Distributed Computation of Mixing Time

Anisur Rahaman Molla^{*}
ACM Unit
Indian Statistical Institute
Kolkata 700108, India
anisurpm@gmail.com

Gopal Pandurangan[†]
Department of Computer Science
University of Houston
Houston, Texas 77204, USA
gopalpandurangan@gmail.com

ABSTRACT

The mixing time of a graph is an important metric, which is not only useful in analyzing connectivity and expansion properties of the network, but also serves as a key parameter in designing efficient algorithms. We present an efficient distributed algorithm for computing the mixing time of undirected graphs. Our algorithm estimates the mixing time τ_s (with respect to a source node s) of any n -node undirected graph in $O(\tau_s \log n)$ rounds. Our algorithm is based on random walks and require very little memory and use lightweight local computations, and work in the *CONGEST* model. Hence our algorithm is scalable under bandwidth constraints and can be an helpful building block in the design of topologically aware networks.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems; G.2.2 [Discrete Mathematics]: Graph Theory—*Graph algorithms, Network problems*

Keywords

Distributed Algorithm, Random Walk, Mixing Time, Conductance, Spectral Properties

1. INTRODUCTION

Mixing time of a random walk in a graph is the time taken by a random walk to converge to the *stationary distribution* of the underlying graph. It is an important parameter which is closely related to various key graph properties such as graph expansion, spectral gap, conductance etc. Mixing time is related to the *conductance* Φ and *spectral gap*

^{*}Work done while at Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany. Research partially supported by ERC Grant No. 336495 (ACDC).

[†]Supported, in part, by NSF grants CCF-1527867, CCF-1540512, and IIS-1633720.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDCN '17, January 04 - 07, 2017, Hyderabad, India

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4839-3/17/01...\$15.00

DOI: <http://dx.doi.org/10.1145/3007748.3007784>

$(1 - \lambda_2)$ of a n -node graph due to the known relations ([9]) that $\frac{1}{1-\lambda_2} \leq \tau \leq \frac{\log n}{1-\lambda_2}$ and $\Theta(1 - \lambda_2) \leq \Phi \leq \Theta(\sqrt{1 - \lambda_2})$, where λ_2 is the second largest eigenvalue of the adjacency matrix of the graph. Small mixing time means the graph has high expansion and spectral gap. Such a network supports fast random sampling (which has many applications [7]) and low-congestion routing [8]. Moreover, the spectral properties reveal a lot about the network structure [6]. Mixing time is also useful in designing efficient randomized algorithms in communication networks [1, 2, 5, 6, 11, 15].

In this paper, we focus on developing a simple and efficient distributed algorithm for computing mixing time in graphs. Given an undirected n -node network G and a source node s , our algorithm estimates the mixing time τ_s for the source node in $O(\tau_s \log n)$ rounds and achieves high accuracy of estimation. We note that this running time is non-trivial in the *CONGEST* model.¹ Moreover, our algorithm is simple, lightweight (low-cost computations within a node) and easy to implement.

Our approach crucially uses random walks. Random walks are very local and lightweight and require little index or state maintenance that makes it attractive especially to self-organizing and dynamic networks. Our approach, on a high-level, is based on efficiently performing many random walks from a particular node and computing the fraction of random walks that terminate over each node. We show that this fraction estimates the random walk probability distribution. Our approach achieves very high accuracy which is a requirement in some applications [4, 5, 11, 18]. Due to space constraints, we omit some proofs from the article which can be found in the full version [14].

We remark that we can compute the mixing time of a graph in $O(m)$ time in the *CONGEST* model, where m is the number of edges in the graph. In the worst case, the mixing time τ could be $O(n^3)$ for some graph, e.g., the Lollipop graph. On the other hand, the complete graph topology can be collected to a single node (e.g., by electing a leader which takes $O(D)$ rounds [12]) by flooding in $O(m)$ rounds and then the node can compute the mixing time locally. The source node can compute the number of edges in the beginning (which can be done in $O(D)$ time) and then run the two algorithms in parallel and stop when one of them terminates.

