phi(u, r)$$

which, due to antisymmetry, shows that conservation holds at r.

We finally need to show that ϕ is uniquely determined. To see this, note that the assignments performed by Algorithm 1 are *forced* at each step by the conservation and antisymmetry conditions. That is to say, if there were a different completion ϕ' of ϕ_0, the first ϕ-assignment performed such that $\phi(u, v) \neq \phi'(u, v)$ would prove that either antisymmetry or conservation is violated by ϕ'. □

D Cut Pairs

Proof (of Proposition 6: cut pairs are induced). Since e is not a cut edge $G \backslash \{e\}$ is connected, and so $G \backslash \{e, f\}$ must have exactly two connected components. Let U be the vertex set of one of them, and so the other is $V \backslash U$.

Note, f must be incident on both U and $V \backslash U$; indeed, otherwise $G \backslash \{e\}$ would not be connected. A similar claim holds for e; i.e., both e and f lie in $\delta(U)$. No other edges can lie in $\delta(U)$ since this would contradict the fact that U is a connected component of $G \backslash \{e, f\}$. Hence $\delta(U) = \{e, f\}$. □

The following two lemmas are used to prove Claim 13.

Lemma 23 *If C is a cycle and $U \subset V$ then $|C \cap \delta(U)|$ is even.*

Proof. As we traverse the cycle once, we enter U as many times as we exit U. Then note that $|C \cap U|$ is the total number of entrances and exits. □

Lemma 24 *If a cycle C and cut class K satisfy $K \cap C \neq \emptyset$ then $K \subseteq C$.*

Proof. Suppose that $e \in K \cap C$ but $f \in K \backslash C$. Then by Proposition 6, $\{e, f\}$ is an induced edge cut and this violates Lemma 23 since $|\{e, f\} \cap K| = 1$. □

Proof (of Claim 13). Note that K cannot contain two non-tree edges $\{e, f\}$ for then $G \backslash \{e, f\}$ would not be connected, but would also contain the spanning tree T. Hence, since $|K| > 1$ by definition, K contains at least one tree edge e. Since e is not a cut edge, $G \backslash \{e\}$ is connected, and hence there is a non-tree edge f that spans the two connected components of $T \backslash \{e\}$. The fundamental cycle of f and T thus contains e, and by Lemma 24, all of K. □

E Cut Vertices

Proof (of Claim 10). Let V_1 be the vertex set of one of the connected components of $G \backslash \{v\}$. Note that $\delta(v)$ can be partitioned into two induced edge cuts $\delta(V_1)$ and $\delta(\{v\} \cup V_1)$. By Proposition 1, the set of columns of $M^{[v]}$ corresponding to $\delta(V_1)$ adds to zero, and by Fact 9 these columns are linearly dependent. Similarly, the columns indexed by $\delta(\{v\} \cup V_1)$ are linearly dependent. So $M^{[v]}$ has at least 2 columns that are linearly dependent on the others, and the result follows. □

Proof (of Claim 11). Note that $G \backslash D$ is connected. For each fixed i, by Corollary 4, for all $e \in D$, the values of $\phi_i(e)$ are mutually independent uniformly random 0-1 variables. Further, since D is nonempty, $\sum_{e \in D} \phi_i(e)$ is, modulo 2, a uniformly random 0-1 variable.

Note that the ith-row entry in the sum of the columns indexed by D is precisely

$$\sum_{e \in D} M_{ie}^{[v]} = \sum_{e \in D} \phi_i(e),$$

which is zero (mod 2) with probability 1/2. Applying the fact that the b rows of $M^{[v]}$ are independent, we are done. □

F Distributed Cut Vertices and Pipelining

Let π be a distributed algorithm in which for each edge e, the total number of messages sent on e by π is bounded by some universal constant C_0. The messages' content may be random but the message-passing schedule must be deterministic. To *pipeline s instances of π* means to execute s instances $\{\pi_i\}_{i=1}^{s}$ of π, each one delayed by a unit time step from the previous. When multiple instances need to simultaneously send messages along the same edge we concatenate them, increasing the message sizes by a factor of at most C_0. Compared to π, pipelining adds $s-1$ to the time complexity and increases the message complexity by a factor of s.

Proof (of Theorem 8). To find all cut edges we implement Algorithm 5 distributively, obtaining probability $1/V$ of failure by Theorem 4. Theorem 6 gives an algorithm π to construct a random $O(\log V)$-bit circulation; note π sends a constant number of messages along each edge. We pipeline $s = b/\log V$ instances of π to construct a random b-bit circulation. Finally, each vertex v locally computes the rank of $M^{[v]}$ to determine if it is a cut vertex. Since π takes $O(\mathcal{D})$ rounds and sends $O(E)$ messages, and $b = O(\Delta + \log V)$, the implementation takes $O(\mathcal{D} + \Delta/\log V)$ time and $O(E(1 + \Delta/\log V))$ messages. □

G Distributed Communication Protocols

The results of this section rely on *pipelining* which is described in Appendix F. In order to describe fc-cast, we require the following subroutine, adapted from Peleg [20, §3.2].

