Estimation of edge resistances using MCMC

Karan Chadha Roll No: 140070014

***Abstract*— We provide an iterative Markov Chain Monte Carlo based algorithm for estimating effective edge resistances in a given graph. The sample complexity analysis is provided. The fact that the ordering of edge sets w.r.t resistances converges to the correct order faster than individual edge resistance convergence is illustrated through simulations.**

I. INTRODUCTION

Given a simple connected graph $G(V, E)$, with vertex set V and edge set E, the resistance of the network between two nodes is the resistance that would result by considering the graph as an electrical network and edges as resistances with resistance equal to the edge weights. The effective resistance between two nodes i and j is defined as the potential difference that would result when a unit current source is applied across nodes i and j. The inverse of effective resistance is the effective conductance.

When defining the similarity between two objects, a popular approach is to first map each object of the domain to a point in a metric space, and then use the distance between the geometric embeddings of two objects to define their similarity. *Graph sparsification* is the task of approximating a dense graph by a sparse graph that can be effectively used in place of the dense one. Here, we need the original graph and the new sparse graph to be similar. For effective sparsification, we want to retain most of the important properties of the graph. What properties are important in a graph are defined by the application. Retaining the spectral properties is important in many of the applications. Two graphs are said to be spectrally similar [15] when the quadratic forms of their laplacians not far from each other for any real input vector. Graphs can be effectively mapped to a metric space with the Euclidean distance between points representing nodes being the effective resistance between the two nodes. It has been shown in [13] that fast and effective graph sparsification can be performed by taking each edge in the new graph with probability p_e proportional to $w_e R_e$ with a modified weight. Hence, estimating edge resistances fast helps speed up the process of graph sparsification.

Our goal in this paper is to present an effective resistance estimation algorithm using Markov Chain Monte Carlo techniques where the Markov Chain is a random walk on a graph. A (simple) random walk on a graph picks from the current node, one of its neighbours with equal probability. It can be modeled as a Markov Chain where the state is the node where the walk has reached. The effective resistance of an edge is known to be equal to the probability that the edge appears in a random spanning tree of the graph [11]. We use this fact to find the effective resistance across any edge using Aldous' [2] and Wilson's [16] algorithm to generate random spanning trees.

The next section sets up the framework of the problem and gives the current algorithms for estimating edge resistance. In Section 3, we propose our algorithm to estimate conductances and provide subsequent complexity and numerical analysis. Section 4 consists of an ordinal application of estimating edge conductances.

II. BACKGROUND

Let the given graph be $G(V, E)$, where V represents the set of vertices and E represents the set of edges with $|V| = n$ and $|E| = m$. Let $d(i)$ represent the degree of node i and $\partial(i)$ represent its set of neighbours, i.e. $\partial(i) = \{j|(i, j) \in E\}$. Let $d_{max} = \max_{i\in V} d(i)$ and $d_{avg} = \frac{1}{n}\sum_{i\in V} d(i)$ be the maximum and average degrees of nodes. Let the resistance of an edge (i, j) is represented by R_{ij} and its conductance by $G_{ij} = 1/R_{ij}$

Let $\{X_t, t \geq 1\}$ be the discrete-time Markov Chain associated with the random walk on the graph G with transition probabilities given by $p(j|i)$, where j is the next node and i is the current node.

$$p(j|i) = \begin{cases} \frac{1}{d(i)} & \text{for } j \in \partial(i), \\ 0 & \text{otherwise.} \end{cases}$$

Here, X_t represents the state of the Markov Chain at time t and X_1 is the initial state. Let $\mathcal{L}(X_t)$ represent the distribution of the Markov chain. It will be assumed that the Markov Chain is aperiodic and irreducible, and hence has a unique stationary distribution denoted by π. Since the graph is undirected, the Markov Chain is reversible with the stationary distribution given by $\pi(i) = \frac{d(i)}{2m}$.

We define few standard notions related to Markov Chains below:

- The hitting time (t_v) of the Markov Chain to be the first time it hits vertex v starting at X_1, i.e., $t_v = \min\{t : t \geq 1, X_t = v\}$.
- The corresponding cover time t_C is defined to be the time taken to reach all the vertices. It is easy to see that $t_C = \max_{v\in V} t_v$.
- The mixing time of the Markov chain defined as the time taken by the Markov chain to get close to the

stationary distribution. To be precise,

$$t_{mix} := min\Big\{t \geq 1 \Big| \max_{i \in V} \delta(\mathcal{L}(X_n|X_1 = i), \pi) \leq \frac{1}{4}\Big\},$$

where $\delta(\cdot, \cdot)$ represents the total variational distance between two distributions. It is defined as

$$\delta(P, Q) := \sup_{A \subseteq V} |P(A) - Q(A)|,$$

where P and Q are probability distributions on the alphabet V.

