Abstract

We study the graph cut problem under local settings: algorithms have the constraint that its run time must be proportional to the set it outputs, instead of the entire graph. Quality of cut is measured under conductance metric. We experiment modified versions of Nibble algorithm in [1] and local pagerank algorithm in [2] on graphs that are designed to be “hard” on these algorithms.

Keywords: graph cut, local algorithms

1 Introduction

With the fast growth in number of social media users and size of community network graph, it becomes more and more important to answer the following type of query efficiently: given a node in a community network graph, find a cluster around given node. Here, we measure the quality of cluster by conductance metric, intuitively, it measures the ratio of out-of-cluster connectivity and internal connectivity of the cluster. We also restrict our attention to a class of algorithms we refer to local algorithms: run time of these algorithms must be proportional to the size of cluster it outputs instead of the size of entire graph. We remark local algorithms are crucial to today’s applications, especially on large social network graphs such as Facebook Graph [3] and Pinterest Graph [4], these graphs typically have billions of nodes and clustering query will be made for every node in a daily basis [5], so any algorithm whose run time is proportional to size of entire graph, even linear, is prohibitive to be deployed. In contrast, these graphs generally have small and relatively high quality clusters, which means local algorithms will be more than feasible in solving this kind of problems.
Local graph clustering algorithms have a relatively short but successful history. Starting from the groundbreaking work by Spielman and Teng [1], various algorithms such as local pagerank [2] and evolving set based algorithm [6] have been studied. All these algorithms exploit Lovasz-Simonovits Theorem [7], which connects conductance of a set and evolution of random walk, and inherits a Cheeger-type bound [8] in their output cluster. This essentially means that the cluster they output might have a $O(\sqrt{\lambda_2})$ from the best-possible cluster, where $\lambda_2$ is the second smallest eigenvalue of normalized Laplacian matrix of the graph. Recently, Wang et al. [9] have proposed a novel algorithm that is based on preflow-relabel algorithm due to Goldberg and Tarjan [10], and they claim that their algorithm won’t suffer from Cheeger’s bound. However, they don’t give a concrete proof that they can avoid such problem. Instead, unlike previous random walk based work, their analysis does not depend on Lovasz-Simonovits Theorem. We remark that, though all of above algorithms have great theoretical interests and practical usage, and local pagerank algorithm has been deployed in various applications, there’s almost no empirical evidence in showing that random walk based algorithms suffer from Cheeger’s bound, and Wang et al.’s algorithm does not have this problem.

We are interested in the following two problems: 1). Does random walk based algorithms really have Cheeger’s bound embedded in them, and can we verify this through experiments? 2). Can we confirm that Wang et al.’s algorithm really solve the problem of Cheeger’s bound least through experimental evidence? Especially, we setup the experiment on graphs which are supposed to have a poor performance on algorithms rely on Cheeger’s bound and verify modified versions of Nibble [1], local pagerank [2] and Wang et al.’s flow algorithm [9] on it. This report will focus on experiments of random walk based algorithms, and present some novel algorithms on simulating random walk on these graphs. For flow based algorithm by Wang et al., we refer readers to report by Andy Yang. For more analysis on graph construction and performance evaluation of random walk based algorithms, we refer readers to report by Tian Luo.

2 Problem Setup & Notations

2.1 Conductance

Let $G = (V, E)$ be an undirected graph, let $d(v)$ denote the degree of $v \in V$. Suppose $S \subseteq V$, define $\text{vol}(S) := \sum_{v \in S} d(v)$, and $E(S, T) = \{\{u, v\} \in E : u \in S, v \in T\}$, the conductance of set $S$ is defined as

$$\Phi(S) := \frac{|E(S, \overline{S})|}{\min\{\text{vol}(S), \text{vol}(\overline{S})\}}$$

In case $S = G$, we define $\Phi(G) = 1$. And the conductance of $G$ is defined as

$$\Phi_G := \min_{S \subseteq V} \Phi(S)$$

The local graph clustering problem is defined as follows: given $v \in V$, find a subset $C \subseteq V$ such that $v \in C$, and

$$\Phi(C) = \min_{S \subseteq V, v \in S} \Phi(S)$$

2.2 Graph Laplacian and Cheeger’s Inequality

The graph Laplacian matrix, $L$ is defined as $L := D - A$, where $D$ is the degree matrix of $G$, which is a diagonal matrix with each entry corresponds to degree of each vertex, and $A$ is the adjacency matrix of $G$. The normalized Laplacian matrix, $\tilde{L}$, is defined as $\tilde{L} = D^{-1/2}LD^{-1/2}$, let $\lambda_2$ denote the
second smallest eigenvalue of $\tilde{L}$. We remark that $0 < \lambda_2 \leq 1$. The Cheeger’s inequality [8] connects $\Phi_G$ to $\lambda_2$, specifically, it says

$$\frac{\lambda_2}{2} \leq \Phi_G \leq \sqrt{2\lambda_2}$$

Notice lower bound and upper bound differ by a $O(\sqrt{\lambda_2})$ factor, in Cheeger’s proof of the upper bound [8], he presented an algorithm to find a cut of conductance $\sqrt{2\lambda_2}$, which means there exists a class of algorithms that finds cut whose conductance is $O(\sqrt{\lambda_2})$ away from best cut. Theoretically, Nibble algorithm [1], local pagerank [2] and evolving set [6] all have such problem.