1.1 Distributed Computing Model

We model the communication network as an undirected, unweighted, connected graph $G = (V, E)$, where $|V| = n$

¹In the *LOCAL* model, all problems can be trivially solved in $O(D)$ rounds.

and $|E| = m$. Every node has limited initial knowledge. Specifically, assume that each node is associated with a distinct identity number (e.g., its IP address). At the beginning of the computation, each node v accepts as input its own identity number and the identity numbers of its neighbors in G . We also assume that the number of nodes and edges i.e., n and m (respectively) are given as inputs. (In any case, nodes can compute them easily through broadcast in $O(D)$ time; this does not affect the asymptotic bounds of our algorithm.) The nodes are only allowed to communicate through the edges of the graph G . We assume that the communication occurs in synchronous *rounds*. We will use only small-sized messages. In particular, in each round, each node v is allowed to send a message of size $O(\log n)$ bits through each edge $e = (v, u)$ that is adjacent to v . The message will arrive to u at the end of the current round. This is a widely used standard model known as the *CONGEST model* to study distributed algorithms (e.g., see [17, 16]) and captures the bandwidth constraints inherent in real-world computer networks.

We focus on minimizing the *the running time*, i.e., the number of *rounds* of distributed communication. Note that the computation that is performed by the nodes locally is “free”, i.e., it does not affect the number of rounds; however, we will only perform polynomial cost computation locally (in particular, very simple computations) at any node.

1.2 Random Walk Preliminaries

We consider a *simple random walk* in an undirected graph: In each step the walk goes from the current node to a random neighbor i.e., from the current node v , the probability of moving to node u is $\Pr(v, u) = 1/d(v)$ if $(v, u) \in E$, otherwise $\Pr(v, u) = 0$, where $d(v)$ is the degree of v .

Suppose a random walk starts at some vertex v in a graph G . Let $\mathbf{P}_0$ be the initial distribution with probability 1 at the node v and zero at all other nodes. Then we get a probability distribution $\mathbf{P}_t$ at time t starting from the initial distribution $\mathbf{P}_0$. Note that we hide the starting node in the notation of the probability distribution $\mathbf{P}_t$. We hope that it is clear to the reader from the context. (Informally) we say that the distribution $\mathbf{P}_r$ is stationary (or steady-state) for the graph G when no further changes on the distribution i.e., $\mathbf{P}_{r+t} = \mathbf{P}_r$ for $t \geq 1$. It is known that the stationary distribution of an undirected connected graph is a well defined quantity which is $(\frac{d(v_1)}{2m}, \frac{d(v_2)}{2m}, \dots, \frac{d(v_n)}{2m})$, where $d(v_i)$ is the degree of the node v_i . We denote the stationary distribution vector by π , i.e., $\pi(v) = d(v)/2m$ for every node v . The stationary distribution of a graph is fixed irrespective of the starting node of a random walk, however, the time to reach to the stationary distribution could be different for the different starting nodes.

The *mixing time* of a random walk starting from the source node v , denoted by τ_v , is the time (or number of steps) taken to reach to the stationary distribution of the graph. The mixing time, denoted by τ , is the maximum mixing time among all (starting) nodes in the graph. The formal definitions are given below.

DEFINITION 1.1. ($\tau_v(\epsilon)$ -mixing time for the source v and $\tau(\epsilon)$ -mixing time of the graph)

Define $\tau_v(\epsilon) = \min\{t : \|\mathbf{P}_t - \pi\|_1 \leq \epsilon\}$, where $\|\cdot\|_1$ is the L_1 norm. Then $\tau_v(\epsilon)$ is called the ϵ -near mixing time for any ϵ in $(0, 1)$. The mixing time of the graph is denoted by $\tau(\epsilon)$

and is defined by $\tau(\epsilon) = \max\{\tau_v(\epsilon) : v \in V\}$. It is clear that $\tau_v(\epsilon) \leq \tau(\epsilon)$.

We note that mixing time definition usually takes ϵ to be $1/2e$. Our goal is to estimate $\tau_v(\epsilon)$ for any small $\epsilon \in (0, 1)$, say $\epsilon = 1/n^2$. We omit ϵ from the notation when it is understood from the context. The definition of τ_v is consistent due to the following standard monotonicity property of the random walk probability distributions. The proof can be found in [14] or from the Exercise 4.3 in [13].