- The commute time ($t_{com}(u, v)$) of a Markov chain is the expected number of steps required by a random excursion starting at u to pass through v and come back again to u.

A. Spielman-Srivastava Scheme

Spielman and Srivastava [13] gave a method for graph sparsification by effective resistances. The crux of their algorithm was a subroutine with a nearly linear time algorithm which builds a data structure from which we can compute the approximate effective resistance between any two nodes in a given graph in $O(\log n)$ time.

To accomplish this, they made use of the well-known fact that the nodes of a graph can be embedded using the Laplacian and the incidence matrix into a Euclidean space with the resulting pairwise distances encoding the effective resistance. The incidence matrix(A) is an $m \times n$ matrix with

$$A(e, v) = \begin{cases} +1, & \text{if } v \text{ e's head} \\ -1, & \text{if } v \text{ e's tail} \\ 0, & \text{otherwise.} \end{cases}$$

Note that, for an undirected graph to define the above incidence matrix, we assign arbitrary but fixed directions to each edge, since the electrical network is a directed graph but the direction is irrelevant for effective resistance. The Laplacian is defined as $L = A^T W A$, where $W_{m \times m}$ is a diagonal matrix with $W(e, e) = w_e$. Here, w_e is the weight assigned to edge e. To compute the effective resistances quickly, it is first noted that

$$\begin{aligned} R_{ij} &= ((\chi_i - \chi_j)^T L^+ (\chi_i - \chi_j)) \\ &= ||W^{1/2} A L^+ (\chi_i - \chi_j)||_2^2, \end{aligned}$$

where χ_i is the vector with i^{th} entry 1 and all others 0, L^+ is the psuedoinverse of the Laplacian of the graph G and $||\cdot||_2$ represents the Euclidean norm. Since the resistance can just be seen as the pairwise distance between the vectors $\{W^{1/2} A L^+ \chi_i\}_{i \in V}$, using the Johnson-Lindenstrauss Lemma [1] these distances are preserved with low distortion $(1 \pm \epsilon)$ on projecting the vectors to a subspace spanned by $O(\log n)$ vectors. Let this projection be $Q_{k \times n}$, where $k = O(\log n)$. Then the resistance is given by $||Z(\chi_i - \chi_j)||_2^2$ with low distortion where $Z = W^{1/2} A L^+$. Now this Z can be approximated (to $\widetilde{Z}$) using the STSolve algorithm proposed in [14] to give a $\widetilde{O}(m \log(r)/\epsilon^2)$ time algorithm, where ϵ represents the distortion due to Johnson-Lindenstrauss lemma and $r = w_{max}/w_{min}$. Once $\widetilde{Z}$ is computed, we can find the effective resistance for any edge (i, j) in $O(\log n)/\epsilon^2$ time by computing the difference of the i^{th} and the j^{th} column of $\widetilde{Z}$ and computing its norm.

B. MCMC Based schemes

The first method for edge conductance estimation using MCMC was proposed by Bora et al. in [3]. In this method they used the fact that the probability that a random walk starting at node i visits node j before returning to i is equal to the equivalent conductance between the two nodes i and j, normalized by the degree of node i. Although the above fact can be used to find the equivalent conductance between any two nodes (not necessarily adjacent), they focused on the case of conductances of edges.

```
Result: Estimated conductance of each edge of the
        given graph G
Input T,G = (V,E);
For each $i \in V$, initialize $N_i = 0$;
For each $(i, j) \in E$, initialize $\widehat{p_{ij}}, \widetilde{p_{ij}} = 0$;
Sample an initial node $X_1$ from a stationary
 distribution; for $t = 1$ to $T$ do
    $i = X_t$;
    for $j \in \partial(i)$ do
        $\widehat{p_{ij}} = \widehat{p_{ij}} N_i + \widetilde{p_{ij}}/(N_i + 1)$;
        $\widetilde{p_{ij}} = 0$;
        $\widetilde{p_{ji}} = 1$;
    end
    $N_i = N_i + 1$;
    Choose $X_{i+1}$ as the next node as identified by the
     walk.
end
For each $(i, j) \in E$,
```

$$\widehat{G_{ij}} = max\Big(1, \frac{d_i}{2}\widehat{p_{ij}} + \frac{d_j}{2}\widehat{p_{ji}}\Big)$$

