Fast Computation of Small Cuts via Cycle Space Sampling[*]

David Pritchard and Ramakrishna Thurimella

September 1, 2009

Abstract

We describe a new sampling-based method to determine cuts in an undirected graph. For a graph (V, E), its cycle space is the family of all subsets of E that have even degree at each vertex. We prove that with high probability, sampling the cycle space identifies the cuts of a 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.

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 $\mathcal{D}$, and the maximum degree by Δ. We obtain simple $O(\mathcal{D})$-time distributed algorithms to find all cut edges, 2-edge-connected components, and cut pairs, matching or improving upon previous time bounds. Under natural conditions these new algorithms are *universally optimal* — i.e. a $\Omega(\mathcal{D})$-time lower bound holds on every graph. We obtain a $O(\mathcal{D} + \Delta/\log |V|)$-time distributed algorithm for finding cut vertices; this is faster than the best previous algorithm when $\Delta, \mathcal{D} = O(\sqrt{|V|})$. A simple extension of our work yields the first distributed algorithm with *sub-linear* time for 3-edge-connected components. The basic distributed algorithms are Monte Carlo, but they can be made Las Vegas without increasing the asymptotic complexity.

In the model of *parallel computing* on the EREW PRAM our approach yields a simple algorithm with optimal time complexity $O(\log V)$ for finding cut pairs and 3-edge-connected components.

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 identify cuts by sampling the cycle space. For a graph (V, E) we say that $\phi \subseteq E$ is a *binary circulation* if every vertex has even degree in (V, ϕ); the *cycle space* of graph (V, E) is the set of all its binary circulations. For $S \subseteq V$, 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; cut edges and cut pairs are induced edge cuts[1]. The family of all induced edge cuts is called the *cut space* of a graph. The cycle space and

[*] A preliminary version of this work appeared as "Fast Distributed Computation of Cuts via Random Circulations" at ICALP 2008 [31].

[1] Our convention is that $\delta(\emptyset) = \delta(V) = \emptyset$ is an induced edge cut — so we don't in general assume $\delta(S)$ is a cut.

cut space are orthogonally complementary vector subspaces of $\mathbb{Z}_2^E$ (see Section 2), which implies that the intersection of any binary circulation and induced edge cut is of even size. At a high level, our algorithms depend on a probabilistic converse (Proposition 5): if $F \subset E$ is *not* an induced edge cut, the number of edges of F intersecting a uniformly random binary circulation is even with probability exactly 1/2. This specific observation seems new, although it is a simple consequence of standard results on the cut and cycle spaces. To make use of this observation we give efficient algorithms to sample a uniformly random binary circulation in the sequential, parallel, and distributed models of computing.

The Distributed Model. Our approach improves several known time bounds in the *distributed* computing model with *congestion*. The precise model, denoted $\mathcal{CONGEST}$ (e.g. by Peleg [29, §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. $\mathcal{D} := \max_{u,v \in V} \mathrm{dist}_G(u, v)$. 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 [24] 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[2]. 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).

A distributed time complexity faster than $\Theta(V)$ on some graphs is called *sub-linear*. Determining whether a task in this model can be accomplished in sub-linear time, or better yet $O(\mathcal{D})$ time, is a fundamental problem. E.g. one breakthrough was a sub-linear MST algorithm [12] which was later improved [23] to time complexity $O(\mathcal{D} + \sqrt{|V|} \log^* |V|)$ — here $\log^* x$ is the number of times which log must be iteratively applied to x before obtaining a number less than 1. Our breakthroughs in this regard are $O(\mathcal{D})$ time algorithms for cut pairs, cut edges, and 2-edge-connected components, and a sub-linear algorithm for 3-edge-connected components.

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 [33], cut edges [33], and cut vertex-pairs (cuts $C \subseteq V$ with $|C| = 2$) [17]. These algorithms are based on depth-first search (DFS). Galil & Italiano [11], 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 [5]. **Cut Edges/Vertices.** Two early distributed algorithms for cut edges and vertices, by Ahuja & Zhu [1] and Hohberg [16], 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 [18] gave a non-DFS-based algorithm with $\Theta(V)$ time complexity. The first sub-linear distributed algorithms for any type of small cuts appear in Thurimella [36]; using an MST subroutine, he obtained time complexity $O(\mathcal{D} + \sqrt{V} \log^* V)$ for both cut edges and cut vertices. **Cut Pairs.** For cut pairs, Jennings and

[2] 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.

Motyckova [20] gave a distributed algorithm with worst-case time and message complexity $\Theta(n^3)$, and Tsin [38] 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 [12]. However, as Thurimella's algorithm [36] showed, there are some graphs on which $\Theta(V)$ time is not asymptotically optimal. The stronger term *universally optimal* [12] applies to algorithms which, on *every* graph, have running time within a constant factor of the minimum possible.

Parallel. In the PRAM model, an optimal $O(\log V)$-time and $O(V + E)$-work Las Vegas algorithm for cut edges and vertices was obtained by Tarjan & Vishkin [35] (provided that for spanning forests, recent work of Halperin & Zwick [14] is used). For cut pairs, it may be possible to combine a a 3-vertex-connectivity algorithm of Fussell, Ramachandran & Thurimella [10] with the reduction of Galil & Italiano [11] (and spanning forest routines from [14]) to yield a time- and work-optimal EREW algorithm. This is mentioned as a "future application" in Halperin & Zwick [14]. However, this approach appears not to have been fully analyzed and is fairly complicated.

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 at most $1/V$ 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 [37]) or open ear decomposition (e.g., see references in Fussell et al. [10]).

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 [36]. 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 a Las Vegas algorithm for cut pairs and 3-edge-connected components with time complexity $O(\log V + T(E))$, space complexity $O(E + S(E))$, and work complexity $O(E + W(E))$, where $T(n), S(n), W(n)$ are respectively the time, space, work complexity to sort n numbers of length $O(\log n)$ bits. E.g. on the EREW PRAM, we can implement our algorithm in $O(\log V)$ time, $O(E)$ space and $O(E \log E)$ work using a sorting subroutine of Kruskal, Rudolph and Snir [22], or in $O(\log V)$ time, $O(E^{1+\epsilon})$ space and $O(E\sqrt{\log E})$ work using a subroutine of Han and Shen [15].

[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	$O(V)$	$O(E)$
[36]	'95	Vertices & Edges	$O(\mathcal{D}+\sqrt{V}\log^* V)$	$O(E\cdot(\mathcal{D}+\sqrt{V}\log^* V))$
[38]	'06	Pairs	$O(V+\mathcal{D}^2)$	$O(E+V\cdot\mathcal{D})$
Theorem 6†		Edges	$O(\mathcal{D})$	$O(E)$
Theorem 9†		Pairs	$O(\mathcal{D})$	$O(\min\{V^2, E\cdot\mathcal{D}\})$
Theorem 7†		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 Organization of the Paper

In Section 2 we define random circulations and show how to construct them efficiently. In Section 3 we show how random circulations yield algorithms for small cuts and give sequential implementations. In Section 4 we precisely define the assumptions in our distributed model and give the Monte Carlo algorithms; we introduce a technique called *fundamental cycle-cast* which may be of independent interest. In Section 5 we discuss 2- and 3-edge-connected components. In Section 6 we give the Las Vegas analogues of our distributed algorithms. We give $\Omega(\mathcal{D})$ distributed time lower bounds under precise assumptions in Section 7. We give the parallel cut pair algorithm in Section 8.

2 Preliminaries on Circulations

Results on the cut space and cycle space over $\mathbb{Z}$ in directed graphs goes back to the 1970s (e.g., [4]) but for our purposes it is convenient to work modulo 2; informally, the results then apply to undirected graphs since $+1 \equiv -1 \pmod 2$. For the sake of completeness, we prove the needed results. See also Diestel [7] which proves material equivalent to Propositions 1, 2, and 3.

For notational convenience we identify any subset S of E with its *characteristic vector* $\chi^S \in \mathbb{Z}_2^E$ defined by $\chi^S_e = 1$ for $e \in S$ and $\chi^S_e = 0$ for $e \notin S$. We use $\oplus$ to stand for vector addition modulo 2, so in accordance with our notational convention, for $S, T \subset E$ the expression $S \oplus T$ denotes the symmetric difference of S and T.

As mentioned earlier, $\phi \subseteq E$ is a *binary circulation* if in (V, ϕ) every vertex has even degree; the *cycle space* of graph (V, E) is the set of all its binary circulations; $\delta(S)$ denotes the edges of G with exactly one end in S; an *induced edge cut* is a set of the form $\delta(S)$ for some S; and the family of all induced edge cuts is called the *cut space* of a graph. For $v \in V$ we use $\delta(v)$ as short for $\delta(\{v\})$.

Proposition 1. *The cut space and cycle space are vector subspaces of $\mathbb{Z}_2^E$.*

Proof. Note it suffices to show each space contains $\varnothing$ and is closed under $\oplus$. For the cut space, this holds since $\delta(\varnothing) = \varnothing$ and $\delta(S \oplus T) = \delta(S) \oplus \delta(T)$. For the cycle space, clearly $(V, \varnothing)$ has even degree at each vertex; and if (V, ϕ_1) and (V, ϕ_2) have even degree at each vertex, then the degree of vertex v in $(V, \phi_1 \oplus \phi_2)$ is $\deg_{\phi_1}(v) + \deg_{\phi_2}(v) - 2\deg_{\phi_1 \cap \phi_2}(v) \equiv 0 + 0 - 0 \pmod 2$, so $\phi_1 \oplus \phi_2$ is a binary circulation. □

Proposition 2. *The cut space and cycle space are orthogonal.*

Proof. We need precisely to show that for any binary circulation ϕ and any $S \subset V$ that the dot product $\phi \cdot \delta(S) \equiv 0 \pmod 2$, or equivalently that $|\phi \cap \delta(S)|$ is even. Now $\sum_{s \in S} \deg_\phi(s) = \sum_{s \in S} |\phi \cap \delta(s)|$ and the former quantity is even since ϕ is a circulation. The latter sum counts every edge of $\phi \cap \delta(S)$ once, every edge of ϕ with both ends in S twice, and every other edge zero times. Since this sum is even, $|\phi \cap \delta(S)|$ is even. □

In the next proposition, we assume G is connected, and hence has a spanning tree T. We need to define the *fundamental cuts* and *fundamental cycles* of T. For each edge e of $E \backslash E(T)$, we define the *fundamental cycle* C_e to be the unique cycle in $T \cup \{e\}$. Note cycles are binary circulations. For each edge e of T, we define S_e to be one of the two connected components of $T \backslash e$, and define the *fundamental cut of* e to be $\delta(S_e)$ (note $\delta(S_e)$ does not depend on which connected component we chose).

