A Distributed Minimum Cut Approximation Scheme

Hsin-Hao Su *
University of Michigan

Abstract

In this paper, we study the problem of approximating the minimum cut in a distributed message-passing model, the CONGEST model. The minimum cut problem has been well-studied in the context of centralized algorithms. However, there were no known non-trivial algorithms in the distributed model until the recent work of Ghaffari and Kuhn. They gave algorithms for finding cuts of size $O(\epsilon^{-1}\lambda)$ and $(2+\epsilon)\lambda$ in $O(D)+\tilde{O}(n^{1/2+\epsilon})$ rounds and $\tilde{O}(D+\sqrt{n})$ rounds respectively, where λ is the size of the minimum cut. This matches the lower bound they provided up to a polylogarithmic factor. Yet, no scheme that achieves $(1+\epsilon)$-approximation ratio is known. We give a distributed algorithm that finds a cut of size $(1+\epsilon)\lambda$ in $\tilde{O}(D+\sqrt{n})$ time, which is optimal up to polylogarithmic factors.

1 Introduction

The minimum cut problem is a fundamental problem in graph algorithms and network design. Given a weighted undirected graph $G=(V,E)$, a cut $C=(S,V\setminus S)$ where $\emptyset\subset S\subset V$, is a partition of vertices into two non-empty sets. The weight of a cut, $w(C)$, is defined to be the sum of the edge weights crossing C. The minimum cut problem is to find a cut with the minimum weight. The exact version of the problem as well as the approximate version have been studied for many years [6, 10, 8, 13, 12, 3, 15, 9] in the context of centralized models of computation, resulting in nearly linear time algorithms [9, 12, 8].

Elkin [2] and Das Sarma et al. [1] addressed the problem in the distributed message-passing model. The problem has trivial time complexity of $\Theta(D)$ (unweighted diameter) in the LOCAL model, where the message size is unlimited. Ghaffari and Kuhn [5] recently developed approximation algorithms for this problem in the CONGEST model where each message is bounded by $\Theta(\log n)$ bits. They assume that the edges of G have integer weights from $\{1,\ldots,n^{\Theta(1)}\}$ and treat G as an unweighted multigraph, where an edge e with weight $w(e)$ is converted to $w(e)$ parellel edges, while still only $\Theta(\log n)$ bits can be sent over these parallel edges together in each round. Let λ be the value of the minimum cut, they give an algorithm that finds a cut of size at most $O(\epsilon^{-1}\lambda)$ in $O(D)+O(n^{1/2+\epsilon}\log^3 n\log\log n\log^* n)$ time. Moreover, they gave an algorithm that finds a cut of size at most $(2+\epsilon)\lambda$ in $O((D+\sqrt{n}\log^* n)\log^2 n\log\log n\frac{1}{\epsilon^5})$ time. Das Sarma et al. [1] showed α-approximating the minimum cut requires $\tilde{\Omega}(D+\sqrt{n})$ rounds for weighted graphs for any $\alpha\geq 1$. Ghaffari and Kuhn extended their lower bound for unweighted multigraphs (which is equivalent to the setting where one is allowed to send messages of size $w\cdot\Theta(\log n)$ over an edge of weight w in weighted graphs). For unweighted simple graphs, they also gave a lower bound of $\tilde{\Omega}(D+\sqrt{n/\alpha})$. Therefore, the upper bound and lower bound provided by Ghaffari and Kuhn match up to a polylogarithmic factor.

*This work is supported by NSF grants CCF-0746673, CCF-1217338, and CNS-1318294 and a grant from the US-Israel Binational Science Foundation. This work was done while visiting MADALGO at Aarhus University.

However, still no approximation algorithms exist for any approximation factor less than 2. In this paper, we give a simple algorithm that finds a minimum cut of size at most $(1+\epsilon)\lambda$ in $\tilde{O}(D+\sqrt{n})$ time. In particular, our algorithm runs in $O((\log^{11} n/\epsilon^{17})(D+\sqrt{n}\log^* n))$ rounds.