Algorithm 1: Bora's Scheme

In words:

- $\widetilde{p_{ij}}$ denotes the success or failure of visiting node j in a random walk instance of a path from node i to i.
- N_i represents the number of times one visits node i in the random walk.
- $\widehat{p_{ij}}$ is the running average of $\widetilde{p_{ij}}$. It estimates the probability that node j occurs in a random path from i to i.
- Note that the update above is not for a single node, i.e. the updates in each iteration are not for any fixed node but for all nodes. A random walk is run starting from any node and whenever any node is visited for the second time and thereafter, one random excursion ends and a new one starts signified by increasing the parameter N_i.
- In the inner for loop, since every visit to node i marks the end of a path, $\widehat{p_{ij}}$ is updated using $\widetilde{p_{ij}}$.

Note that although the above algorithm is to find conductances, the values can be inverted to get resistances.

The number of iterations (T) required for a fixed edge $(i, j) \in E$ for this scheme as stated in the Theorem 1 of [3] is

$$\widetilde{O}\Big(D_{ij} \cdot max\{|E|, d_{ij}t_{mix}\} \cdot \frac{1}{\epsilon^2} log\frac{1}{\delta}\Big), \tag{1}$$

where $D_{ij} = max\{d_i, d_j\}$, $d_{ij} = min\{d_i, d_j\}$ $0 < \delta < 1/2$, $0 < \epsilon < d_{ij}^2/(4m)$ and t_{mix} is the mixing time of the Markov chain $\{X_n\}$. The number of time steps given in (1) are sufficient to ensure

$$\mathbb{P}(|\widehat{G}_{ij} - G_{ij}| \geq \epsilon) \leq \delta.$$

Hence, the algorithm requires $O(d_{avg}T)$ computation steps on an average and $O(d_{max}T)$ computation steps in the worst case, and uses $O(mlogT)$ space. It is shown in [3] that the above complexity of number of iterations also holds while considering the relative error on the effective resistances, i.e.

$$\mathbb{P}(\frac{|R_{ij} - \widehat{R}_{ij}|}{R_{ij}} \geq \epsilon) \leq \mathbb{P}(|\widehat{G}_{ij} - G_{ij}| \geq \epsilon) \leq \delta.$$

By union bounding over all edges, we get

$$T = \widetilde{O}\Big(d_{max} \cdot max\{m, d_{max}t_{mix}\} \cdot \frac{1}{\epsilon^2} log\frac{1}{\delta}\Big)$$

steps are enough to ensure

$$\mathbb{P}(\max_{(i,j)\in E} \frac{|R_{ij} - \widehat{R}_{ij}|}{R_{ij}} \geq \epsilon) \leq \delta.$$

Another approach to estimate conductances was suggested in [10] which used the fact that the effective resistance is equal to the probability that an edge appears in a random spanning tree. To simulate random spanning trees, a tree valued markov chain with a uniform stationary distribution over the set of spanning trees was generated. The long run fraction of times an edge appears in a random spanning tree is then simply the effective resistance. This algorithm was computationally cumbersome since it was requires to simulate a tree valued markov chain.

III. The Resistance Estimation Algorithm

The effective resistance of an edge is known to be equal to the probability that the edge appears in a random spanning tree of G. The effective conductance is the inverse of the effective resistance. We estimate this probability using an MCMC approach as given in Algorithm 2. Now, to sample the random spanning trees we use the Aldous' algorithm or the Wilson's algorithm which are described in sections III-A and III-B

```
Result: Estimated resistance of each edge of the given
        graph G
Input T,G = (V,E);
For each $(i, j) \in E$, initialize $N_{(i,j)} = 0$;
Initialize t=1;
while $t \leq T$ do
    Sample a random spanning tree $\mathcal{T}$(Using
      Aldous'/Wilson's Algorithm);
    if $(i, j) \in \mathcal{T}$ then
        $N_{(i,j)} = N_{(i,j)} + 1$ ;
    end
    $\hat{R}_{(ij)}(t) = \frac{N_{(i,j)}}{t}$
end
```

Algorithm 2: Edge Occurrences MCMC Scheme

A. Aldous' Scheme

Consider the random walk $\{X_t, t \geq 1\}$ over the graph G. We assume the initial node (X_1) called root is chosen uniformly at random. Consider the hitting time of hitting a node v as t_v and the cover time to be t_C. Then, from the first t_C steps of the graph we define a subgraph $\mathcal{T}$ of G consisting of all the nodes and $N-1$ edges $\{(X_{t_v-1}, X_{t_v})\}$ for each node $v \in V$. It is easy to see that this is a spanning tree. It can be shown (see Aldous[2]) that a spanning tree constructed this way is equivalent to uniformly sampling spanning trees amongst all spanning trees.