Proposition 3. *(a) The cut space and cycle space are orthogonal complements. (b) The cycle space has dimension* $|E| - |V| + 1$ *and the cut space has dimension* $|V| - 1$*. (c) For any spanning tree* T *of* G*, its fundamental cycles form a basis of the cycle space, and its fundamental cuts form a basis of the cut space.*

Proof. We will show that the $|E| - |V| + 1$ fundamental cycles are linearly independent in the cycle space and the $|V| - 1$ fundamental cuts are linearly independent in the cut space. Basic linear algebra shows the sum of the dimensions of two orthogonal subspaces of $\mathbb{Z}_2^E$ is at most $|E|$, with equality only if they are orthogonal complements, thus by Proposition 2, Proposition 3(a) and (b) follow, and so does (c). We use the following claim.

Claim 4. *Let* $H \subset E$ *and consider a family of vectors* $\{x^e\}_{e \in H}$ *over* $\mathbb{Z}_2^E$*. If* $x^e_e = 1$ *for all* $e \in H$*, and* $x^e_f = 0$ *for all distinct* $e, f \in H$*, then* $\{x^e\}_{e \in H}$ *is linearly independent.*

Proof. Suppose for the sake of contradiction that $\bigoplus_{e \in H} a_e x^e$ is the zero vector, where $a_e \in \{0, 1\}$ for each e and not all a_e are zero. Pick f such that $a_f = 1$, then $\sum_{e \in H} a_e x^e_f = 1$, a contradiction. □

Note that $e \in C_e$ but for any other edge f of $E \backslash E(T)$, $f \notin C_e$, so by Claim 4 with $H = E \backslash E(T)$ and $x^e = C_e$, these vectors are linearly independent. Note that $e \in \delta(S_e)$ but for any other edge f of T, $f \notin \delta(S_e)$, so by Claim 4 with $H = E(T)$ and $x^e = \delta(S_e)$, these vectors are linearly independent. This completes the proof of Proposition 3. □

2.1 Random Circulations

Next we show why uniform sampling of the cycle space is useful for identifying cuts.

Proposition 5. *Let* $F \subset E$ *be a set that is not an induced edge cut. If* ϕ *is a uniformly random binary circulation, then* $\Pr[|F \cap \phi|$ *is even*$] = 1/2$.

Proof. Since F is not in the cut space, by Proposition 3(a) it is not orthogonal to the cycle space, i.e. there is a binary circulation ϕ_F with $|F \cap \phi_F|$ odd. Now we pair up each binary circulation ψ on G with the binary circulation $\psi' := \psi \oplus \phi_F$. This yields a pairing of all binary circulations on G since for all ψ, $\psi' \neq \psi$ and $\psi'' = \psi$. Modulo 2, $|F \cap \psi'| \equiv |F \cap \psi| + |F \cap \phi_F| \equiv |F \cap \psi| + 1$, so in each pair, exactly one of the two binary circulations has even intersection with F. Thus, exactly half of all binary circulations have even intersection with F, which proves the result. □

Next we give a method for constructing binary circulations (it is implicit in [4, Ex. 12.1.1]). Given a spanning tree T and subset ψ of $E \backslash E(T)$, we say that ϕ is a *completion* of ψ if ϕ is a binary circulation and $\phi \cap (E \backslash E(T)) = \psi$.

Proposition 6. *For any* $\psi \subseteq E \backslash E(T)$*,* ψ *has a unique completion* ϕ*.*

Proof. First, we give a succinct proof sketch. By Proposition 3(c) the cycle space can be expressed as $\{\bigoplus_{e \in E \backslash E(T)} a_e C_e | a \in \mathbb{Z}_2^{E \backslash E(T)}\}$. For which a does this yield a completion of ψ? From the observations in the proof of Proposition 3, for $f \in E \backslash E(T)$, the coordinate of $\bigoplus_{e \in E \backslash E(T)} a_e C_e$ at index f is just a_f, hence the unique completion of ψ is the one in which a is the indicator vector of ψ, i.e. the unique completion is $\phi = \bigoplus_{e \in \psi} C_e$. Explicitly, for $f \in T$, we have $f \in \phi$ iff f appears in an odd number of the fundamental cycles $\{C_e \mid e \in \psi\}$. This completes the proof, but we now give a second, algorithmic proof, which is needed later.

For a leaf node v incident to $e \in E(T)$, since the degree of (V, ϕ) at v must be even, notice that we must have $e \in \phi$ if $|\psi \cap \delta(v)|$ is odd, and $e \notin \phi$ if $|\psi \cap \delta(v)|$ is even. Iterating this argument on $T \backslash v$ yields Algorithm 1, which will show constructs the unique completion of ψ.

Algorithm 1 Given G, T and $\psi \subset E\backslash E(T)$, construct binary circulation ϕ such that $\phi\backslash E(T) = \psi$.

```
Initialize $\phi := \psi, S := T$          ▷ $S$ is the subtree of $T$ where $\phi$ is not yet defined
while $S$ has any edges,
    Let $v$ be any leaf of $S$ and $e$ be the unique incident edge of $v$ in $S$
    if $|\delta(v) \cap \phi|$ is odd then $\phi := \phi \cup \{e\}$          ▷ Satisfy degree constraint at $v$
    Delete $v$ from $S$
Output $\phi$
```

See Figure 1 for an illustration of Algorithm 1. Now we prove Proposition 6 using Algorithm 1. It is clear that every vertex of (V, ϕ) has even degree except possibly the last vertex left in S. However, by the handshake lemma, no graph can have exactly one vertex of odd degree, so ϕ is indeed a binary circulation. To show uniqueness, suppose for the sake of contradiction that ψ has two distinct completions ϕ, ϕ'. Then $\phi \oplus \phi' \subset E(T)$, and as such the nonempty forest $\phi \oplus \phi'$ has at least one vertex of degree 1. This contradicts the fact that $\phi \oplus \phi'$ is a binary circulation. □

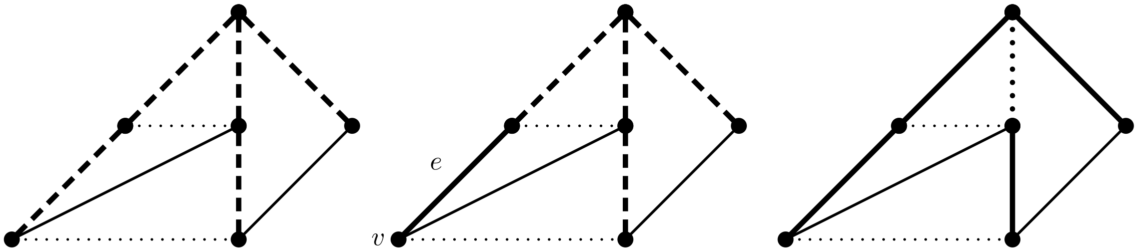

Figure 1: Completing a binary circulation. The spanning tree T is given by thick edges. Solid edges are in the circulation, dotted edges will not be in the circulation, and dashed edges are undecided. Left: the initial value of ϕ (which equals ψ). Middle: we ensure a leaf vertex v has even degree. Right: repeating the previous step yields the completed circulation ϕ.

We now give the method for constructing uniformly random binary circulations, illustrated in Algorithm 2: pick a uniformly random subset of $E\backslash E(T)$ and then compute its completion.

Algorithm 2 Given G and spanning tree T, output a uniformly random binary circulation.

```
for each $e$ in $E\backslash E(T)$, put $e$ in $\psi$ with independent probability 1/2
Return the completion of $\psi$, using Algorithm 1
```

Theorem 1. *Algorithm 2 outputs a uniformly random binary circulation.*

Proof. By Proposition 3(b) the cycle space contains exactly $2^{|E|-|V|+1}$ elements. Algorithm 2 makes one of $2^{|E|-|V|+1}$ choices of ψ each with probability $2^{-|E|+|V|-1}$, and each distinct choice of ψ leads to a distinct binary circulation. □

To increase the probability of identifying a particular cut beyond 1/2, our algorithms will sample multiple independent random circulations. For this reason it is convenient to introduce notation that incorporates multiple circulations into a single object. 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 7. *$\phi : E \to \mathbb{Z}_2^b$ is a b-*bit circulation *if for each $1 \leq i \leq b$, $\{e \mid \phi_i(e) = 1\}$ is a binary circulation.*

Hence, to say that ϕ is a uniformly random b-bit circulation is the same as saying that $\{\phi_i\}_{i=1}^b$ are mutually independent, uniformly random binary circulations. For brevity, we use the phrase *random b-bit circulation* to stand for "uniformly random b-bit circulation" in the rest of the paper. Let $\mathbf{0}$ denote the all-zero vector and $\oplus$ denote addition of vectors modulo 2. Using Proposition 2 and Proposition 5 we obtain the following corollary.