Our approach uses the semi-duality between minimum cuts and tree packings as in [9, 16]. Karger [9] showed that if we greedily pack enough trees, then for any minimum cut, there is a tree crossing the cut at most twice. However, it is technically not easy to utilize this fact to find minimum cuts in the distributed model. Instead, we use a lemma by Thorup [16], which shows that if we pack more trees then there is at least one minimum cut that is crossed by a tree exactly once. We take some ingredients from Ghaffari and Kuhn's algorithm and Thurimella's algorithm [17] for identifying biconnected components to devise a procedure that is able to simultanously test the values of the $n-1$ cuts induced by deleting one of the $n-1$ edges in a tree. Note that the number of trees we have to pack is polynomial in the value of the minimum cut. Thus, we will first use the sampling lemma of Karger [7] to obtain a sampled graph that scales the value of the minimum cut down to $O(\log n/\epsilon^2)$. Then we only have to pack polylogarithmic number of trees. Finally, we combine the resampling procedure, the tree packing, and the procedure for testing tree-induced-cuts to find an approximate minimum cut.

2 Distributed Minimum Cut Approximation

Let G be a connected graph with integer weights from $\{1,\ldots,W\}$, where $W = n^{\Theta(1)}$. We will treat G as a multigraph with uniform edge weights. Let λ be the weight of the minimum cut of G. We show how to find such an approximate minimum cut whose weight is at most $(1+\epsilon)\lambda$.

An edge e is a *bridge* if it does not exist a cycle in G passing e (or equivalently, deleting e breaks G into two connected components). Given two graph A and B with the same vertex set, $A+B$ is the multigraph obtained by including edges in A and edges in B.

A *tree packing* $\mathcal{T}$ is a multiset of spanning trees. The *load* of an edge e with respect to $\mathcal{T}$ is the number of trees in $\mathcal{T}$ containing e. Given a tree T, we say a cut is *induced* by T if such a cut is obtained by deleting an edge $e \in T$. We will denote this cut by $C(T,e)$. A tree packing $\mathcal{T} = \{T_1,\ldots,T_k\}$ is *greedy* if each T_i is a minimum spanning tree with respect to the loads induced by $\{T_1,\ldots,T_{i-1}\}$. Let $\epsilon' = \Theta(\epsilon)$ such that $(1+\epsilon')^3/(1-\epsilon') = 1+\epsilon$.

Lemma 2.1 (Thorup [16])**.** *A greedy tree packing with $96(\lambda+1)^7\log^3 m$ trees contains a tree crossing some min-cut only once.*

Remark 2.2. *The number of trees in the original statement of the lemma is $\omega(\lambda^7\log^3 m)$, though the proof actually implies that $\Theta(\lambda^7\log^3 m)$ is enough. In particular, Thorup showed $24\lambda\ln m/\epsilon^2$ trees is sufficient, where ϵ satisfies $\frac{\epsilon(3+\log_{1+\alpha} m)}{\lambda} + \alpha < \frac{2}{\lambda(\lambda+1)}$ for some $\alpha < 1$. We can choose $\alpha = \frac{1}{\lambda(\lambda+1)}$ and $\epsilon = \frac{1}{2(\lambda+1)^3\ln m}$ to make the inequality hold. Therefore, $96(\lambda+1)^7\ln^3 m$ trees is sufficient.*

We describe our algorithm in Algorithm 1. The subroutine Test(T,κ) returns a cut whose weight is at most $(1+\epsilon')\kappa$ w.h.p. if there exists a cut in G induced by T with weight at most κ.

We show that w.h.p. the algorithm will output a cut C with $w(C) \le (1+\epsilon)\lambda$. In particular, consider the iteration i where $\lambda \in [X_i, X_{i+1}]$. Let λ' denote the value of the minimum cut in the sampled graph H_i. If $i=0$, then it is clear that $\lambda' = \lambda \le X_1 = 20\ln n/\epsilon'^2$. If $i>0$, since we sampled with probability $1/2^i = 20\ln n/(\epsilon'^2 X_{i+1}) = 10\ln n/(\epsilon'^2 X_i) \ge 10\ln n/(\epsilon'^2\lambda)$, we know that w.h.p. for any cut C [7, Corollary 2.4],

$$(1-\epsilon')\cdot w_G(C)/2^i \le w_{H_i}(C) \le (1+\epsilon')\cdot w_G(C)/2^i.$$