B. Wilson's Scheme

Wilson's algorithm [16] samples a uniform random spanning tree of the graph with expected run-time of the order of mean hitting time of the random walk on the graph. The algorithm is described in Algorithm 3.

In this algorithm, first a random initial node is fixed to be in the tree. Then, a random excursion is started from the first node and keep including the edges in the tree until it reaches a node already in the tree. If a node in the tree is reached, we stop and start a random excursion from the next node which is not in the tree. Note that, here by first node it is meant that the smallest node when the nodes are assigned an arbitrary but fixed ordering. As given in Theorem 2 of [16], for a given root the expected run time to generate a uniform random sampling tree is $O(\sum_{u\in V} \pi(u)t_{com}(u, root))$, where $\pi(\cdot)$ is the stationary distribution of the Markov chain and $t_{com}(u, root)$ is the commute time between two vertices.

C. Complexity and Numerical Experiments

The complexity of our algorithm given in Algorithm 2 depends upon the number of iterations T required and what we do in each iteration. In each iteration, the bottleneck step is to generate a random tree which we do using either Aldous' or Wilson's scheme, the complexities of which are given in III-A and III-B respectively. To find the number of iterations required we give a PAC Learning guarantee using

```
Result: A uniformly sampled spanning Tree$(V, \tilde{E})$
Input G = (V,E);
$\tilde{E} = \phi$;
For each $i \in V$, initialize $InTree[i] = 0$;
root = UnifRand(V);      % Choose a random vertex
NextNode[root] = $\phi$;
InTree[root] = 1;
for $i = 1$ to $n$ do
    CurrentNode = i;
    while InTree[CurrentNode] == 0 do
        NextNode[CurrentNode] = RandomSuccessor;
        CurrentNode = NextNode[CurrentNode];
    end
    CurrentNode = i;
    while InTree[CurrentNode] == 0 do
        InTree[CurrentNode] = 1;
        CurrentNode = NextNode[CurrentNode];
    end
end
for $i = 1$ to $n$ do
    if $i \neq root$ then
        $\tilde{E} = \tilde{E} \cup (i, NextNode[i])$;
    end
end
```

Algorithm 3: Wilson's Algorithm

the following Hoeffding's inequality.

Lemma: (Hoeffding [9]) Let $Z_1, Z_2, \ldots, Z_T$ be independent Bernoulli random variables and $Z = \sum_{i=1}^{T} Z_i$. Then for any $0 < \epsilon < 1$,

$$\mathbb{P}(|Z - \mathbb{E}[Z]| > \epsilon T) \leq 2exp(-2\epsilon^2 T).$$

We use this for each edge (say e) by considering,

$$Z_i(e) = \begin{cases} 1, & \text{if } edge\ e \in T_i \\ 0, & \text{otherwise,} \end{cases}$$

where T_i denotes the tree generated at the i^{th} iteration. Note that for an edge e, $\mathbb{E}[Z(e)] = T \cdot R_e$, where R_e represents the resistance of edge e and $Z(e) = T \cdot \widehat{R}_e$, which is the current estimate of resistance from our algorithm. Hence, using the lemma for edge e, we get for any $0 < \epsilon < 1$,

$$\mathbb{P}(|Z(e) - \mathbb{E}[Z(e)]| > \epsilon T) \leq 2exp(-2\epsilon^2 T).$$

By using the Union Bound over all edges to ensure the resistance of all of them are within ϵ of their true values, we get

$$\mathbb{P}(\bigcup_{e \in E} \{|Z(e) - \mathbb{E}[Z(e)]| > \epsilon T\}) \leq 2m \cdot exp(-2\epsilon^2 T). \quad (2)$$