Corollary 8. *Let ϕ be a random b-bit circulation and $F \subseteq E$. Then*

$$\Pr\left[\bigoplus_{e \in F} \phi(e) = \mathbf{0}\right] = \begin{cases} 1, & \textit{if } F \textit{ is an induced edge cut;} \\ 2^{-b}, & \textit{otherwise.} \end{cases}$$

To generate a random b-bit circulation, it suffices to modify Algorithms 1 and 2 slightly so as to operate independently on each of b positions at once: on Line 1 of Algorithm 2 we set $\phi(e)$ to a uniformly independent b-bit string, and on Line 4 of Algorithm 1 we set $\phi(e) := \bigoplus_{f \in \delta(v) \setminus e} \phi(f)$. We denote the resulting algorithm by Rand-b-Bit-Circ and illustrate it in Figure 2. Under the standard assumption that the machine word size is $\Theta(\log V)$, the running time of Rand-b-Bit-Circ in the sequential model of computing is $O(E\lceil \frac{b}{\log V} \rceil)$.

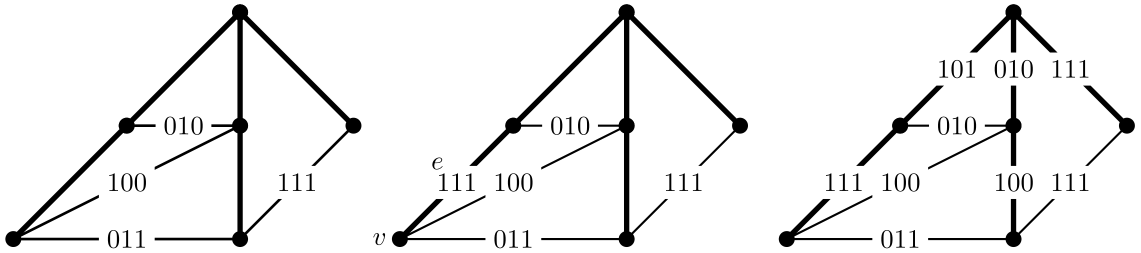

Figure 2: Constructing a random 3-bit circulation; thick edges are tree edges and thin edges are non-tree edges. Left: we assign random ϕ values to the non-tree edges. Middle: we set $\phi(e) := \bigoplus_{f \in \delta(v) \setminus e} \phi(f)$ for a leaf vertex v. Right: repeating the previous step yields the completed circulation ϕ.

3 Basic 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.

3.1 Finding All Cut Edges

We provide pseudocode in Algorithm 3 and then prove its correctness. It is based on the easy fact that e is a cut edge if and only if $\{e\}$ is an induced edge cut, which we state without proof.

Algorithm 3 Given a connected graph G, compute the cut edges of G.

```
1: Let $b = \lceil \log_2 VE \rceil$ and let $\phi$ be a random $b$-bit circulation on $G$.
2: Output all edges $e$ for which $\phi(e) = \mathbf{0}$
```

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

Proof. Using the fact above, Corollary 8, and a union bound, the probability of error is at most $E/2^b \leq 1/V$. The subroutine Rand-b-Bit-Circ as well as Line 2 of Algorithm 3, each take $O(E)$ sequential time. □

3.2 Finding All Cut Pairs and Cut Classes

Proposition 9, whose easy proof we omit, leads to our approach for finding cut pairs.

Proposition 9 (Cut pairs are induced)**.** *Let e and f be edges that are not cut edges. Then $\{e, f\}$ is a cut pair if and only if $\{e, f\}$ is an induced edge cut.*

With Corollary 8 we immediately obtain the following.

Corollary 10. *Let e, f be two distinct edges that are not cut edges. Then $\Pr[\phi(e) = \phi(f)] = 1$ if $\{e, f\}$ is a cut pair, and 2^{-b} otherwise.*

This yields a cute probabilistic proof of the following basic fact.

Corollary 11 (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 1-bit circulation on G. By Corollary 10, $\phi(e) = \phi(f)$ and $\phi(f) = \phi(g)$. So $\phi(e) = \phi(g)$ with probability 1. By Corollary 10, $\{e, g\}$ must be a cut pair. □

Definition 12. *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.*

We illustrate a cut class in Figure 3. Note the cut class has a natural cyclic order.

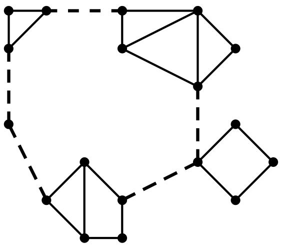

Figure 3: A graph is shown with one cut class highlighted using dashed edges. Deleting any two dashed edges disconnects the graph.

Corollary 11 implies that any two distinct cut classes are disjoint. Hence, even though there may be many cut pairs, we can describe them all compactly by listing all cut classes of the graph. We now give our simple linear-time algorithm to find all cut classes, with pseudocode given in Algorithm 4.

Algorithm 4 Given a connected graph G, compute the cut classes of G.

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

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

Proof. There are $|E|$ edges and the analysis in Section 3.1 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 Corollary 10 shows that $\Pr[\phi(e) = \phi(f)] \leq 1/2^b$ for each such pair. Hence, by a union bound, 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 has time complexity $O(E)$. It remains to implement Line 2 of Algorithm 4 in $O(E)$ time. To do this, we sort all edges e according to the key $\phi(e)$ using 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 [6, §9.3] — then the sort takes $O(E)$ time. □

3.3 Finding All Cut Vertices

The following characterization of cut vertices underlies our approach.

Proposition 13. *The cut $\delta(v)$ properly contains a nonempty induced edge cut if and only if v is a cut vertex.*

Proof. First, suppose v is a cut vertex. Let V_1 be the vertex set of one of the connected components of $G \backslash \{v\}$. Then $\delta(v)$ properly contains the nonempty induced edge cut $\delta(V_1)$.

Second, suppose v is not a cut vertex, so there is a spanning tree T' of $G\backslash\{v\}$. Suppose $S \subset V$ has $\delta(S) \subseteq \delta(v)$. Without loss of generality (by complementing S if necessary) we assume $v \in S$. Since no edges of T' are in $\delta(S)$, S either contains all of $V\backslash\{v\}$ or none of $V\backslash\{v\}$. Thus either $S = V$ in which case $\delta(S)$ is empty, or $S = \{v\}$, in which case $\delta(S)$ is not a proper subset of $\delta(v)$. □

Using Proposition 13, the essential idea in our approach to find cut vertices is to detect for each vertex v whether $\delta(v)$ properly contains any nonempty induced edge cuts. As usual we detect induced edge cuts via Corollary 8, this time rephrasing the detection problem as one of finding linearly dependent rows of a matrix. Hence we need the following fact, when $\mathbb{Z}_2$ is viewed as a field.

Fact 14. *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* $(\bmod 2)$.

Our approach works as follows. Note — it does not have a very efficient sequential implementation, but yields an efficient distributed algorithm. 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.

Claim 15. *If v is a cut vertex then* $\text{rank}(M^{[v]}) \leq d(v) - 2$.

Proof. 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 Corollary 8 the set of columns of $M^{[v]}$ corresponding to $\delta(V_1)$ adds to zero, and by Fact 14 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. □

Claim 16. *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* $(\bmod 2)$ *is* 2^{-b}.

Proof. By Proposition 13, D is not an induced edge cut, and the result follows from Corollary 8. □

Next we show that for $b = \lceil \Delta + 2\log_2 V \rceil$, it is very likely that $\text{rank}(M^{[v]}) < d(v) - 1$ iff v is a cut vertex. Thus our approach, with pseudocode given in Algorithm 5, is correct with high probability.

Algorithm 5 Given a connected graph G, compute the cut vertices of G.

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

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

Proof. Claim 15 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 16, Fact 14, 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 at least $1 - V^{-2}$, we have $\text{rank}(M^{[v]}) \geq |D| = d(v) - 1$ and v is not output. By another union bound, the probability that any vertex is misclassified by Algorithm 5 is at most $V/V^2 = 1/V$. □

4 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, 36, 38] for small cuts. Nonetheless, these assumptions can be removed at a cost if desired, e.g. using the synchronizer of Awerbuch and Peleg [3] at a polylog(V) factor increase in complexity, Peleg's [28] $O(\mathcal{D})$-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 [29, §5.1]), as follows.

Proposition 17. *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.

4.1 Random Circulations and Cut Edges

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

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

Proof. We implement Rand-b-Bit-Circ distributively. The size bound ensures that b-bit strings can be sent in a message. We compute a BFS tree T, using Proposition 17. Then for each non-tree edge $\{e\}$ in parallel, the endpoint with the higher ID picks a random b-bit value for $\phi(e)$ and sends it to the other endpoint. In the following $h(T)$ rounds, for $i = h(T)$ down to 1, each level-i vertex computes $\phi(\{v, p(v)\}) := \bigoplus_{f \in \delta(v) \setminus \{v, p(v)\}} \phi(f)$ and sends this value to $p(v)$. The complexity is $O(\mathcal{D} + h(T)) = O(\mathcal{D})$ time and $O(E + E)$ messages. □

Theorem 5 yields our distributed cut edge algorithm.

Theorem 6. *There is a distributed algorithm to compute all cut edges with probability at least $1 - 1/V$ 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 $k = VE$, we use Theorem 5 to compute a random k-circulation in the required complexity bounds. Then we identify e as a cut edge if $\phi(e) = 0$. □

4.2 Pipelining and Cut Vertices

Our cut vertex algorithm requires a circulation on $\Theta(\Delta + \log V)$ bits, and in order to construct such a circulation efficiently, we use a *pipelining* technique. 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.

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

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

Proof. We implement Algorithm 5 distributively, obtaining probability $1/V$ of failure by Theorem 4. Let $b = \lceil \Delta + 2\log_2 V \rceil$. Theorem 5 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 $b/\log V$ instances of π to construct a random b-bit circulation. Then, 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. □

4.3 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. We assume that $\mathtt{d}[v]$ includes the ID, level, and parent ID of v, since this information can be appended to $\mathtt{d}[v]$ while increasing its length by at most $O(\log V)$ bits. 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.

Theorem 8. *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.*

As a subroutine, we need a tree broadcast subroutine adapted from Peleg [29, §3.2].