Therefore, $\lambda' \leq (1+\epsilon')\lambda/2^i \leq (1+\epsilon')20\ln n/\epsilon'^2$. If we pack $96(\lambda'+1)^7\log^3 m$ trees in $\mathcal{T}$, then by Lemma 2.1 there exists a tree crossing some minimum cut C^* of H_i only once. Notice that for any other cut C',

$$w_G(C^*) \leq 2^i \cdot \frac{w_{H_i}(C^*)}{1-\epsilon'} = 2^i \cdot \frac{\lambda'}{1-\epsilon'} \leq 2^i \cdot \frac{w_{H_i}(C')}{1-\epsilon'} \leq \frac{1+\epsilon'}{1-\epsilon'} \cdot w_G(C')$$

1: $X_0 \leftarrow 1$
2: $i \leftarrow 0$
3: **repeat**
4:    $X_{i+1} \leftarrow 2^i \cdot 20\ln n/\epsilon'^2$
5:    (We are assuming $\lambda \in [X_i, X_{i+1}]$ in this iteration)
6:    Let H_i be the subgraph sampled with probability $p = 1/2^i$ on each edge of G.
7:    Find a greedy tree packing $\mathcal{T}$ with $96((1+\epsilon')20\ln n/\epsilon'^2+1)^7\ln^3 m$ trees in H_i
8:    $\gamma \leftarrow X_i$
9:    **repeat**
10:       **for** each $T \in \mathcal{T}$ **do**
11:          Call Test$(T, (1+\epsilon')\gamma)$.
12:          If Test$(T, (1+\epsilon')\gamma)$ returns a cut C, output C and terminate.
13:       **end for**
14:       $\gamma \leftarrow (1+\epsilon')\gamma$
15:    **until** $\gamma > \frac{1+\epsilon'}{1-\epsilon'} \cdot X_{i+1}$
16:    $i \leftarrow i+1$
17: **until** $X_{i+1} > nW$

Algorithm 1: $(1+\epsilon)$-approximate minimum cut

Therefore, one of the cuts induced by some $T \in \mathcal{T}$ is an $(1+\epsilon')/(1-\epsilon')$ approximate minimum cut. Denote this cut by C', so $w(C') \in [X_i, ((1+\epsilon')/(1-\epsilon')) \cdot X_{i+1}]$. Therefore in the i'th iteration, there exists γ in the loop (Line 9–Line 15) such that $w(C') \in [\gamma, (1+\epsilon')\gamma]$. So w.h.p. we will output a cut with weight at most $(1+\epsilon')^2\gamma \leq (1+\epsilon')^2 w(C') \leq (1+\epsilon')^3/(1-\epsilon')w(C^*) = (1+\epsilon)w(C^*)$.

2.1 Distributed Implmentation

We have shown the correctness of this algorithm. It remains to show how to implement it in $\tilde{O}(D+\sqrt{n})$ distributed rounds, and in particular, to implement the tree packing (Line 7) and Test(T,κ) in Algorithm 1. To pack k trees, it is striaghtfoward to apply k MST computations on the graph where the edge weights are equal to the number of trees including it. This can be done in $O(k(D+\sqrt{n}\log^* n))$ rounds [11].

Given a partition $\mathcal{P}$ of G into components, Ghaffari and Kuhn [5] devised a testing procedure to test if there is a cut induced by a component in $\mathcal{P}$ that has weight less than κ in $\widetilde{O}(D+\sqrt{n})$ rounds. Given a spanning tree T, we will show how to test the $n-1$ cuts induced by T also in $\widetilde{O}(D+\sqrt{n})$ rounds.

1: **for** $i \leftarrow 1 \ldots k = \Theta(\frac{\log n}{\epsilon^2})$ **do**
2: Let G_i be the subgraph obtained by sampling each edge of G independently with probability $1 - 2^{-1/\kappa}$.
3: For each edge $e \in T$, determine if e is a **bridge** in the graph $G_i + T$.
4: Let $Y_{e,i} = \begin{cases} 1 & \text{if } e \text{ is not a bridge in the graph } G_i + T. \\ 0 & \text{otherwise.} \end{cases}$
5: **end for**
6: If there is $e \in T$ such that $\sum_{i=1}^{k} Y_{e,i} \leq k/2 + \epsilon' k/8$, then return the cut $C(T, e)$

Algorithm 2: Test(T, κ). Test(T, κ) returns a cut whose weight is at most $(1 + \epsilon')\kappa$ w.h.p. if there exists a cut in G induced by T with weight at most κ. Note that the sample probability $1 - 2^{-1/\kappa} = \Theta(1/\kappa)$.