LEMMA 1.2. $\|\mathbf{P}_{t+1} - \pi\|_1 \leq \|\mathbf{P}_t - \pi\|_1$.

1.3 Related Work

Das Sarma et al. [7] presented a fast decentralized algorithm for estimating mixing time, conductance and spectral gap of the network. In particular, they show that given a starting node s , the mixing time with respect to s , i.e., τ_s , can be estimated in $\tilde{O}(n^{1/2} + n^{1/4}\sqrt{D\tau_s})$ rounds. This gives an alternative algorithm to the only previously known approach by Kempe and McSherry [10] that can be used to estimate τ_s in $\tilde{O}(\tau_s)$ rounds. In fact, the work of [10] does more and gives a decentralized algorithm for computing the top k eigenvectors of a weighted adjacency matrix that runs in $O(\tau \log^2 n)$ rounds if two adjacent nodes are allowed to exchange $O(k^3)$ messages per round, where τ is the mixing time and n is the size of the network.

The algorithm of Das Sarma et al. [7] is based on sampling nodes by performing sub-linear time random walks of certain length and comparing the distribution with the stationary distribution. Their algorithm can be faster than our approach in certain cases, but slower in some cases than ours. In particular, if τ is smaller than $\max\{\sqrt{n}, n^{1/4}\sqrt{D}\}$, then our algorithm is faster. Also there is a grey area for the accuracy parameter ϵ for which their algorithm cannot estimate the mixing time. They use a testing result from Batu et al. [3] to determine if the two distributions are close enough. This test may fail if the difference between the two distributions falls in a certain interval. More precisely, the algorithm of Das Sarma et al. estimates the mixing time for accuracy parameter $\epsilon = 1/(2e)$ with respect to a source node v , $\tau_v(1/2e)$ as follows: the estimated value will be between the true value and $\tau_v(O(1/(\sqrt{n} \log n)))$. In contrast, our algorithm estimates $\tau_v(\epsilon)$, for even small values of ϵ , say $\epsilon = 1/n^2$: the estimated value will be between the true value and $\tau_v(1/n^2)$.

2. ALGORITHM FOR MIXING TIME

We present an algorithm to compute the mixing time of a graph G from a specified source node. In other words, we present an algorithm which finds a length ℓ such that the probability distribution of a random walk of length ℓ reaches close to the stationary distribution. The main idea of our algorithm is to perform many random walks from the source node of some length ℓ in parallel. After ℓ steps, every node u estimates the probability distribution $\mathbf{P}_\ell(u)$ as the fraction of random walks that terminate at u over all the walks. Then we compare the estimated distribution of $\mathbf{P}_\ell$ with the stationary distribution π to determine if they are sufficiently close; otherwise, we double the length ℓ and retry. Once we find the correct consecutive lower and upper bound of the length, a binary search will determine the mixing length (up to the allowed accuracy parameter ϵ). The monotonicity property (cf. Lemma 1.2) admits the binary search.

Our algorithm starts with $\ell = 1$ and runs $K = 80n^8 \log n$ random walks of length ℓ from a source node, say node s . When the difference (i.e., the L_1 -norm difference) between the estimated ℓ -length walk distribution of $\mathbf{P}_\ell$ with the stationary distribution π is greater than ϵ , ℓ is doubled and retried. This process is repeated to identify the largest ℓ such that the difference is greater than ϵ and the smallest ℓ such that the difference is less than ϵ . These give lower and upper bounds on the required τ_s respectively. In fact, the upper bound (i.e., the smallest ℓ such that the difference is less than ϵ) is at most twice as τ_s , since we are doubling the length each time. Then a binary search between lower and upper bounds will determine the exact mixing time τ_s . Note that the monotonicity property (cf. Lemma 1.2) guarantees that once the difference becomes less than ϵ , then it would be always less for any larger length. We show in the analysis that the estimated probability of $\mathbf{P}_\ell(u)$ is close to the actual probability for every node u in each step of the walk (cf. Lemma 2.1). The pseudocode is given in Algorithm 1.