Proposition 18. *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 8. 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. □

We can implement fc-cast in $O(h(T))$ time with message complexity even smaller than $\min\{E \cdot h(T), V^2\}$ using a nearest common ancestor labeling scheme of Alstrup et al. [2]. We only sketch the idea since the precise improved complexity is somewhat awkward to state (seemingly cannot be expressed in terms of parameters $V, E, \Delta, h(T)$) and does not seem universally optimal. If uw is a edge not in T, call w a *non-tree neighbour* of u and vice-versa. The general idea behind the optimized implementation is that, while the implementation in Theorem 8 sends $\mathtt{d}[v]$ to each descendant of v and each non-tree neighbour of a descendant of v, we can actually send $\mathtt{d}[v]$ to a smaller subset of these nodes while meeting the definition of a fundemental cycle-cast.

In more detail, the scheme of Alstrup et al. [2] gives each vertex an $O(\log V)$-bit label such that given *just the labels* of any two nodes, we can also compute the label of their *nearest common ancestor* (with a deterministic algorithm independent of T). Alstrup et al. do not work in any specific distributed model, but their scheme is built out of standard primitives like the number of descendants of a given node, and as such can be implemented in the model we consider in $O(h(T))$ time and $O(E)$ messages. The first step of our new implementation is to compute these labels. Then, in unit time and $2|E|$ messages, we have each node inform each of its neighbours of its label.

At a high level, the labeling scheme allows the implementation to be optimized as follows. In the first step we send $\mathtt{d}[v]$ down to its descendant u only if there is some fundamental cycle containing both u and v; in the second step each v asks for $\mathtt{d}[\cdot]$ values from its non-tree neighbours in such a way that u receives each $\mathtt{d}[\cdot]$ value at most once, and only asks for $\mathtt{d}[v]$ from w if C_{uw} contains v. Implementing these steps requires that nodes have some knowledge about the relative position of their neighbours in the tree, which is accomplished using the labels. There are some slightly complicated details in implementing the first step, for which a pipelined *convergecast* (see Proposition 21) suffices.

4.4 Distributed Cut Pair Algorithm

When computing the cut pairs, it helps if we assume that G has no cut edges, i.e. G is 2-edge-connected. To make this assumption without loss of generality, for our input graph G, we compute the set E_C of cut edges using Theorem 11 and then report the cut pairs of the *2-edge-connected components*, which are the connected components of $G \backslash E_C$ (we elaborate in Section 5). It is straightforward to show that the cut pairs of G are the cut pairs of these components, that each component has no cut edge, and that no component has diameter greater than G.

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 claims are used to relate fundamental cycles to cut classes. (The first is fairly intuitive given Figure 3 on page 8.)

Lemma 19. *If a cycle C and a 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 9, $\{e, f\}$ is an induced edge cut. But then $|\{e, f\} \cap C| = 1$, contradicting Proposition 2 (the orthogonality of the cut space and cycle space). □

Claim 20. *Let K be a cut class. Then $K \subset C_e$ for some $e \in E \backslash E(T)$.*

Proof. First we claim K contains at most one non-tree edge. Suppose otherwise, for the sake of contradiction, that K contains two non-tree edges $\{e, f\}$. Then $\{e, f\}$ is a cut pair and so $G \backslash \{e, f\}$ is not connected. However, this contradicts the fact that $G \backslash \{e, f\}$ contains the spanning tree T.

The definition of a cut class implies $|K| > 1$, so 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 connects the two connected components of $T\backslash\{e\}$. The fundamental cycle C_f of f and T thus contains e, and by Lemma 19, all of K. □

To describe our cut pair algorithm we introduce a variant of a standard technique, the *convergecast* (e.g., see Peleg [29, §4.2]). Informally, it allows each node to independently query its descendants. In this paper we take the convention that v is always a descendant of itself. Let $\mathit{Desc}(v)$ denote the set of v's descendants. For each $v \in V$, and each $u \in \mathit{Desc}(v)$, let $\mathtt{w}[u, v]$ be a variable of length $\Theta(\log V)$ stored at u.

Proposition 21. *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* $\max\{\mathtt{w}[u, v] \mid u \in \mathit{Desc}(v)\}$.

Proof. We assume some familiarity with the basic implementation of convergecast in order to gloss over some basic details; see [29, §4.2]. We use π to represent a generic distributed algorithm that sends messages from leaves to the root in level-synchronized fashion. The "standard" convergecast uses π to compute $\max\{\mathtt{w}[u, r] \mid u \in V\}$ at r; in round i, for i from $h(T)$ down to 1, every level-i node passes up the largest value that it knows about to its parent. A slight modification yields instances $\{\pi_i\}_{i=0}^{h(t)}$ of π so that for every vertex v at level i, instance π_i propagates $\max\{\mathtt{w}[u, v] \mid u \in \mathit{Desc}(v)\}$ to v. Since there are $h(T)+1$ pipelined instances and π takes $O(h(T))$ time and $O(V)$ messages, the complexity follows. □

Theorem 9. *There is a distributed algorithm to compute all cut classes with probability at least $1 - 1/V$ in $O(\mathcal{D})$ time and using $O(\min\{E \cdot \mathcal{D}, V^2\})$ messages.*

Proof. As in Algorithm 4, for $b = \lceil \log_2(VE^2) \rceil$ we compute a random b-bit circulation ϕ on G, using Theorem 5. Denote the following assumption by $(\star)$.

$$\text{For all edges } e, f,\ \phi(e) = \phi(f) \text{ if and only if } \{e, f\} \text{ is a cut pair.} \qquad (\star)$$

By the analysis in the proof of 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, which follows immediately from Claim 20, lets each non-tree edge determine if it is a member of any cut pair.

Claim 22. *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 $u \in \mathit{Desc}(v)$, define

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

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

Claim 23. *Tree edge $\{v, p(v)\}$ is in a cut pair if and only if $\exists u \in \mathit{Desc}(v)$ such that $\mathtt{w}[u, v] > 0$.*

Proof. If $\{v, p(v)\}$ lies in a cut pair then by Claim 20 there is a fundamental cycle C_e containing that cut pair. It is easy to see that one endpoint u of e is a descendant of v and has $\mathtt{w}[u, v] > 0$. □

By Proposition 21, after the convergecast, each tree edge can use Claim 23 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. □

5 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 24. *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 $\tau(v)$ so that $\tau(u) = \tau(v)$ 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 25. *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 26. *$T \backslash E_C$ is a spanning forest of the 2-edge-connected components.*

In particular, for each 2-edge-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 10. *There is a distributed algorithm to compute all 2-edge-connected components with probability at least $1 - 1/V$ 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 is the root r of T, 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 6. Vertex r sets $\tau(r)$ 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 $\tau(p(v))$ to v. Upon receiving this message, v sets $\tau(v) := ID(v)$ if $\{v, p(v)\}$ is a cut edge, and $\tau(v) := \tau(p(v))$ 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 [36] gave a distributed connected components algorithm in $O(\mathcal{D} + \sqrt{V} \log^* V)$ time, using an MST subroutine in which the weight of edge e is 1 for $e \notin F$ and 0 for $e \in F$. Hence we have the following corollary to our cut pair algorithm, Theorem 6.

Corollary 27. *There is a distributed algorithm to compute all 3-edge-connected components with probability at least $1 - 1/V$ in $O(\mathcal{D} + \sqrt{V} \log^* V)$ time and using $O(E(\mathcal{D} + \sqrt{V} \log^* V))$ messages.*

6 Las Vegas Distributed Implementation

In this section we describe how to turn our Monte Carlo distributed algorithms into Las Vegas algorithms, by giving a *verifier* for each one. Given the output of the Monte Carlo algorithm, the verifier determines whether the output is correct or not; we re-run the Monte Carlo algorithm until the output is verified correct. For each of our verifiers, the time complexity is no more than the time complexity of the corresponding Monte Carlo algorithm; this fact and the fact that our algorithms work with high probability together imply that the resulting Las Vegas algorithms have the same asymptotic complexity as the Monte Carlo ones. See [27, §1.2] for more details.

Here is a high-level description of the three verifiers. The cut edge verifier works by labeling vertices according to their 2-edge-connected component; the cut vertex verifier works by labeling edges according to their *blocks*; the cut pair verifier works by exploiting relations between cut classes and fundamental cycles. All three of the verifiers rely on the fact that our Monte Carlo algorithms have one-sided error.

6.1 Verifier for Cut Edges

Recall that Algorithm 3 always outputs all cut edges, but may erroneously output some non-cut edges. Observe that a non-tree edge cannot be a cut edge; so we may assume the Monte Carlo algorithm outputs a set E'_C such that $E(T) \supseteq E'_C \supseteq E_C$, by having the verifier reject any output containing a non-tree edge. Here is the key idea: we compute the connected components of $T\backslash E'_C$. We only need to show how to determine if $E'_C\backslash E_C$ is nonempty; this can be done using the following proposition and its converse, which follows.

Proposition 28. *If $E'_C\backslash E_C$ is nonempty, there is a non-tree edge joining vertices in different connected components of $T\backslash E'_C$.*

Proof. Let e be any element of $E'_C\backslash E_C$. Since e is not a cut edge, there is another edge $f \in E$ connecting the two connected components of $T\backslash e$. The endpoints of f lie in different connected components of $T\backslash E'_C$. □

Proposition 29. *If $E'_C\backslash E_C$ is empty, then the connected components of $T\backslash E'_C$ are the 2-edge-connected components, and every non-tree edge has its endpoints in the same connected component of $T\backslash E'_C$.*

Proof. Corollary 26 guarantees that the connected components of $T\backslash E'_C$ are the 2-edge-connected components of G. Since each non-tree edge lies in at least one cycle (e.g. its fundamental cycle with T) its endpoints lie in the same 2-edge-connected component. □

Theorem 11. *There is a Las Vegas distributed algorithm to compute all cut edges in $O(\mathcal{D})$ time and using $O(E)$ messages, in expectation.*

Proof. We run the $O(\mathcal{D})$-time, $O(E)$-message Monte Carlo cut edge algorithm from Theorem 6, and as remarked earlier, we know its output E'_C satisfies $E'_C \supseteq E_C$. Then we run the following verifier, terminating if it accepts, and restarting from scratch (i.e., re-running the Monte Carlo algorithm) as long as it rejects.