### 2.3 Cartesian Product of Graphs

Let $G = (V_G, E_G), H = (V_H, E_H)$, we define the Cartesian product of $G$ and $H$, denoted by $G \square H$, as follows: let $A_G, A_H$ be adjacency matrices of $G$ and $H$, then

$$A_G \square H := A_G \oplus A_H = A_G \otimes I_H + I_G \otimes A_H$$

Here $\oplus$ denotes the Kronecker sum, $\otimes$ denotes the Kronecker product and $I_H, I_G$ denote the identity matrices of size $|V_H|, |V_G|$. We remark several properties of $L_{G \square H}$ and $L_G, L_H$. Let $\lambda(L)$ denote set of eigenvalues of $L$, then

$$\lambda(L_{G \square H}) = \{\lambda_i + \mu_j : \lambda_i \in \lambda(L_G), \mu_j \in \lambda(L_H)\}$$

In words, eigenvalues of $L_{G \square H}$ are direct sums of eigenvalues of $L_G$ and $L_H$, and eigenvectors have similar structure: suppose $x_i$ is an eigenvector of $L_G$ and $y_j$ is an eigenvector of $L_H$, then one of eigenvectors of $L_{G \square H}$ is computed as follows: 1) compute $x_i y_j^T$, i.e., outer product of two vectors, 2) flatten the matrix into a single vector. In words, eigenvectors of $L_{G \square H}$ can be viewed as direct products of eigenvectors of $L_G, L_H$. We further remind reader that since Laplacian matrix has rank $n - 1$, the smallest eigenvalue is 0, which means

$$\lambda_2(L_{G \square H}) = \min \{\lambda_2(L_G), \lambda_2(L_H)\}$$

### 2.4 Problem Setup

For the sake of simplicity, let $N \in \mathbb{N}_+$ such that $N = n^3$ for some $n \in \mathbb{N}_+$, consider a balanced binary tree $T$ on $O\left(N^{2/3}\right)$ nodes and a path $P$ on $O\left(N^{1/3}\right)$ nodes, it’s worth noting that $\lambda_2(L_T) = \Theta\left(\frac{1}{N^{2/3}}\right)$ and $\lambda_2(L_P) = \Theta\left(\frac{1}{N^{2/3}}\right)$, so they are at the same order and differ by only a constant. Specifically, $\lambda_2(P) \leq \frac{12}{N^{2/3}}$ and $\lambda_2(T) \geq \frac{0.5}{N^{2/3}}$, so we can set $|V_P| = \lceil 4\sqrt{3} \rceil N^{1/3}$ and $|V_T| = N^{2/3}$, so that
\[ \lambda_2(L_T \square P) = \lambda_2(L_P). \] Intuitively, one can think of the graph as balanced tree lie from top to bottom, and each two consecutive trees are connected with path. Such eigenstructhure will cause Cheeger-type to find a cut that cuts all paths between the two trees at center, which cuts \( N^{2/3} \) edges, however, a much better cut will be cutting one edge near root for each tree, which only cuts \( O(N^{1/3}) \) edges. We expect algorithms like Nibble and local pagerank to fail on this example — i.e., they will find a cluster that cuts the graph horizontally, not vertically. On the other hand, we expect algorithm by Wang et al. \cite{9} to perform well on this example. We refer to this particular \( T \square P \) graph as tree-cross-line graph.

3 Main Results

We present two experiments using Nibble and local pagerank on tree-cross-line graph. Specifically, we implement modified versions of these algorithms, since our focus is to evaluate the quality of cut they output instead of getting a fast algorithm. We perform both experiment on a tree-cross-line graph with \( N = 15625 \), with tree size 625 and path size 25, notice the path size is \( \lceil 4\sqrt{3} \rceil \) times smaller than the size that will make \( \lambda_2(L_P) \) smaller (this is due to computation cost for matrix vector product or linear system solve for \( N \times N \) matrices), so we expect these algorithms can find a decently good cut on our graph. However, the experiments show that the cut they produce is more of a horizontal cut, and this gives evidence that these random walk based algorithms are subject to Cheeger’s bound.

3.1 Experiment Setup

We implement modified versions of both Nibble and local pagerank.