Lemma 2.3. *If T induces a cut $C(T, e)$ with weight at most κ, then Test(T, κ) returns a cut w.h.p. Moreover, any cut returned by the algorithm has weight at most $(1 + \epsilon')\kappa$ w.h.p.*

Proof. Consider a cut $C(T, e)$. First observe that G_i contains an edge crossing $C(T, e)$ if and only if e is not a bridge in the graph $G_i + T$. Therefore, $\mathrm{E}[Y_{e,i}] = 1 - (1 - (1 - 2^{-1/\kappa}))^{w(C(T,e))} = 1 - 2^{-w(C(T,e))/\kappa}$.

If there is $C(T, e) \leq \kappa$, then $\mathrm{E}[Y_{e,i}] \leq 1/2$ and $\mathrm{E}[\sum_i Y_{e,i}] \leq k/2$. By Hoeffding's inequality, $\Pr(\sum_i Y_{e,i} > k/2 + \epsilon' k/8) \leq \Pr(\sum_i Y_{e,i} > \mathrm{E}[\sum_i Y_{e,i}] + \epsilon' k/8) \leq e^{-\frac{2(\epsilon' k/8)^2}{k}} = e^{-\epsilon'^2 k/32} = 1/\operatorname{poly}(n)$. By taking the union bound over the $n-1$ cuts induced by T, we conclude that w.h.p. the algorithm will return a cut if there is cut whose weight is at most κ.

On the other hand if $w(C(T, e)) > (1 + \epsilon')\kappa$, then $\mathrm{E}[Y_{e,i}] = 1 - 2^{-1-\epsilon'} \geq 1/2 + \epsilon'/4$ when $\epsilon' \leq 1$, since $2^{-\epsilon'} \leq 1 - \epsilon'/2$ when $\epsilon' \leq 1$. So $\mathrm{E}[\sum_i Y_{e,i}] \geq k/2 + \epsilon' k/4$. By Hoeffding's inequality, $\Pr(\sum_i Y_{e,i} \leq k/2 + \epsilon' k/8) \leq \Pr(\sum_i Y_{e,i} \leq \mathrm{E}[\sum_i Y_{e,i}] - \epsilon' k/8) \leq e^{-\frac{2(\epsilon' k/8)^2}{k}} = e^{-\epsilon'^2 k/32} = 1/\operatorname{poly}(n)$. By taking the union bound over the $n - 1$ cuts induced by T, we conclude the cut returned by the algorithm has weight at most $(1 + \epsilon')\kappa$ w.h.p. $\square$

2.2 Computing the Bridges

Given a subgraph G_i of G, it remains to show how to determine what edges of T are bridges in the subgraph $T + G_i$ in $\tilde{O}(D + \sqrt{n})$ rounds. Thurimella [17] gave an algorithm for computing the biconnected components of the underlying graph in $\tilde{O}(D + \sqrt{n})$ rounds. With simple modifications, it can be applied to compute which edges of T are bridges in the *subgraph* G_i of the underlying graph G. Note that even we have the algorithm for computing the bridges of T in $G + T$, it is not clearly whether we can directly simulate it to compute the bridges of T in $G_i + T$, because we want the running time to depend on the diameter of G rather than that of G_i. Therefore, we describe the algorithm and necessary changes below.

Fix a root r in T. Let $pre(u) \in [0, n - 1]$ be the preorder number which denote the time u is visited if we perform a depth-first search on T starting at r. Denote the subtree rooted at u by T_u

and let $size(u)$ be the size of T_u. Let

$$low(u) \stackrel{\text{def}}{=} \min \begin{cases} pre(u) & \\ low(v) & v \text{ is a child of } u \text{ in } T \\ pre(v) & uv \in G_i \ ^{\dagger} \end{cases}$$

$$high(u) \stackrel{\text{def}}{=} \max \begin{cases} pre(u) & \\ high(v) & v \text{ is a child of } u \text{ in } T \\ pre(v) & uv \in G_i \ ^{\dagger} \end{cases}$$

Lemma 2.4. *Let $uv \in T$ with v being a child of u, i.e. $pre(u) < pre(v)$. uv is a bridge if and only if $low(v) \geq pre(v)$ and $high(v) \leq pre(v) + size(v) - 1$.*

Proof. First notice that every vertex $x \in T_v$ must have $pre(x) \in [pre(v), pre(v) + size(v) - 1]$. If uv is a bridge, then no descendent of v will be adjacent to anything outside the subtree rooted at v, for otherwise a cycle passing uv will be created. Therefore, $low(v), high(v) \in [pre(v), pre(v) + size(v) - 1]$.