If E'_C contains a non-tree edge, we reject. Otherwise (if $E'_C \subset E(T)$) we compute the connected components of $E(T)\backslash E'_C$ using an implementation like that in the proof of Theorem 10, which takes $O(V)$ messages and $O(\mathcal{D})$ time. If any non-tree edge has both endpoints in different components we reject, otherwise the verifier accepts; this can be checked in unit time and $O(E)$ messages. It follows from Propositions 28 and 29 that the verifier accepts if and only if $E'_C = E_C$. Since the probability of acceptance is $\Omega(1)$, the expected time complexity is $O(\mathcal{D} + \mathcal{D} + 1)$ and the expected message complexity is $O(E + V + E)$. □

6.2 Verifier for Cut Pairs

As in Section 4.4 we assume without loss of generality in this section that G is 2-edge-connected.

Consider the output of our Monte Carlo cut pair algorithm, Algorithm 4. The sense in which its output is one-sided is that every cut class is a subset of one of its output classes; the verifier must ensure that no cut class is "too big." To explain our approach, we define a notion of "wanting." Recall Φ_e, the multiset $\{\phi(f) \mid f \in C_e\}$ defined in Section 4.4; if the value x appears more than once in Φ_e, say that e *wants* the set $\{f \in C_e \mid \phi(f) = x\}$. With high probability, the wanted sets are precisely the cut classes. First, our verifier checks that whenever an edge lies in two wanted sets, those sets are the same; second, we use the following proposition to verify that no wanted set is "too big."

Proposition 30. *Let T be any spanning tree and e, f be edges that are not cut edges. If $\{e, f\}$ is not a cut pair, then some fundamental cycle of T contains exactly one of e and f.*

Proof. We prove the contrapositive; hence we assume that the characteristic vector of $\{e, f\}$ has even dot product with every fundamental cycle. By Proposition 3(c) the fundamental cycles form a basis of the cycle space; so $\{e, f\}$ is orthogonal to the cycle space, and by Proposition 3(a), lies in the cut space. Thus $\{e, f\}$ is an induced edge cut, and so (by Proposition 9) a cut pair. □

In order to apply Proposition 30, we count the size of all wanted sets, since then each non-tree edge can determine if its fundamental cycle is "missing" some members. Our strategy uses a modified Converge-Cast (Proposition 21) where we interpret max as lexicographic comparison of data. We need to give each edge a distinct $O(\log V)$-bit name, e.g. by concatenating the IDs of its endpoints. When e wants S, it sends the ordered pair $(e, |S|)$ towards all of S. (Concretely, for each tree edge $\{v, p(v)\}$ in S, this data is sent to v.) If two pairs (e, k) and (e', k') such that $k \neq k'$ are sent to the same location, the verifier rejects. Otherwise, each tree edge takes the label (e, k) where e is the lexicographically-maximal edge that wants it. We run another fc-cast with the new labels; then each non-tree edge f checks, for each distinct label (e, k) occurring in C_f, that there are exactly k edges in C_f with label (e, k). The complexity of the verifier is dominated by the fc-cast, and we thereby obtain the following theorem.

Theorem 12. *There is a Las Vegas distributed algorithm to compute all cut classes in $O(\mathcal{D})$ time and using $O(\min\{E \cdot \mathcal{D}, V^2\})$ messages, in expectation.*

6.3 Verifier for Cut Vertices and Blocks

For edges e, f in $E(G)$, define $e \sim f$ if either $e = f$, or $e \neq f$ and there is a cycle that contains both e and f. It is well-known that $\sim$ is an equivalence relation on E; its equivalence classes are called the *blocks* of G. The overall strategy is to try to label the edges according to the blocks, and then check via a generating relation that our labeling is correct.

The strategy for this verifier is more involved than for the other two, and a high-level description is as follows. Given two equivalence relations R and R' on the same set, we say that R *refines* R' if every equivalence class of R is a subset of some equivalence class of R'. Note that R refines R' and R' refines R if and only if $R = R'$. We use the notion of *local blocks*:

Definition 31. *The* local blocks at v, *denoted $\sim_v$, is an equivalence relation on $\delta(v)$ obtained by restricting $\sim$ to $\delta(v)$: namely we write $e \sim_v f$ iff $e, f \in \delta(v)$ and $e \sim f$.*

An analogue of Claim 16 will show that with high probability, the linear dependencies amongst columns of $M^{[v]}$ correspond to the local blocks at v. We hence compute equivalence relations $\sim'_v$ on $\delta(v)$, for each v, with the following properties:

- $\sim'_v$ always refines $\sim_v$
- we can collect the local relations $\sim'_v$ into a global equivalence relation $\sim'$ on E
- $\sim'$ always refines $\sim$
- with high probability, $\sim'_v = \sim_v$ for all v
- if $\sim'_v = \sim_v$ for all v, then $\sim' = \sim$

Finally, we need to check whether $\sim' = \sim$. To perform this check, we adapt an approach from work of Tarjan & Vishkin [35] and Thurimella [36], exemplified in the following proposition.

Proposition 32. *In $O(\mathcal{D})$ time and $O(E)$ messages we can compute a relation $\sim_0$ on E so that (1) whenever $e \sim_0 f$, e and f meet at a vertex, and (2) the symmetric reflexive transitive closure of $\sim_0$ is $\sim$.*

Some logical manipulation shows that

$$\forall v : (\forall u, w \text{ adjacent to } v : \{u, v\} \sim_0 \{v, w\} \Rightarrow \{u, v\} \sim' \{v, w\}) \quad \Longleftrightarrow \quad \sim \text{ refines } \sim'$$

and as a result, local checks complete the verification. We now give the details.

6.3.1 Computing $\sim'_v$

What do the local blocks look like? It is not hard to see that the local blocks at v correspond to the connected components of $G\backslash v$, in the sense that $\{u, v\} \sim_v \{w, v\}$ if and only if u and w are connected in $G\backslash v$. It is also straightforward to see that $F \subset \delta(v)$ is an induced edge cut if and only if F is a disjoint union of equivalence classes of $\sim_v$. We take $b = \lceil \Delta + 2\log_2 V \rceil$ and just as in Claim 16, with probability $1 - O(1/V^2)$, the following "good" case holds: the minimal sets of linearly dependent columns of $M^{[v]}$ correspond to the parts of $\sim_v$. (Notice that C is a minimal set of linearly dependent columns iff C's sum is the zero vector and no subset of C adds to the zero vector.) This leads to a simple idea, but we need to use some finesse in order that the $\sim'_v$ we compute from $M^{[v]}$ always refines $\sim_v$.

Our starting point is to compute an arbitrary partition π of the columns of $M^{[v]}$ into minimal zero-sum sets (such a partition exists because the sum of all columns is zero). It is possible that such a partition does not refine $\sim'_v$; so we need to check an additional property of π, namely that each pair of parts of π has mutually orthogonal span. (If this property does not hold, the verifier rejects and we re-start the Monte Carlo algorithm.) This property ensures that the only zero-sum sets of columns are unions of parts of π, which in turn shows that $\sim_v$ refines π. (Moreover, this property holds in the "good" case.) So we obtain $\sim'_v$ from π by replacing each column by its index in $\delta(v)$.

6.3.2 Computing $\sim'$ from $\sim'_v$

For the rest of the section we consider the spanning tree T upon which our algorithms operate as fixed; hence when we say "fundamental cycle of e" we mean with respect to T. We assume T is rooted at the leader vertex r and we let $p(v)$ denote the parent of v in T. In collecting the local relations into a global relation, it is instructive to consider the interaction between T and the blocks of the graph; Figure 4 gives an illustration. It is not hard to argue that the intersection of T with any given block B is a subtree of T; we define the *root* $r(B)$ of the block to be the root of this subtree. For example, in Figure 4, r and u are each the root of two blocks, and w is the root of one block. In general, the blocks for which v is the root correspond to the equivalence classes of $\sim_v$ not containing $\{v, p(v)\}$ (if $v = r$, all equivalence classes of $\sim_v$).

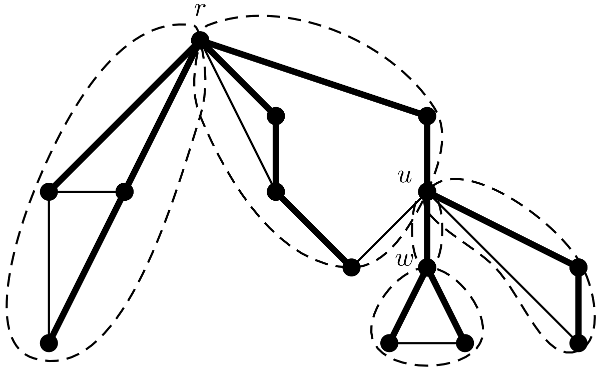

Figure 4: The interaction between a spanning tree and the blocks of a graph. Thick lines are tree edges, thin lines are non-tree edges, and the dashed regions indicate the five blocks of the graph.

For computational purposes, assign each equivalence class X of $\sim_v$ a number $i_v(X)$, using the numbers $1, 2, \ldots$ for each v. Then assign each block B the label $(r(B), i_{r(B)}(X))$ where the equivalence class X is the intersection of $\delta(r(B))$ with B. At a high level, to compute $\sim$ from $\sim_v$, within in each block, we broadcast its label starting from the block's root. Now given $\sim'_v$ instead of $\sim_v$, we can mimic this strategy so as to compute a global relation $\sim'$. We give pseudocode in Algorithm 6; the phrase "v sets directed label (v, u) to ℓ" means that v stores ℓ as the label of $\{v, u\}$ and notifies u of this fact with a message.