A PAC(ϵ, δ) guarantee is given by

$$\mathbb{P}(\bigcup_{e \in E} \{|\widehat{R}_e - R_e]| > \epsilon\}) \leq \delta. \quad (3)$$

On comparing equations 2 and 3, we get $T = \lceil log(\frac{2m}{\delta})/\epsilon^2 \rceil$. The expected run time can be found by considering the complexities of Aldous' and Wilson's scheme to be $\lceil log(\frac{2m}{\delta})/\epsilon^2 \rceil \cdot O(\mathbb{E}[T_C])$ and $\lceil log(\frac{2m}{\delta})/\epsilon^2 \rceil \cdot O(\sum_{u \in V} \pi(u) t_{com}(u, root))$ respectively. Since the , Wilson's algorithm is faster than Aldous' algorithm.

The complexity of out algorithm is worse than that of [13] as described in Section II-A. But our algorithm is amenable to parallelization because many random walks can be run on different machines and the results can be averaged to get sharper results. Also, as is the case with many MCMC schemes, the complexity provided is far worse than we see in practice. It is shown in the simulations of [6] that our algorithm is faster than that of [13] in practice.

To compare the performance to [3], we need to consider specific graphs since the mixing times and mean hitting times are of different orders for different graphs. Since effective resistances are useful for graph sparsification, we may consider dense graphs and compare the performance. In dense Erdos-Renyi graphs, it can be seen that $d_{max} \sim \tilde{O}(n)$, it is shown in [12] that the mean hitting time is of the order of number of nodes. Hence, Bora's Scheme runs in $\widetilde{O}(n^2 m \log(\frac{1}{\delta})\frac{1}{\epsilon^2})$ whereas our algorithm with Wilson's scheme runs in $O(n \log(\frac{2m}{\delta})\frac{1}{\epsilon^2})$ which is clearly better than Bora's Scheme.

The simulations performed were as follows. Given a graph G, with a Laplacian L, it is known that the edge resistances between two nodes is given by

$$R_{ij} = ((\chi_i - \chi_j)^T L^+ (\chi_i - \chi_j)),$$

where χ_i is the vector with i^{th} entry 1 and all others 0, L^+ is the psuedoinverse of the Laplacian of the graph G. Let $\widehat{R}_{(ij)}(t)$ represent the estimated resistance for the edge (i,j) after t iterations of the algorithm. The error plotted below is the L^1 norm of the difference normalized with the number of edges.

$$\epsilon(t) = \frac{1}{|E|} \sum_{(i,j) \in E} |R_{(ij)}(t) - \widehat{R}_{(ij)}(t)|. \quad (4)$$

The graph on which the algorithm was tested was a dense Erdos-Renyi Graph with 100 nodes and $p = 0.5$. Connectedness of the graph was ensured before simulation.

In Figure 1, we plot the error as stated in (4) v/s the iterations. Here, to generate the random spanning trees we have used Wilson's Algorithm. It can be seen that the error decreases very sharply initially and then the decay slows down. To note that the decay doesn't stop, observe the Figure 2 wherein we have plotted a close up of Figure 1. We can see that the error is still decreasing albeit slowly. Similar plots can be obtained for the convergence of estimated resistances using Aldous' algorithm for generating spanning

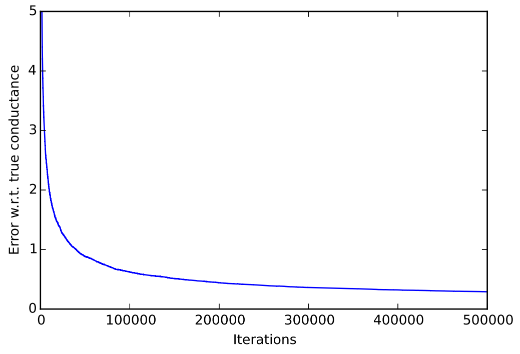

Fig. 1. Convergence when using Wilson's scheme

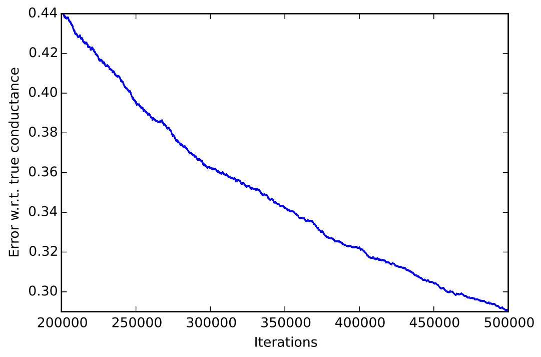

Fig. 2. Convergence when using Wilson's scheme (Close Up)