Proposition 25 *There is a distributed algorithm* Tree-Broadcast *using $O(h(T))$ time and $O(V \cdot h(T))$ messages that sends $\mathtt{d}[v]$ to u for each $v \in V$ and each descendant u of v.*

Proof. Let π be a generic distributed algorithm that sends one message from $p(v)$ to v at time $l(v)$; in particular, π takes $O(V)$ messages, $O(h(T))$ time, and sends at most one message on each edge. Define instances $\{\pi_i\}_{i=0}^{h(t)}$ of π so that for every vertex v at level i, and for every descendant u of v, instance π_i is responsible for propagating $\mathtt{d}[v]$ to u. Each instance π_i sends empty messages for the first i rounds, and in round $t > i$, for each v with $l(v) = i$, propagates $\mathtt{d}[v]$ down the level-t tree edges descending from v. Since there are $h(T)+1$ pipelined instances and π takes $O(h(T))$ time and $O(V)$ messages, the complexity follows. □

Proof (of Theorem 9: FC-CAST*).* An fc-cast has two steps. First, we execute TREE-BROADCAST, and as a result we may assume that each vertex has a *list* of the data of all its ancestors.

In the second step, for each non-tree edge $\{v, w\}$ in parallel, v sends its list to w and vice-versa. Note that each non-tree edge e can determine its fundamental cycle with T by comparing its endpoints' lists. (More precisely, either endpoint of e can determine such.) Each list has at most $1+h(T)$ items, each of which is $O(\log V)$ bits long and can be sent in a single message, so both steps in the fc-cast take $O(h(T))$ time. The message complexity of the second step as just described is $O(E \cdot h(T))$, but now we give a refinement that achieves $O(\min\{E \cdot h(T), V^2\})$ message complexity.

The essential idea is for all $u, v \in V$, we want to avoid sending $\mathtt{d}[u]$ to v more than once. Implement the second step of the fc-cast so that each vertex v sends one $\mathtt{d}[\cdot]$ value per round, and in the order $\mathtt{d}[v]$ first, then $\mathtt{d}[p(v)]$, etc., with the data of the root last. When a vertex u receives $\mathtt{d}[x]$ for the second time for some x, u asks the sender to stop sending its list. Likewise, if u receives $\mathtt{d}[x]$ from multiple neighbors at the same time, u asks all but one to stop sending their lists. Along each edge, at most one redundant message and one stop request can be sent in each direction. There can only be $|V|^2$ non-redundant messages; hence the total number of messages sent in this step is $O(V^2 + E)$. Considering the tree-broadcast as well, the total message complexity is $O(V \cdot h(T) + \min\{E \cdot h(T), V^2 + E\}) = O(\min\{E \cdot h(T), V^2\}$ as claimed. □

Proof (of Proposition 14: CONVERGE-CAST*).* Let π be a generic distributed algorithm that sends one message from v to $p(v)$ at time $1 + h(T) - l(v)$; in particular, π takes $O(V)$ messages, $O(h(T))$ time, and sends at most one message on each edge. Define instances $\{\pi_i\}_{i=0}^{h(t)}$ of π so that for every vertex v at level i, instance π_i is responsible for propagating $\bigvee_{u \in desc(v)} \mathtt{w}[u, v]$ to v.

We implement π_i as follows. For $v' \in desc(v)$, define $x[v', v] := \bigvee_{u \in desc(v')} \mathtt{w}[u, v]$. Each level-$h(T)$ vertex v' can immediately compute $x[v', v]$ for all its ancestors v. In $h(T)$ rounds, for j from $h(T)$ down to 1, for each vertex v' at level j and each ancestor v of v' at level i, v' computes $x[v', v]$ and sends $x[v', v]$ to $p(v')$. Observe that

$$x[v', v] = \mathtt{w}[v', v] \vee \bigvee_{v'' \text{ a child of } v'} x[v'', v].$$

Hence, $x[v', v]$ can be computed by v' by taking the OR of $\mathtt{w}[v', v]$ and all values sent up to v' from its children in the previous round.

Since there are $h(T) + 1$ pipelined instances and π takes $O(h(T))$ time and $O(V)$ messages, the complexity follows. □

H Distributed Cut Pairs

Proof (of Claim 15). Since $G \backslash \{e\} \supset T$ is connected, we may assume e is not a cut edge.

If e is not in any cut pair, then by $(\star)$, $\phi(e) \neq \phi(f)$ for every $f \neq e$, so $\phi(e)$ occurs once in Φ_e.

If e is in a cut pair $\{e, f\}$, Claim 13 implies that $\{e, f\} \in C_e$ (because no fundamental cycle but C_e contains e). By $(\star)$ $\phi(e) = \phi(f)$. Since $\{e, f\} \subset C_e$, $\phi(e)$ occurs at least twice in Φ_e. □

Proof (of Claim 16). If $\{v, p(v)\}$ is a cut edge, then $\{v, p(v)\}$ is not contained in any cycle, so $\{v, p(v)\} \in C_e$ cannot hold for any e. If $\{v, p(v)\}$ is not a cut edge and does not lie in any cut pair, then by $(\star)$, $\phi(v, p(v))$ cannot occur multiple times in any Φ_e. Hence, $\mathtt{w}[u, v]$ is false for all u.

If $\{v, p(v)\}$ lies in some cut pair, then by Claim 13 there is some non-tree edge e so that C_e contains the cut class of $\{v, p(v)\}$. Let u be either endpoint of e; since $\{v, p(v)\} \in C_e$, u is indeed a descendant of v. Due to $(\star)$, $\mathtt{w}[u, v] =$ TRUE. □