Any pair of $\sim'$-wise related edges are connected by a path of edges related pairwise by local $\sim'_v$ relations; since $\sim'_v$ refines $\sim_v$ which is a restriction of $\sim$, we see that $\sim'$ refines $\sim$. When $\sim'_v = \sim_v$ for all v, the preceding

Algorithm 6 Given local relations $\sim'_v$, compute a global relation $\sim'$.

```
at each vertex $v$, number the equivalence classes of $\sim'_v$ by $1, 2, \ldots$
at each vertex $v$, for each equivalence class $X$ of $\sim'_v$ not containing $\{v, p(v)\}$, for each $\{v, u\} \in X$, set
    directed label $(v, u)$ to $(v, i_v(X))$
when vertex $w$ sets directed label $(w, v)$ to $\ell$, if the label of $(v, w)$ exists and is not equal to $\ell$ then FAIL,
    else if directed label $(v, w)$ is unassigned, for each $\{v, u\} \sim'_v \{v, w\}$, set directed label $(v, u)$ to $\ell$
take the edge labels to identify the equivalence classes of $\sim'$
```

discussion implies that $\sim' = \sim$. The message complexity of Algorithm 6 is $O(E)$. When $\sim'_v = \sim_v$ for all v, the time complexity is $\mathcal{D}$ rounds; if more rounds than this elapse we restart the Las Vegas algorithm.

6.3.3 The Generating Relation $\sim_0$

In order to define $\sim_0$ we need a few preliminaries. Let $pre(v)$ denote a preordering of T starting from the root, and for each vertex v, let $desc(v)$ denote the number of descendants of v. Thus the set of descendants of v is the set of vertices with preorder labels in $\{pre(v), \ldots, pre(v) + desc(v) - 1\}$. The *subtree-neighbourhood* of v is defined to be v's descendants, in addition to every other vertex that is adjacent to a descendant of v via a non-tree edge. For each vertex v let the values $low(v)$ and $high(v)$ denote the minimum and maximum preorder label in the subtree-neighbourhood of v. Tarjan [34] introduced these *low* and *high* functions; they have been used in several biconnectivity algorithms [35, 36].

Definition 33. *The relation $\{w, v\} \sim_1 \{v, p(v)\}$ holds if and only if $\{w, v\} \notin T$ and either $pre(w) < pre(v)$ or $pre(w) \geq pre(v) + desc(v)$ (i.e., if w is not a descendant of v). The relation $\{v, p(v)\} \sim_2 \{p(v), p(p(v))\}$ holds if and only if either $low(v) < pre(p(v))$ or $high(v) \geq pre(p(v)) + desc(p(v))$ (i.e., if the subtree-neighbourhood of v is not contained in the descendants of $p(v)$). Define $\sim_0$ to be the union of $\sim_1$ and $\sim_2$.*

We illustrate these relations in Figure 5. Earlier work [35, 36] uses a different generating relation for $\sim$; ours is simpler and also has the crucial property that every two edges related by $\sim_0$ have a common vertex.

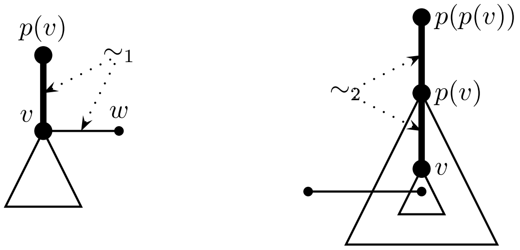

Figure 5: Schematic illustrations of the relations $\sim_1$ (left) and $\sim_2$ (right). Thick edges are tree edges, thin edges are non-tree edges, and triangles depict sets of descendants. Dotted arrows indicate pairs of edges related by $\sim_i$.

From now on, given a relation R, let R^* denote the equivalence relation obtained by taking the reflexive symmetric transitive closure of R. We now prove the most important property of $\sim_0$.

Proof of $\sim_0^* = \sim$ (Proposition 32). First, we argue that $\sim_0$ refines $\sim$; for this it suffices to show that when $e \sim_i f$ for $i \in \{1, 2\}$, e and f lie in the same block. If $\{w, v\} \sim_1 \{v, p(v)\}$, the fundamental cycle of $\{v, w\}$ contains $\{v, p(v)\}$, so $\{v, w\} \sim \{v, p(v)\}$ as needed. If $\{v, p(v)\} \sim_2 \{p(v), p(p(v))\}$ then there is edge from a descendant of v to a non-descendant of $p(v)$; the fundamental cycle of this edge contains both $\{v, p(v)\}$ and $\{p(v), p(p(v))\}$, as needed.

Second, we must show that $\sim$ refines $\sim_0^*$. Define $e \sim_{FC} f$ if e and f lie on a common fundamental cycle. In [35, Theorem 1], Tarjan & Vishkin show that $\sim_{FC}^* = \sim$. So it suffices to show that when $e \sim_{FC} f$, $e \sim_0^* f$

holds. In other words, we need to show that each fundamental cycle lies in a single equivalence class of $\sim_0^*$. We provide a pictorial argument of this fact in Figure 6. □

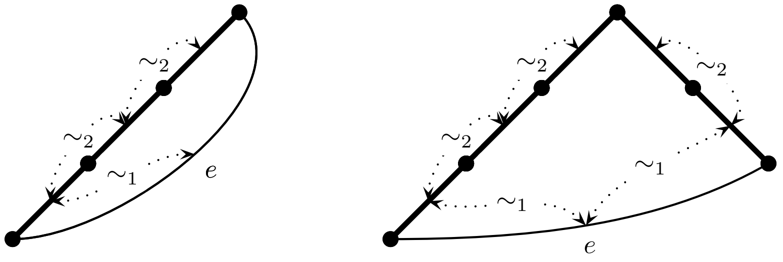

Figure 6: The fundamental cycle C_e in the proof of Proposition 32. Edges of T are thick lines and e is labeled. The left diagram shows the case that one of e's endpoints is a T-descendant of the other, while the right diagram shows the case that e's endpoints are unrelated. Dotted arrows indicate pairs of edges related by $\sim_i$.

We now recap the distributed implementation of our cut vertex verifier.

Theorem 13. *There is a Las Vegas distributed algorithm to compute all cut vertices in $O(\mathcal{D} + \Delta/\log V)$ time and using $O(E(1 + \Delta/\log V))$ messages, in expectation.*

Proof. We compute a random b-bit circulation for $b = \lceil \Delta + 2\log_2 V \rceil$ and use the resulting values to compute local relations $\sim_v'$. (As mentioned in Section 6.3.2 the verifier may reject at this stage.) We then combine this information into a global labeling $\sim'$ of edges (and again, the verifier may reject at this stage).

There is a straightforward distributed protocol to compute $pre(v), desc(v), low(v)$ and $high(v)$ at each v in $O(h(T)) = O(\mathcal{D})$ time and using $O(E)$ messages; see e.g. [30, 36]. After this, each vertex sends these four values to all of its neighbours, with communication taking place along all edges in parallel; this takes $O(1)$ time and $O(E)$ messages.

At this point, for each pair e, f of edges that are related by $\sim_0$, their common endpoint v checks that $e \sim' f$ holds. If there is a violation at any vertex, the verifier rejects, and if not, the verifier accepts. The labels $\sim'$ give the blocks; vertex v is a cut vertex iff at least two blocks meet at v.

Computing ϕ dominates the time and message complexity; each other step takes $O(\mathcal{D})$ time and $O(E)$ messages. Noting that the verifier accepts each time with probability at least $1-1/V$, Theorem 13 follows. □

7 Lower Bounds on Distributed Time

In this section we give precise assumptions under which our distributed cut edge and cut pair algorithms achieve universal optimality. 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 [29, §3.4 & Ch. 24].

Definition 34. *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.*

The *state* of a vertex means the contents of its memory. We omit the straightforward inductive proof of the following standard proposition.

Proposition 35. *Let two graphs both contain a vertex v and have the same graph topology and node IDs in the distance-d neighbourhood of v. If the same deterministic distributed algorithm is run on both graphs, the state of v is the same in both instances for the first $d - 1$ rounds. For a randomized algorithm, the distribution over states of v is the same.*

For a graph G, a vertex $v \in V(G)$ and an integer $\ell \geq 3$, we now define graphs G_c and G_p that implicitly depend on ℓ and v. Specifically, let G_c denote the graph obtained from G by attaching a ℓ-edge cycle to G at v, and let G_p denote the graph obtained from G by attaching a $(\ell - 1)$-edge path to G at v, as shown in Figure 7. Give corresponding vertices v_i in the two graphs the same ID.

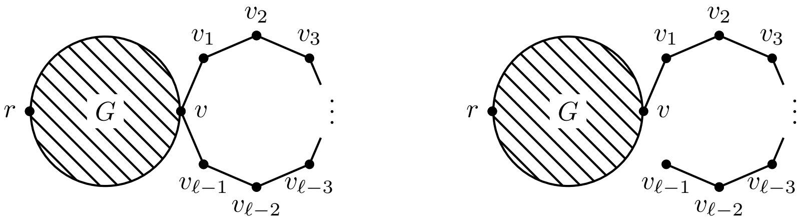

Figure 7: Left: the graph G_c. Right: the graph G_p.

Theorem 14. *Any deterministic distributed algorithm for finding all cut edges that has termination detection takes at least $\mathcal{D}/2$ rounds on every graph.*

Proof. Consider for the sake of contradiction a graph G upon which the algorithm terminates in $t < \mathcal{D}/2$ rounds. Let v be any vertex of distance at least $\mathcal{D}/2$ away from r, and let $\ell = 2t + 2$. By Proposition 35, the algorithm also sets `done` := TRUE at r on G_p and G_c in t rounds, so the algorithms terminate then.

Now consider $v_{\ell/2}$; using Proposition 35 again, we see that its state is the same at termination in both instances. Since the edges incident to $v_{\ell/2}$ are cut edges in G_p but not in G_c, they must have been incorrectly classified at $v_{\ell/2}$ in at least one instance. □

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.