3.1.1 Nibble as Random Walk

The vanilla Nibble algorithm presented by Spielman and Teng \cite{1} is as follows:

\begin{itemize}
  \item INPUT: Graph \( G \), starting node \( u \), epsilon
  \item let \( p \) be an indicator vector on starting node \( u \);
  \item let \( W \) be the random walk transition matrix;
  \item for \( i = 1, 2, \ldots, \text{max iterations} \) do:
    \begin{itemize}
      \item \( p = Wp; \) // perform walk
      \item let \( u \) be the vector such that:
        \begin{itemize}
          \item \( u[j] = p[j] / d(j), \) where \( d(j) \) is the degree of node \( j \);
          \item set \( u[j] = 0 \) if \( u[j] < \) epsilon;
        \end{itemize}
      \item sort \( u \) in descending order;
      \item examine all level cuts formed by values in sorted \( u \),
      \item record the best one;
      \item output the best level cut;
    \end{itemize}
\end{itemize}

Notice the step that removes probability on entry where \( \frac{p[j]}{d(j)} \) is too small only affects the run time of algorithm but not the quality of cut, in fact, without removing those small entries, the cut it outputs will have a better quality guarantee. Our modified version won’t remove small probability entries in \( u \).
3.1.2 Local PageRank as Solving Laplacian System

The vanilla version of local pagerank algorithm presented by Anderson, Chung and Lang [2] is even simpler than Nibble:

**INPUT:** Graph G, starting node u, reset probability a
let p be a vector that is supposed to be the solution to a linear system;
let W be random walk transition matrix;
let m be an indicator vector on starting node u;
solve system \([I - (1 - a)W]p = am\);
form vector u where \(u[j] = p[j] / d(j)\);
sort u in descending order;
examine all level cuts formed by values in sorted u;
output the best level cut;

We will then show can be done by one Laplacian solve.

**Proposition 3.1.** The vector \(u\) can be computed directly via a Laplacian solve, specifically, \(u\) is the solution to the system
\[
(\beta D + L)u = \beta m
\]
Where we assume \(W = A^T D^{-1}\) to be the standard random walk transition matrix, and \(\beta = \frac{a}{1-a}\).

We defer the proof to appendix section.

3.1.3 Experiment Environment & Parameter Setup

We conduct all experiments under Python 3.7, with numerical computation package **numpy** and graph package **networkx**.

Following our proceeding discussion, we implement Nibble by performing matrix vector product of \(W\) and \(p\), this is achieved via **numpy**’s matrix multiplication, i.e., @ operation. For local pagerank, we solve the Laplacian system
\[
(\beta D + L)u = \beta m
\]
Directly using **np.linalg.solve**.

For parameter selection, we choose \(N = 15625\), which is \(25^3\). We choose tree size to be \(625 = 25^2\), and path size to be 25, it is worth noting that if we want to ensure \(\lambda_2(P) < \lambda_2(T)\), we need path size to be 7 times larger. However, due to space constraint on the size of the matrix, we have to relax the condition by making the path smaller. This will make \(\lambda_2(T) < \lambda_2(P)\), but the difference in magnitude is less than \(10^{-3}\).

3.2 Experiment Result & Analysis

In this section, we also include surprises, since the general results are quite surprise and a little bit of out of expectation. We first give the benchmark of our experiment:

<table>
<thead>
<tr>
<th></th>
<th>number of cut edges</th>
<th>conductance</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertical cut</td>
<td>25</td>
<td>0.0008</td>
</tr>
<tr>
<td>horizontal cut</td>
<td>625</td>
<td>0.0204</td>
</tr>
</tbody>
</table>

Table 1: Benchmarks for vertical and horizontal cuts
We remark that a balanced binary tree of size 625 has height 10. For both Nibble and local pagerank, we start from one of the leaves at the top tree.

For Nibble, we simulate random walk for 100 iterations, the resulting cut has size 961 edges and conductance 0.03140. The shape of the cut is drawn as follows: Here $S$ refers the cut that Nibble outputs, we draw it in a square fashion, where the vertical sides can be viewed as paths and horizontal sides can be viewed as trees. Intuitively, one can view it as we lie trees horizontally, from top to bottom on the square, so the horizontal cut should correspond to a horizontal line on the square, and a vertical cut should correspond to a vertical line on the square. As we can see from the figure, the cut is more of a horizontal cut, but it also cuts many edges on trees much lower on the path, so the cut looks like a diagonal cross. It also contains fewer and fewer nodes as it goes down the path, which is a sign that the cut it produces resembling the horizontal Cheeger’s cut. More specifically, it contains the first 9 trees entirely, and most part of next 3 trees, and subsequent lower trees have fewer and fewer parts being in the cut. To better illustrate the idea, we draw the first 4 levels near root of the 17th tree:

![Figure 2: Shape of cut produced by Nibble](image)