On the other hand, if $low(v) < pre(v)$ or if $high(v) \geq pre(v) + size(v)$, then there exists a vertex $y \in T_v$ and $z \notin T_v$ such that y and z are adjacent. Since $z \notin T_v$, there must exists a path from z to u such that it does not pass uv. Therefore, $u \to v \rightsquigarrow y \to z \rightsquigarrow u$ forms a cycle and uv is not a bridge. $\square$

Remark 2.5. *Note that the second condition $high(v) \leq pre(v) + size(v) - 1$ is needed because T is not necessarily a DFS tree.*

Now it remains to show how to compute $pre(u)$, $low(u)$, and $high(u)$ in $\tilde{O}(D + \sqrt{n})$ time. It is explicitly described in [17] how to compute $pre(u)$. Note that $pre(u)$ is independent of the G_i. Although $low(u)$ and $high(u)$ depend on G_i, they can be computed in a similar way in $\tilde{O}(D + \sqrt{n})$ time. For completeness, we describe how to compute these functions in the following.

Lemma 2.6 ([4, 11])**.** *A tree of n vertices can be divided into $O(\sqrt{n})$ connected subgraphs each of diameter $O(\sqrt{n})$ in $O(\sqrt{n} \log^* n)$ time*

First, use Lemma 2.6 to decompose the rooted tree T into components $F_1, \ldots F_{O(\sqrt{n})}$. For each component F_i, there is a root r_i which is either the root of T, r, or the unique vertex in F_i connecting to its parent outside F_i. It is shown in [14] that the root r is able to downcast distinct messages of size $O(\log n)$ to each of r_i in $O(D + \sqrt{n})$ time. Conversely, it is possible for each of the r_i to upcast a message of size $O(\log n)$ to the root r in $O(D + \sqrt{n})$ time.

Suppose each vertex has a unique ID. The component ID of F_i is defined to be the ID of r_i. The component ID can be broadcast to every vertex in the component in $O(\sqrt{n})$ rounds. We can then assume that the root r knows the topology of the contracted tree where each component is contracted into a single vertex. This can be done if every root r_i upcasts a message about the component ID of its parent and itself.

To compute $pre(u)$, each root r_i in each component first calculate the size of F_i then upcast it to r. Since r knows the topology of the contracted tree, r can calculate the size of each subtree rooted at each of r_i. Then r downcasts the size of subtree rooted at r_i back to r_i. Now each F_i

†It can be the case that v is the parent of u in T, which happens when there are parallel edges between u and v in $G_i + T$, and one of them is in T. Note that an edge is not a bridge if it is a multiedge.

computes its preorder number internally in $O(\sqrt{n})$ time assuming r_i has number 0. During the computation, each r_i also records what its preorder number is supposed to be if the depth-first search started from the root of its parent component. Finally, each r_i upcasts this number to r and then r computes the correct offset for each subtree and downcasts the offsets back to the r_i. After adding the offset internally, we get the correct preorder number.

To compute $low(u)$, initially each vertex u computes $\min(pre(u), \min_{uv \in G_i} pre(v))$ in constant rounds. Then the problem becomes aggregating the minimum in the subtree T_u for each u. First, each r_i computes the minimum in F_i in $O(\sqrt{n})$ time and then upcasts to r. Using the information, r calculates the minimum of the subtrees rooted at each r_i and downcasts to each r_i. Now each r_i sends the minimum to its parent via the inter-component links. The parent replace its minimum if it is smaller. Finally, each component F_i internally updates the minimum toward the root r_i. Then each vertex has the correct minimum. $high(u)$ can be computed in the same way.

Therefore, the step of computing the bridges in T of $G_i + T$ takes $O(D + \sqrt{n}\log^* n)$ time. Each invocation of Test(T, κ) takes $O(\frac{\log n}{\epsilon^2}(D + \sqrt{n}\log^* n))$ time.