Lemma 36. *In an algorithm with a single initiator, every vertex at distance t from r is quiescent for the first t rounds.*

Theorem 15. *Any deterministic distributed algorithm for finding all cut edges that has a single initiator takes at least $\mathcal{D}/2$ rounds on every graph.*

Proof. Suppose the algorithm terminates in $t < \mathcal{D}/2$ rounds on a graph G. Let v be any vertex of distance at least $\mathcal{D}/2$ away from r. Then by Proposition 35 the algorithm also terminates in t rounds on $G_c^{3,v}$ and $G_p^{3,v}$. By Lemma 36 vertex v_1 is quiescent during the entire execution of the algorithm on these new graphs; hence the incident edges cannot be correctly classified in both instances. □

For randomized algorithms we have the following lower bound.

Theorem 16. *Any randomized distributed algorithm with error probability less than $1/4$ for finding all cut edges takes at least $\mathcal{D}/4$ rounds in expectation, if it has a single initiator or termination confirmation.*

Proof. We use the same setup as in the proofs of Theorems 14 and 15. Markov's inequality shows that when running the algorithm on G, the time of termination t satisfies $\Pr[t \leq \mathcal{D}/2] \geq 1/2$. The distribution on the state of the crucial vertex — $v_{\ell/2}$ for termination confirmation, v_1 for single initiator — is the same on both G_c and G_p at time $\mathcal{D}/2$. So of the $\geq 1/2$ probability mass of termination before $\mathcal{D}/2$, either $1/4$ incorrectly classifies edges of G_c as cut edges or edges of G_p as not cut edges. □

The same lower bounds hold for finding 2-edge-connected components and cut pairs, since the new edges of G_c are in cut pairs, while the new edges of G_p are not. It is straightforward to verify that our distributed algorithms can be implemented so as to have a single initiator and termination detection; then their universal optimality follows.

If we do not require a single initiator or termination detection, and if we change our input model to allow additional parameters of G to be initially known at each node, *neighbourhood cover* techniques of Elkin [9] 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.

8 Parallel Cut Pairs on the EREW PRAM

In this section we give a parallel cut pair algorithm of time complexity $O(\log V)$ for the EREW PRAM. Computing the OR of n bits has a lower bound of $\Omega(\log n)$ time in this model; from this an easy combinatorial reduction yields an $\Omega(\log V)$ time lower bound for finding all cut pairs of a graph, so our algorithm is time-optimal. As in Section 4.4 we assume without loss of generality in this section that G is 2-edge-connected.

We will require several common subroutines. In $O(V + E)$ work and space and $O(\log V)$ time we can compute a spanning forest (Halperin and Zwick [14]). An *ear decomposition* can be computed in the same complexity using the approaches in [25, 26] and plugging in the result of [14] for the spanning forest subroutine. *Expression evaluation* of an n-node tree can be accomplished in $O(n)$ work and space and $O(\log n)$ time (e.g. see the book of JáJá [19, Ch. 3]). We let $T(n), S(n), W(n)$ denote the time, space, work complexity to sort n numbers of length $O(\log n)$ bits; we give references to the best known algorithms for this problem in Section 1.2. First, we give our Monte Carlo cut pair algorithm.

Theorem 17. *There is a parallel algorithm to compute all cut pairs with probability at least* $1 - 1/V$ *in* $O(\log V + T(E))$ *time,* $O(E + S(E))$ *space, and* $O(E + W(E))$ *work.*

Proof. We implement Algorithm 4 distributively. First, we claim we can implement the subroutine RAND-b-BIT-CIRC distributively to generate a random $O(\log V)$-bit circulation in logarithmic time and linear work; the completion steps (Algorithm 1) are accomplished via a call to expression evaluation in which we compute the expression $\phi(e) := \bigoplus_{f \in \delta(v) \setminus e} \phi(f)$ for each tree edge $e = \{v, p(v)\}$. We implement Line 2 of Algorithm 4 via a sort. □

8.1 Las Vegas Cut Pair Algorithm

The verifier for our parallel cut pair algorithm works by attempting to construct the *2-cactus* of G which acts as a certificate for all of the cut pairs. Our terminology is derived from a more general sort of cactus originally due to Dinits, Karzanov & Lomonosov [8] that acts as a certificate for all minimum edge cuts. Say that $u \equiv v$ in G if the edge-connectivity between u and v is at least 3; it is easy to show (e.g. using the max-flow min-cut theorem) that $\equiv$ is an equivalence relation. For any equivalence relation R on V, let G/R denote the multigraph with loops obtained by contracting all equivalence classes of R.

Definition 37. *The* 2-cactus $\mathsf{Ca}(G)$ *of* G *is* $G/\equiv$.

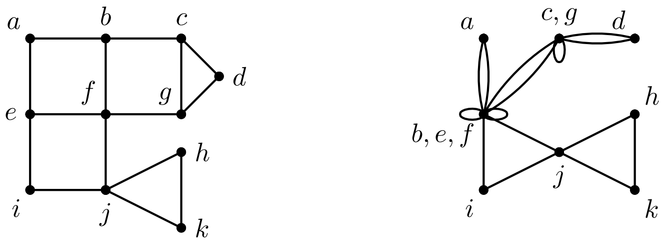

Figure 8: A graph (left) and its 2-cactus (right).

An example of a 2-cactus is given in Figure 8.

Proposition 38. *(a) For any equivalence relation* R *on* V*, every cut pair of* G/R *is a cut pair of* G*. (b) Every cut pair of* G *is a cut pair of* $\mathsf{Ca}(G)$*.*

Proof. In part (a), let $\delta(X)$ be a cut pair of G/R. Let $\cup X$ be the union of the equivalence classes in X. Then the edge sets $\delta(\cup X)$ in G and $\delta(X)$ in G_c are the same.

In part (b), let $\delta(S)$ be a cut pair of G. Note that $s \not\equiv t$ holds for each $s \in S, t \notin S$. Let $[S]$ be the set of all $\equiv$-equivalence classes in S; the cut pair $\delta(S)$ becomes cut pair $\delta([S])$ in $G/\equiv$ which is $\mathsf{Ca}(G)$. □

Now we recall the earlier convention (from Section 2.1) of using ϕ to denote a (random) b-bit binary circulation, wherein for each i the sets $\{e \mid \phi_i(e) = 1\}$ are independent binary circulations selected uniformly at random. Recall also Corollary 10 which (together with the assumption that G is 2-edge-connected) says that $\phi(e) = \phi(f)$ always holds when $\{e, f\}$ is a cut pair, and holds with probability $1/2^b$ otherwise. Define an *illusory cut pair* to be a pair of edges $\{e, f\}$ that has $\phi(e) = \phi(f)$ but is not a cut pair. Our strategy will be to define another relation $\equiv'$ so that $\equiv$ and $\equiv'$ will agree when there are no illusory cut pairs; we will then use Proposition 38 to verify that there are no illusory cut pairs.

8.1.1 Pinching Ears

The relation $\equiv'$ must provide an alternate way of constructing $\mathsf{Ca}(G)$, when there are no illusory cut pairs. To this end we examine the properties of $\mathsf{Ca}(G)$ in more detail. We take the convention that a parallel pair of edges or a self-loop is a simple cycle.

An *ear decomposition* of G is a sequence of graphs $G_0 \subset G_1 \subset G_2 \subset \cdots \subset G_k = G$ such that G_1 is just a vertex and each G_i is obtained from G_{i-1} by attaching a simple cycle (a *closed ear*) or path with both endpoints in G_{i-1} (an *open ear*). It is well-known that a graph is 2-edge-connected if and only if it admits an ear decomposition. The edges $E_i := E(G_i)\backslash E(G_{i-1})$ are the ith *ear* of G.

We omit the straightforward proof of Proposition 39.

Proposition 39. *Every pair of nodes in* $\mathsf{Ca}(G)$ *has edge-connectivity equal to 2.*

Call a graph *cactuslike* if every pair of nodes has edge-connectivity equal to 2.

Proposition 40. *The following are equivalent for any graph: (1) it is cactuslike, (2) in some ear decomposition, all its ears are closed, (3) in every ear decomposition, all its ears are closed.*

Proof. Trivially, (3) implies (2). It is easy to see that (2) implies (1) by induction on the ears. To see that (1) implies (3), suppose that there is an open ear E_i with endpoints u, v. But then there are three edge-disjoint paths between u and v, two in G_{i-1} as well as E_i. □

We obtain the following corollary using induction on the ears.

Corollary 41. *In a cactuslike graph, for any ear decomposition, the cut classes are the same as the nonsingleton ears.*

Next we start looking at ear decompositions of G, which will be useful algorithmically.

Lemma 42. *Every cut class of G lies within a single ear of any ear decomposition of G.*

Proof. Suppose otherwise, that there is a cut pair $\{e, f\}$ with $e \in E_i$ and $f \in G_{i-1}$. Since G_{i-1} is 2-edge-connected, $G_{i-1}\backslash f$ is connected. But $G_i\backslash\{e, f\}$ is obtained by attaching 2 paths to G_{i-1}, and so is connected. By induction on the remaining ears we see that each $G_j\backslash\{e, f\}$ for $j \geq i$ is connected; in particular for $j = k$ this means $G\backslash\{e, f\}$ is connected, a contradiction. □

Consider now the image E'_i of any ear E_i under contraction by $\equiv$. Lemma 42 implies E'_i is a union of cut classes. The cut classes of $\mathsf{Ca}(G)$ and G agree (Proposition 38), and by Corollary 41 every cut class of $\mathsf{Ca}(G)$ is a simple cycle. Furthermore, there is a natural cyclic order on each cut class in E_i (e.g. see Figure 3 on page 8) and it is not hard to see that the corresponding cycle of formed by E_i in $\mathsf{Ca}(G)$ has the same cyclic order. Hence we can obtain E'_i from E_i by "pinching" all of the cut classes in E_i to become cycles. We now make this precise.