We perform the experiment of local pagerank, starting at the same node as Nibble. There is an extra reset probability $a$, and we set $a = 0.2$. The cut it produced cuts 989 edges and conductance 0.0323. The shape of the cut is almost the same as in Figure 2. We also draw the first 4 levels of the 17th tree:

![Figure 3: First 4 levels of 17th tree, orange nodes are in cut S](image)
Figure 4: First 4 levels of 17th tree, orange nodes are in cut $S$

<table>
<thead>
<tr>
<th></th>
<th>number of cut edges</th>
<th>conductance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nibble</td>
<td>961</td>
<td>0.0314</td>
</tr>
<tr>
<td>local pagerank</td>
<td>989</td>
<td>0.0323</td>
</tr>
</tbody>
</table>

Table 2: Experiment summary of both algorithms

Notice both cuts produced by Nibble and local pagerank are worse than Cheeger’s cut, we have several conjectures about this result without formal proofs.

Conjecture 1 Cheeger’s cut uses only a single eigenvalue and eigenvector, so the cut structure is much simpler, on the other hand, both Nibble and local pagerank are variations of random walk, which means the vector we are working is a linear combination of multiple eigenvectors. Our previously thought was other eigenvectors will help to mitigate the problem caused by Cheeger’s cut, but through experiments, it seems the eigenstructure is not very easy to understand.

Conjecture 2 Notice the path size is a constant factor smaller than the desired size, which means due to Cheeger, the cut should look like a vertical cut, this might be a potential reason why the cut contains some small portion of lower trees.

4 Conclusions & Open Questions

We implemented modified versions of Nibble and local pagerank algorithms and experimented them on tree-cross-line graph, which is designed to be hard instance for these random walk based algorithms. Through experiments, we discovered that the cut these algorithms detected were not the best vertical cut nor the standard Cheeger-type cut, but a hybrid of both. We also have open questions for further research and work:

1. Our modified versions simplify the experiments, and by [1] and [2], it should only affect run time. However, it is possible that the cut quality is also affected by their remove of small entries, so one of the future directions is to implement vanilla version of these algorithms. Also, the path size in our experiment is a constant smaller than the desired size, so a more dedicated experiment on a “correct” size path will be performed.
2. Ultimately, tree-cross-line graph is a Cartesian product, so does there exist a class of algorithms that also has such product structure? For example, for random walk, can we simulate walks independently on tree and path, then combine the results of two walks in some way. We have been working simulating the most basic random walk on tree-cross-line graph in a product fashion, and hopefully can have an algorithm with a proof in this setting.

3. According to the experiment conducted by one of the coauthors Andy Yang, flow based algorithm by Wang et al. [9] does have a good performance on tree-cross-line graph, i.e., it can detect the best vertical cut on the graph. Can we work out a formal proof on why it works? Can we leverage more on the flow dynamics to develop more interesting algorithms on this problem?

Acknowledgment

We appreciate the help from Di Wang and Satish Rao on giving more clarifications and intuitions on their algorithm [9]. This work is a class project of the sequence of courses 15-300, Research and Innovation in Computer Science and 15-400, Research Practicum in Computer Science. We appreciate the advise of Bogdan Vasilescu and Jonathan Aldrich during the span of the project.

References


A Proof of Proposition 3.1

Proposition A.1. The vector \( u \) can be computed directly via a Laplacian solve, specifically, \( u \) is the solution to the system

\[
(\beta D + L) u = \beta m
\]

Where we assume \( W = A^T D^{-1} \) to be the standard random walk transition matrix, and \( \beta = \frac{a}{1-a} \).

Proof. Recall the system we originally wish to solve can be written as follows:

\[
(I - (1 - a)W) p = am
\]

Notice the vector \( u \) can be formed by a matrix-vector product: \( u = D^{-1} p \), so substitute \( u \) and \( W \) in, we get

\[
(I - (1 - a)A^T D^{-1}) Du = am
\]

\[
(D - (1 - a)A) u = am \quad \text{A is symmetric}
\]

\[
(aD + (1 - a)D - (1 - a)A) u = am
\]

\[
(aD + (1 - a)L) u = am
\]

\[
(\beta D + L) u = \beta m
\]

To see this reduces to a Laplacian solve, consider the following weighted graph \( G' \): we add a ground node to \( G \), call it \( v \), and wire all nodes in \( G \) to \( v \), with each edge weight be \( \beta d(i) \) for edge \( \{i, v\} \), and we can augment \( m \) by adding a -1 on its entry corresponding to \( v \). This reduces the problem to solve a standard Laplacian system:

\[
L'u' = m'
\]

Then we can post-process vector \( u' \) by first subtracting multiples of \( 1 \), the all-one vector to make entry \( v \) being 0, then multiplying it by \( \beta \). This gives our desired vector \( u \). \( \square \)