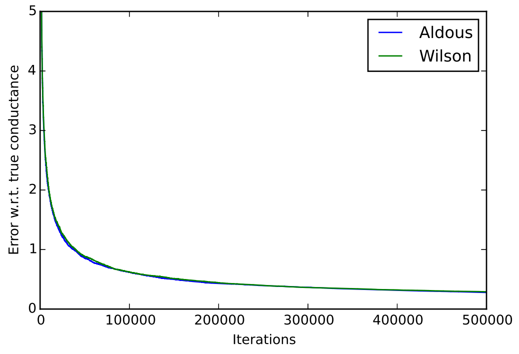

Fig. 3. Comparison when using Wilson's scheme and Aldous' scheme

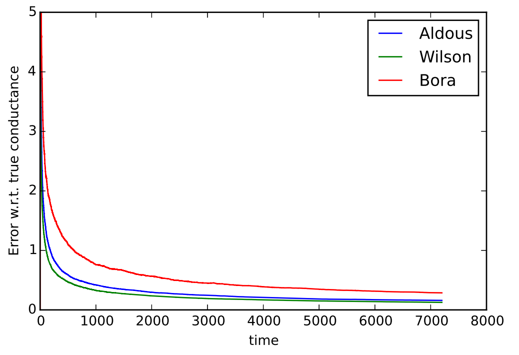

Fig. 4. Comparison between convergence of Aldous, Wilson and Bora

trees.

In Figure 3 we show the convergence of our algorithm using both Aldous' and Wilson's scheme on the same graph. Notice that the two plots are almost the same. This shows there is no change in the number of iterations required for $\epsilon-$convergence for some ϵ. What can create difference in speed is the time required per iteration. For the current graph, the time taken by Aldous' scheme per iteration was $0.032s$ whereas that by Wilson's was $0.024s$. Similar ratios between times per iteration were observed for other graphs.

In Figure 4 we have plotted the error as given in (4) in our scheme using both Aldous' and Wilson's algorithm along with Bora's scheme. It can be seen that our algorithm outperforms Bora's scheme irrespective of whether we use Aldous's or Wilson's algorithm to sample spanning trees. Among Aldous and Wilson, Wilson performs uniformly better than Aldous, but their errors approach each other with time. This is as expected since Aldous's scheme runs in expected time of the order of cover time of the graph whereas Wilson's scheme runs in the order of hitting time.

Finally, we point out that very recently Takanori et. al. [6] independently discovered the same algorithm.

IV. Ordinal Optimization

In optimization literature it is often important to compare two scenarios rather than estimating their exact performance. The question of whether we can compare two scenarios without estimating the output each of those result in is answered by Ordinal Optimization [7]. At its heart, ordinal optimization is based on two ideas: (i) Ordinal Comparison - Order is easier than Value, (ii) Goal Softening : Nothing but the best is very costly to find. The second idea means that getting close to the optimum is much easier than finding the exact optimum and it is often good enough to be close to the optimum. Book length treatments on Ordinal Optimization can be found in [4] and [8]. It has been shown in [5] that the convergence of an ordinal comparison is exponentially fast as compared to estimating actual values convergence of which is of the order $n^{-1/2}$ by the central limit theorem.

In graph sparsification, it has been shown that edges with high resistances retain the spectral properties of the graph. In this section, rather than trying to make the resistances converge, we want to look at how fast they get in the correct order, i.e. even if their values may not be same as the actual values, if when sorted, they form the same ordered set as the original, then it is good enough to choose the top-k% resistance edges for effective graph