Let a given ear have vertices and edges $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$ in that order, where $v_1 = v_k$ iff the ear is closed. To *pinch* a subset $U = \{e_{i(1)}, e_{i(2)}, \ldots, e_{i(t)}\}$ of edges with $i(1) < i(2) < \cdots < i(t)$ means that we contract the pairs $(v_{i(1)}, v_{i(2)-1}), (v_{i(2)}, v_{i(3)-1}), \ldots, (v_{i(t-1)}, v_{i(t)-1})$ and $(v_{i(t)}, v_{i(1)-1})$. After pinching, the set U becomes a simple cycle in the same order as before. Then E'_i is the image of E_i under pinching all cut classes in the ear, as we illustrate in Figure 9. This leads to our verification approach.

Definition 43. *For the ith ear, define the equivalence relation $\equiv_i$ to be the transitive closure of all pairs that are contracted together when pinching the cut classes in the ear.*

(We think of the pinching of the various cut classes as happening simultaneously, or in other words if P_K denotes the pairs contracted when pinching a set K, then $\equiv_i$ is the transitive closure of $\{\cup_K P_K \mid K \text{ a cut class in } E_i\}$.) So making our earlier statement more precise, $(E_i/\equiv_i) = E'_i$. The next claim follows by induction on the ears of the ear decomposition.

Claim 44. *$G/(\cup_i \equiv_i)^* = \mathsf{Ca}(G)$.*

As a result, $(\cup_i \equiv_i)^*$ is identical to $\equiv$.

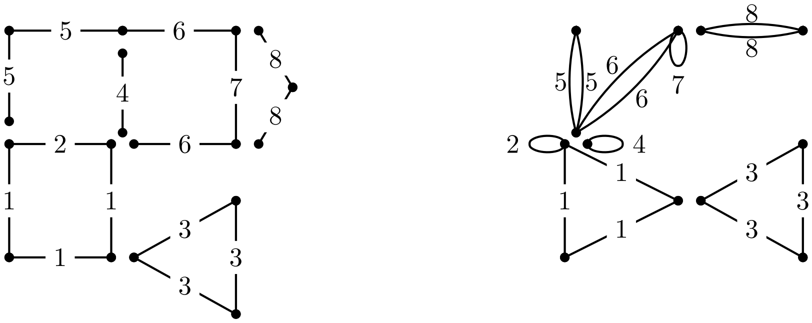

Figure 9: Left: an ear decomposition of the graph from Figure 8; edges are labelled by their cut class. Right: the pinched ears after contracting by $\equiv_i$.

8.1.2 Detecting Errors

In the algorithm we are designing, we don't know the cut pairs; rather, we have computed ϕ and know that with high probability, ϕ labels edges by their cut class. We compute the following instead.

Definition 45. *For the ith ear, define the equivalence relation $\equiv'_i$ to be the transitive closure of all pairs that are contracted together when pinching each of the sets $\{e \in E_i \mid \phi(e) = k\}$ for all $k \in \{\phi(e) \mid e \in E_i\}$ (i.e., each group of edges with the same ϕ value).*

Note first that if there are no illusory cut pairs, then $\equiv'_i$ is the same as $\equiv_i$. Define the equivalence relation $\equiv'$ to be equal to $(\cup_i \equiv'_i)^*$. Our algorithm computes $H := G/\equiv'$ and tries to verify all cut pairs. First, if H is not cactuslike, by Claim 44, we can reject since there is an illusory cut pair. Second, if H is cactuslike, we check that every pair $\{e, f\}$ with $\phi(e) = \phi(f)$ is a cut pair of H. By Proposition 38 this detects any illusory cut pairs. This completes the verifier, and we obtain the following theorem.

Theorem 18. *There is a Las Vegas parallel algorithm to compute all cut pairs in $O(\log V + T(E))$ time, $O(E + S(E))$ space, and $O(E + W(E))$ work, in expectation.*

Proof. To compute $\equiv'_i$ on each ear, we radix sort the edges on that ear lexicographically according to the pair $(\phi(e), pos(e))$ where $pos(e)$ is the position along the ear.

To compute $\equiv'$ from the relations $\equiv'_i$ we build an auxilliary graph on vertex set V and draw an edge for each pair of vertices that is related by some $\equiv'_i$; then the equivalence classes of $\equiv'$ are the connected components of this auxilliary graph. In other words, this can be done using the connected components routine of Halperin & Zwick [13]. From this, computing the multigraph H takes constant time and linear work.

To check if H is cactuslike, we compute an ear decomposition and see if all ears are closed. Finally, using Corollary 41, we can check that every pair $\{e, f\}$ with $\phi(e) = \phi(f)$ is a cut pair of H in the required complexity bound. □

9 Future Work

At the most basic level, it would be interesting to push further and find efficient algorithms for higher types of connectivity, such as finding all 3-edge-cuts in $O(E)$ sequential time or $O(\mathcal{D})$ distributed time. The state of the art for this problem in the sequential model is $O(V^2)$ time [11, 21]. It would also be interesting to reduce the complexity of our parallel cut pairs algorithm to linear work and logarithmic time; it seems plausible that another approach would avoid radix sort.

It is possible to deterministically compute the cut edges in the distributed model using $O(\mathcal{D})$ time and $O(E)$ messages, as was shown in the thesis of Pritchard [30]. (The approach is based on the observation that $\{v, p(v)\}$ is a cut edge if and only if $low(v) \geq v$ and $high(v) < v + desc(v)$.) However, we do not know of any deterministic analogues of our distributed cut pair or cut vertex algorithms.

It would be interesting to know if our distributed cut vertex algorithm could be synthesized with the cut vertex algorithm of Thurimella [36] to yield further improvement. Alternatively, a lower bound showing that no $O(\mathcal{D})$-time algorithm is possible for finding cut vertices would be very interesting.

Acknowledgement

We would like to thank Santosh Vempala for helpful ideas on using randomness to compute small cuts [32].

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] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *Proc. 14th SPAA*, pages 258–264, 2002.

[3] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st FOCS*, pages 514–522, 1990.

[4] A. Bondy and U. Murty. *Graph Theory with Applications.* North-Holland, 1976.

[5] E. J.-H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Trans. Softw. Eng.*, SE-8:391–401, 1982.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[7] R. Diestel. *Graph theory.* Springer-Verlag, New York, 3rd edition, 2006.

[8] Y. Dinitz, A. V. Karzanov, and M. V. Lomonosov. On the structure of the system of minimum edge cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka, 1976.

[9] 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.

[10] D. S. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacement. *SIAM J. Comput.*, 22:587–616, 1993.

[11] Z. Galil and G. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22:57–61, 1991.

[12] 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.

[13] S. Halperin and U. Zwick. An optimal randomised logarithmic time connectivity algorithm for the EREW PRAM. *J. Comput. Syst. Sci.*, 53(3):395–416, 1996. Preliminary version appeared in *Proc. 6th SPAA*, pages 1–10, 1994.

[14] 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.

[15] Y. Han and X. Shen. Parallel integer sorting is more efficient than parallel comparison sorting on exclusive write prams. *SIAM J. Comput.*, 31(6):1852–1878, 2002. Preliminary version appeared in *Proc. 10th SODA*, pages 419–428, 1999.

[16] W. Hohberg. How to find biconnected components in distributed networks. *J. Parallel Distrib. Comput.*, 9(4):374–386, 1990.

[17] J. Hopcroft and R. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comp.*, 2(3):135–158, 1973.

[18] S. T. Huang. A new distributed algorithm for the biconnectivity problem. In *Proc. 1989 International Conf. Parallel Processing*, pages 106–113, 1989.

[19] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[20] E. Jennings and L. Motyckova. Distributed computation and incremental maintainance of 3-edge-connected components. In *Proc. 3rd SIROCCO*, pages 224–240, 1996.

[21] A. Kanevsky and V. Ramachandran. Improved algorithms for graph four-connectivity. *J. Comput. Syst. Sci.*, 42(3):288–306, 1991. Preliminary version appeared in *Proc. 28th FOCS*, pages 252–259, 1987.

[22] C. P. Kruskal, L. Rudolph, and M. Snir. Efficient parallel algorithms for graph problems. *Algorithmica*, 5(1):43–64, 1990. Preliminary version appeared in *Proc. 15th ICPP*, pages 869–876, 1986.

[23] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28:40–66, 1998. Preliminary version appeared in *Proc. 14th PODC*, pages 238–249, 1995.

[24] 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.

[25] Y. Maon, B. Schieber, and U. Vishkin. Parallel ear decomposition search (EDS) and st-numbering in graphs. *Theoretical Comput. Sci.*, 47:277–298, 1986.

[26] G. L. Miller and V. Ramachandran. A new graph triconnectivity algorithm and its parallelization. *Combinatorica*, 12:53–76, 1992. Preliminary version appeared in *Proc. 19th STOC*, pp. 254–263, 1987.

[27] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 2000.

[28] D. Peleg. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.*, 8(1):96–99, 1990.

[29] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[30] D. Pritchard. Robust network computation. Master's thesis, MIT, 2005.

[31] D. Pritchard. Fast distributed computation of cuts via random circulations. In *Proc. 35th ICALP, Part I*, pages 145–160, 2008.

[32] D. Pritchard and S. Vempala. Symmetric network computation. In *Proc. 18th SPAA*, pages 261–270, 2006.

[33] R. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[34] R. E. Tarjan. A note on finding the bridges of a graph. *Inform. Process. Lett.*, 2:160–161, 1974.

[35] 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.

[36] 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.

[37] Y. H. Tsin. A simple 3-edge-connected component algorithm. *Theory Comput. Systems*, 40(2):125–142, 2005.

[38] Y. H. Tsin. An efficient distributed algorithm for 3-edge-connectivity. *Int. J. Found. Comput. Sci.*, 17(3):677–702, 2006.