2.3 Running Time

Now we analyze the running time of Algorithm 1. The outerloop runs for $O(\log n)$ iterations. Therefore, the tree packing, Line 7, is executed $O(\log n)$ times, each taking $O(\log^{10} n/\epsilon^{14}(D + \sqrt{n}\log^* n))$ rounds.

Let $k = O(\log(nW))$ be the largest index such that $X_k \leq nW$. The total number of iterations that the innerloop runs is at most

$$\sum_{i=0}^{k} \log_{1+\epsilon'}\left(\frac{1+\epsilon'}{1-\epsilon'} \cdot \frac{X_{i+1}}{X_i}\right) = O(k) + \sum_{i=0}^{k} \log_{1+\epsilon'} \frac{X_{i+1}}{X_i} = O(k) + \log_{1+\epsilon'}(X_{k+1}) = O(\log n/\epsilon)$$

Therefore, Test(T, κ) is invoked at most $O((\log n/\epsilon)\cdot(\log^{10} n/\epsilon^{14}))$ times, each taking $O((\log n/\epsilon^2)(D + \sqrt{n}\log^* n))$ rounds.

The total running time is

$$\begin{aligned} &O(\log n \cdot (\log^{10} n/\epsilon^{14})(D + \sqrt{n}\log^* n)) + (\log^{11} n/\epsilon^{15}) \cdot ((\log n/\epsilon^2)(D + \sqrt{n}\log^* n)) \\ &= O((\log^{12} n/\epsilon^{17}) \cdot (D + \sqrt{n}\log^* n)) = \tilde{O}(D + \sqrt{n}) \end{aligned}$$

Remark 2.7. *The total iterations of the outerloop and innerloop in Algorithm 1 can be reduced to $O(1)$ and $O(1/\epsilon)$ by first approximating λ within constant factor by Ghaffari and Kuhn's algorithm. Then, we can reduce our running time to $O((\log^{11} n/\epsilon^{17})(D + \sqrt{n}\log^* n))$.*

The exponent of the $\log n$ and the ϵ in our running time depends heavily on the size of the greedy tree packing in Lemma 2.1. If one can show that $O(\lambda^a \log^b n)$ trees is sufficient, then our running time can be improved to $O((\log^{2+a+b} n/\epsilon^{2a+3}) \cdot (D + \sqrt{n}\log^* n))$ rounds. Using Ghaffari and Kuhn's algorithm to approximate λ within a constant (Remark 2.7), we can get a running time of $O((\log^{1+a+b} n/\epsilon^{2a+3} + (\log^2 n \log\log n)/\epsilon^5) \cdot (D + \sqrt{n}\log^* n))$. For comparison, Karger [9] showed that a greedy tree packing of size $O(\lambda \log n)$ is enough for any minimum cut to be crossed at most twice by some tree. It will be interesting to see if the number of trees in Lemma 2.1 can be reduced.

References

[1] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

[2] M Elkin. Distributed approximation: A survey. *SIGACT News*, 35(4):40–57, 2004.

[3] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259 – 273, 1995.

[4] J. A. Garay, S. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. In *Proc. 34th Symposium on Foundations of Computer Science (FOCS)*, pages 659–668, 1993.

[5] M. Ghaffari and F. Kuhn. Distributed minimum cut approximation. In *Proc. 27th Symposium on Distributed Computing (DISC)*, volume 8205, pages 1–15. 2013.

[6] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 21–30, 1993.

[7] D. R. Karger. Random sampling in cut, flow, and network design problems. In *Proce. 26th ACM Symposium on Theory of Computing (STOC)*, pages 648–657, 1994.

[8] D. R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 424–432, 1994.

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

[10] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 757–765, 1993.

[11] S. Kutten and D. Peleg. Fast distributed construction of k-dominating sets and applications. In *Proc. 14th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–251, 1995.

[12] D. W. Matula. A linear time $2+\epsilon$ approximation algorithm for edge connectivity. In *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 500–504, 1993.

[13] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discret. Math.*, 5(1):54–66, February 1992.

[14] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[15] M. Stoer and F. Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, July 1997.

[16] M. Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.

[17] R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *Journal of Algorithms*, 23(1):160 – 179, 1997.