We show (in the next section) that the above algorithm estimates mixing time accurately with high probability². The main technical challenge in implementing the above method in CONGEST model is that performing many walks from a source node in parallel can create a lot of congestion. Our algorithm uses a crucial property of random walks to overcome the congestion (see e.g., [6, 7]). In particular, we show that there will be no congestion in the network even if we perform up to a polynomial (in n) number of random walks from the source node in parallel (cf. Lemma 2.3). The basic idea is to send the *count* of the number of random walks that pass through an edge. As random walks are memoryless processes, it is sufficient to send the number of walks traversing an edge in a given round to estimate $\mathbf{P}_\ell$. Since this number is polynomially bounded, $O(\log n)$ bits suffice. Therefore, it is easy to see that performing ℓ -length of random walks finishes in $O(\ell)$ rounds in CONGEST model. We show that our algorithm computes mixing time accurately in $O(\tau_s \log n)$ rounds with high probability (cf. Theorem 2.4).

2.1 Reducing the number of random bits

To perform $K = 80n^8 \log n$ random walks can require at least so many random bits per node (as well as that much time), if done in a straightforward manner. We can make the algorithm more lightweight by doing only $O(d(u) \log n)$ work and only so many random coin tosses per node. Instead of doing coin-flips for each of the tokens separately, we do the following to reduce the number of random coin-flips overhead at each node. Nodes which have less than $(\text{degree} \times \log n)$ tokens, will select an edge randomly for each token i.e., they will do the coin-flips for each token. At any round, if a node u has tokens T^u such that $T^u \geq d(u) \log n$, then u forwards the average $T^u/d(u)$ tokens to all of its neighbors (there are $d(u)$ many). We show that this simple averaging approach has the bias of at most $O(\log n)$ with high probability. In fact, if we perform coin-flips for each token i.e., select a neighbor randomly for each token then, in expectation, $T^u/d(u)$ tokens will choose to go to one neighbor, where T^u is the total number of tokens at u . Then by a Chernoff bound, it is easy to show that the deviation from the mean is at most $O(\log n)$ with high probability. That is, the bias is $O(\log n)$ tokens per edge with high probability. Therefore, the total bias at a node u is

²With high probability means with probability at least $1 - \frac{1}{n}$.

$O(d(u) \log n)$ in a single round ($O(\log n)$ bias coming from $d(u)$ neighbors). Hence, total bias at node u until the algorithm stops is $O(\tau_s d(u) \log n)$, since the algorithm runs for $O(\tau_s)$ rounds. This quantity is at most $n^4 \log n$, since $d(u) < n$ and $\tau_s \leq O(n^3)$ in worst case. Therefore, the deterministic averaging method has the additional approximation error at most $(n^4 \log n)/K = 1/n^4$, which is negligible compared to the considered error $\epsilon = 1/n^2$.

Algorithm 1 ESTIMATEMIXINGTIME

Input: A graph $G = (V, E)$ and a source node s .

Output: $\tau_s(1/n^2)$ (mixing time starting from the node s).

- 1: Node s creates a BFS tree via flooding (rooted at s) so that each node knows their parent in the BFS tree.
 - 2: Node s broadcasts the value $K = 80n^8 \log n$ to all the other nodes via broadcasting over the tree.
 - 3: **for** $h = 0, 1, 2, \dots$ **do**
 - 4: $\ell \leftarrow 2^h$
 - 5: Node s creates $K = 80n^8 \log n$ random walk tokens.
 - 6: **for** round $i = 1, 2, \dots, \ell$ **do**
 - 7: Each node v holding at least one token, does the following in parallel:
 - 8: For every neighbor w , set $T_w^v = 0$ // [T_w^v indicates the number of tokens chosen to move to w from v at round i]
 - 9: For all the tokens calculate the number T_w^v for each neighbor w (as described in Section 2.1).
 - 10: Send the counter number T_w^v to the neighbor w .
 - 11: **end for**
 - 12: Each node w counts the total number of tokens it holds. Say, $\zeta_w = \sum_{v \in N(w)} T_w^v$.
 - 13: Each node w sends the absolute difference value $\partial_w = |\frac{\zeta_w}{K} - \frac{d(w)}{2m}|$ to node s through the BFS tree. Each node sums up the ∂_w values of their children and forwards the sum to its parent.
 - 14: Node s locally checks:
 - 15: **if** $(\sum_{w \in V} \partial_w \leq 1/n^2)$ **then**
 - 16: BREAK;
 - 17: **else**
 - 18: Continue from Step 4 for the next (doubling) value of ℓ .
 - 19: **end if**
 - 20: **end for**
 - 21: Let's say $\ell = 2^l$ for some integer l .
 - 22: Perform binary search from 2^{l-1} to 2^l to find an integer ℓ such that:
 - 23: For length ℓ , $\sum_{w \in V} \partial_w \leq 1/n^2$, and for length $\ell - 1$, $\sum_{w \in V} \partial_w > 1/n^2$, by computing from Step 5 to Step 15.
 - 24: Output ℓ .
-