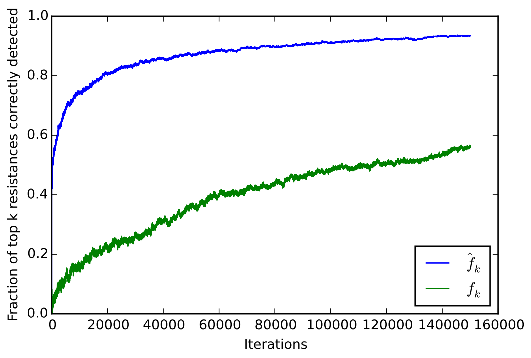

Fig. 5. Comparison between Ordinal and Cardinal $\epsilon = 0.01$

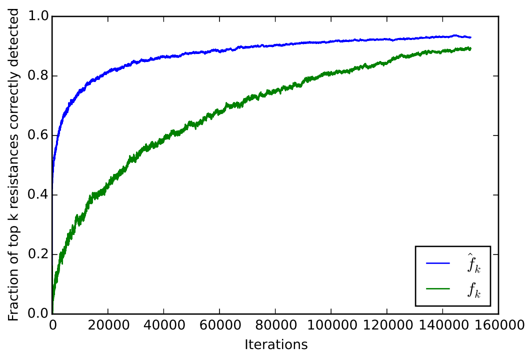

Fig. 6. Comparison between Ordinal and Cardinal $\epsilon = 0.02$

sparsification. This idea is parallel to ordinal optimization because we don't care about the values to which our algorithm converges, but just about whether the ordering of edges w.r.t. resistance converges to the correct order. For all the plots below, we define $\hat{f}_k(t)$ as the fraction of edges correctly estimated as the top k % values in t iterations. To calculate this, we find the set of top k % edges using the current estimate and find the number of elements in the set difference of this set and the set of true top k % edges.

In Figures 5,6 and 7 we plot $f_k(t)$ v/s iterations for k = 40%. Note that $f_k(t)$ in these figures represents what fraction of true top k % edges are within ϵ error (multiplicative) of their true values i.e. we find the fraction of top k % edges, resistances of which satisfy $|G_{ij} - \widehat{G}_{ij}| < \epsilon G_{ij}$. It is more logical to look at the multiplicative error because sensitivity to the additive error ($|G_{ij} - \widehat{G}_{ij}| < \epsilon$) can depend on the value of the resistance, which changes drastically from graph to graph. In multiplicative error, such dependence gets factored out. From the figures, we note that it is much better to start choosing the top resistance edges using the current estimate rather than waiting for the resistances to converge. As expected, as we increase the value of ϵ the value of $f_k(t)$ increases in the following plots. An important factor that these plots illustrate is that to what extent to the resistance estimates need to converge to the true values to start giving good ordinal results. As seen in Figure 7, the top k% resistance values reach the $\epsilon = 0.05$ very fast, but the ordinal results there are still not very good. This also means that the graph has very close resistance values, many of which are not more than 5% away from each other.

Next, in Figures 8,9 and 10 we plot $f(t)$ v/s iterations for k = 40%. Note that $f(t)$ in these figures represents what fraction of all edges are within ϵ error(multiplicative) of their true values i.e. we find the fraction of all edges, resistances of which satisfy $|G_{ij} - \widehat{G}_{ij}| < \epsilon G_{ij}$. Note that the difference here from figures 5,6 and 7 is that here $f(t)$ represents the fraction of all edges which have close values to the actual resistance values. This is important because for the identification of the top k% edges, it is important that not only those resistance estimates but all converge near to the true values. Similar to the observation from figures 5,6 and 7 increasing the value of ϵ increases $f(t)$. Also, the fact that ordinal convergence is faster than cardinal can be observed. In comparison to 5,6,7 these plots for $f(t)$ are just smoother because of more number of edges being considered, but the general trend remains same. This also illustrates that other than the corner edges with resistance 1, all edges converge to their true value at a similar rate.