2.2 Analysis of the Algorithm

We first show a result which will imply that our algorithm correctly estimates the mixing time for a given source node. Let $\mathbf{P}_\ell(u)$ be the probability that a random walk of length ℓ terminates over node u , starting from the given source node. We show that the above process of performing many random walks and then computing the fraction of walks that stop at u can approximate $\mathbf{P}_\ell(u)$ with high accuracy for every node u . For this, we can ignore the probability $\mathbf{P}_\ell(u)$ whose value is less than $1/n^4$. Because in the worst case, there may be at most n such vertices and the sum of these ignored probabilities is less than $1/n^3$, which is negligible compared to $1/n^2$,

the estimation error. (Here we assume that the mixing time estimation error ϵ is $1/n^2$ — see Definition 1.1). However, one can achieve much more accuracy by performing a larger number of random walks. In the following lemma, we show that if we perform $K = 80n^8 \log n$ random walks from a source node s , then the above algorithm can estimate $\mathbf{P}_\ell(u)$ at any node u for which $\mathbf{P}_\ell(u) \geq 1/n^4$.

LEMMA 2.1. *If the probability of an event X occurring is p such that $p \geq 1/n^4$, then in $t = 80n^8 \log n$ trials, the fraction of times the event X occurs is $p \pm \frac{1}{n^8}$ with high probability.*

PROOF. The proof follows from the standard Chernoff bound. We also included in the full version [14]. $\square$

LEMMA 2.2. *The algorithm ESTIMATEMIXINGTIME approximates the probability $\mathbf{P}_\ell(u)$ for every length ℓ and for all node u (such that $\mathbf{P}_\ell(u) \geq 1/n^4$) with high probability.*

PROOF. Let's first consider a particular length ℓ . Suppose the algorithm ESTIMATEMIXINGTIME outputs the estimated probability $\tilde{\mathbf{P}}_\ell(u)$ for each node u for the length ℓ . It follows from the Lemma 2.1 that $\tilde{\mathbf{P}}_\ell(u) = \mathbf{P}_\ell(u) \pm 1/n^6$ with high probability for any node u for which $\mathbf{P}_\ell(u) \geq 1/n^4$ (by taking union bound). Therefore, $|\tilde{\mathbf{P}}_\ell(u) - \mathbf{P}_\ell(u)| \leq 1/n^6$ for any node u (such that $\mathbf{P}_\ell(u) \geq 1/n^4$). Since ℓ can be arbitrary, the lemma holds for every length ℓ . $\square$

Before proceeding to the main result (cf. Theorem 2.4) of this section, we prove a crucial lemma on the congestion of our algorithm. The lemma below guarantees that there will be no congestion even if we perform a polynomial number of random walks in parallel in the network. While the proof intuition is discussed before, the full proof is included in [14].

LEMMA 2.3. *If every node perform at most a polynomial (in n) number of random walks in parallel then there will be no congestion in the network.*

THEOREM 2.4. *Given a graph G and a source node s , the algorithm ESTIMATEMIXINGTIME takes $O(\tau_s \log n)$ rounds and finds the mixing time $\tau_s(1/n^2)$ with high probability.*

PROOF. Correctness. The Lemma 2.2 says that when ℓ reaches the mixing time of the graph, the estimated probability $\tilde{\mathbf{P}}_\ell(u)$ of $\mathbf{P}_\ell(u)$ will be close to the stationary distribution $d(u)/2m$ for each node u . Hence, the L_1 -norm difference is $\sum_{u \in V} |\tilde{\mathbf{P}}_\ell(u) - d(u)/2m| \leq 1/n^5 + 1/n^3 < 1/n^2$. The term $1/n^3$ is coming from the fact that we neglected those probabilities such that $\mathbf{P}_\ell(u) < 1/n^4$; so summing up all of them could be at most $1/n^3$. Further, it follows from the monotonicity property (cf. Lemma 1.2) that once the norm difference becomes less than $1/n^2$, it would always be less than $1/n^2$ for any larger length of the random walk. Hence, our algorithm terminates when the mixing time is correctly estimated (cf. Definition 1.1).

Running time. To estimate the mixing time, we try out increasing values of ℓ that are powers of 2. Once we find the right consecutive powers of 2, the monotonicity property admits a binary search to determine the exact value. Since each node u knows its own stationary probability (determined just by its degree), they can compute the difference $\partial_u = |\frac{\zeta_u}{K} - \frac{d(u)}{2m}|$ locally and send it to the source node s . Recall that ζ_u is the number of walks that stop at u after ℓ steps

and K is the total number of random walks started initially. The node s eventually collects the sum of all ∂_u s and checks if $\sum_{u \in V} \partial_u$ is less than $1/n^2$. Performing K random walks of length ℓ can be done in $O(\ell)$ rounds, as there is no congestion (cf. Lemma 2.3). Further, sending the sum of all ∂_u s to the source node s , can be done in $O(D)$ rounds (D is the diameter). Since this can be done by an upcast through the BFS tree rooted at s (see Step 13 of Algorithm 1). Hence, for a particular ℓ , it requires $O(\ell + D)$ rounds. Therefore, total time required to compute the mixing time for a source node s is $O(\sum_{i=1}^{\log \tau_s} (\ell + D))$ rounds which is $O((\tau_s + D) \log n)$ rounds as ℓ is at most τ_s which is polynomially bounded. Notice that when $\ell \geq \tau_s(1/n^2)$, then the L_1 difference between estimated distribution and stationary distribution becomes less than $1/n^2$ (correctness follows from the Lemma 2.2). This gives the estimated mixing length at most twice as $\tau_s(1/n^2)$, since we are doubling the length each time. Then a binary search will determine the exact mixing time $\tau_s(1/n^2)$. The binary search takes at most $O((\tau_s + D) \log n)$ rounds. Therefore, the algorithm finishes in $O(\tau_s \log n)$ rounds, since $D \leq \Omega(\tau_s)$ for any graph. $\square$

3. REFERENCES

- [1] J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, and E. Upfal. Search and storage in dynamic peer-to-peer networks. In *SPAA*, pages 53–62, 2013.
- [2] J. Augustine, G. Pandurangan, and P. Robinson. Fast byzantine agreement in dynamic networks. In *PODC*, pages 74–83, 2013.
- [3] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *FOCS*, pages 442–451, 2001.
- [4] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Sparse cut projections in graph streams. In *ESA*, pages 480–491, 2009.
- [5] A. Das Sarma, A. R. Molla, and G. Pandurangan. Distributed computation of sparse cuts via random walks. In *ICDCN*, pages 6:1–6:10, 2015.
- [6] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal. Fast distributed pagerank computation. *Theor. Comput. Sci.*, 561:113–121, 2015.
- [7] A. Das Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Distributed random walks. *J. ACM*, 60(1):2, 2013.
- [8] C. Gkantsidis, M. Mihail, and A. Saberi. Throughput and congestion in power-law graphs. In *SIGMETRICS*, pages 148–159, 2003.
- [9] M. Jerrum and A. Sinclair. Approximating the permanent. *SICOMP*, 18(6):1149–1178, 1989.
- [10] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *JCSS*, 74(1):70–83, 2008.
- [11] F. Kuhn and A. R. Molla. Distributed sparse cut approximation. In *OPDIS 2015*, volume 46, pages 1–14, 2016.
- [12] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, 2015.
- [13] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing times*. AMS, USA, 2008.
- [14] A. R. Molla and G. Pandurangan. Distributed computation of mixing time. In *arXiv:1610.05646*, 2016.
- [15] G. Pandurangan. Distributed algorithmic foundations of dynamic networks. In *SIROCCO*, pages 18–22, 2014.
- [16] G. Pandurangan and M. Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook*. CRC Press, 2009.
- [17] D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.
- [18] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.