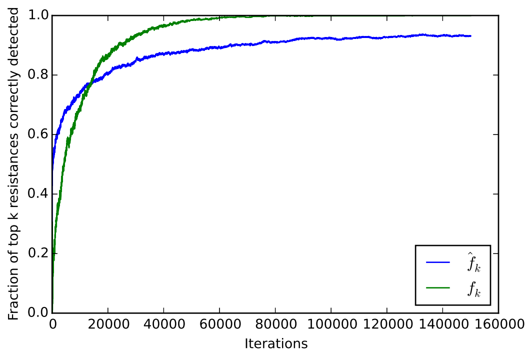

Fig. 7. Comparison between Ordinal and Cardinal $\epsilon = 0.05$

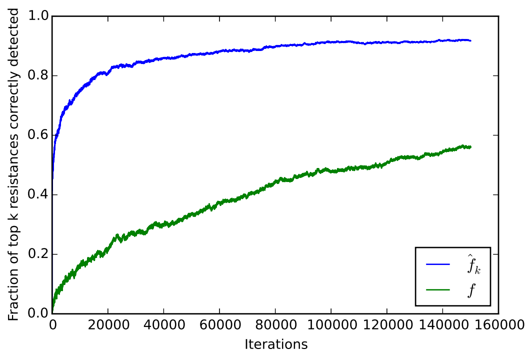

Fig. 8. Comparison between Ordinal and Cardinal $\epsilon = 0.01$

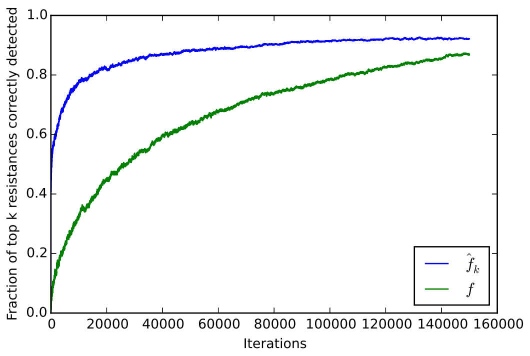

Fig. 9. Comparison between Ordinal and Cardinal $\epsilon = 0.02$

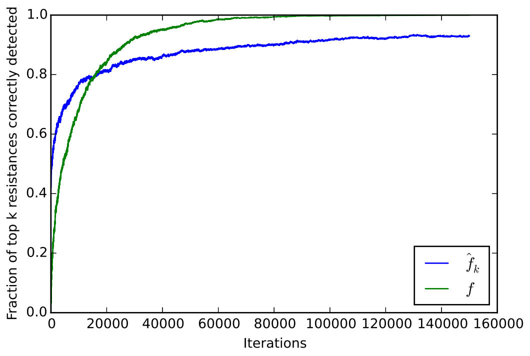

Fig. 10. Comparison between Ordinal and Cardinal $\epsilon = 0.05$

References

[1] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins", *J Comput System Sci 66* (2003), 671687.

[2] D. J. Aldous, "The random walk construction of uniform spanning trees and uniform labelled trees", *SIAM Journal of Discrete Mathematics*, 3(4):450-465,1990.

[3] A. Bora, V. S. Borkar, D. Garg and R. Sundaresan, "Edge Conductance Estimation Using MCMC", *54th Annual Allerton Conference* 2016, pp. 202 - 208.

[4] C. H. Chen, Q. S. Jia, L. H. and Lee,*Stochastic Simulation Optimization for Discrete Event Systems: Perturbation Analysis, Ordinal Optimization and Beyond*, World Scientific. (2013)

[5] L. -Y. Dai, "Convergence Properties of Ordinal Comparison in the Simulation of Discrete Event Dynamic Systems", *Journal of Optimization Theory & Applications,* Vol.91, No.2. pp.363-388, 1996.

[6] T. Hayashi, T. Akiba, and Y. Yoshida, "Efficient algorithms for spanning tree centrality", *25th International Joint Conference on Artificial Intelligence (IJCAI)*, 3733-3739, 2016.

[7] Y. C. Ho, R. Sreenivas and P. Vakili,"Ordinal Optimization of Discrete Event Dynamic Systems", *Journal of DEDS* 2(2), 61-88, (1992).

[8] Y. C. Ho, Q. C. Zhao, Q. S. Jia, *Ordinal Optimization: Soft Optimization for Hard Problems*, Springer, New York (2007).

[9] W. Hoeffding, "Probability inequalities for sums of bounded random variables". *Journal of the American Statistical Association*,58 (301): 1330,1963.

[10] A. Kalvit, V. Borkar, and N. Karamchandani, "MCMC approaches to rumor source inference using pairwise information", to appear in *VALUETOOLS* 2017

[11] L. Lovasz, Random walks on graphs: A survey, *Combinatorics, Paul Erdos is eighty*, vol. 2, no. 1, pp. 146, 1993.

[12] M. Lowe and F. Torres, On hitting times for a simple random walk on dense ErdsRnyi random graphs, *Statistics and Probability Letters* 89 (2014) 8188.

[13] D. Spielman and N. Srivastasa, "Graph sparsification by effective resistances", in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08)*, 2008, pp. 563568.

[14] D. A. Spielman and S. -H. Teng," Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems", in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 04)*, 2004, pp. 8190.

[15] S. -H. Teng, *Scalable algorithms for data and network analysis*, Foundations and Trends in Theoretical Computer Science, 12(1-2):1261, 2016.

[16] D. B. Wilson, "Generating random spanning trees more quickly than the cover time", *STOC*, pp. 296-303, 1996.