Finding Cut on Spectral-Hard Graphs using Local Algorithms

Lichen Zhang, collaborated with Andy Yang, Tian Luo, advised by Professor Gary Miller

Carnegie Mellon University

Carnegie Mellon University
Computer Science Department

Introduction & Background

As the popularity of social media and community network grows, an interesting question arises: given a fixed node on a graph, how can we find a "community" or "cluster" around that node, such that the run time of the algorithm is proportional to the size of output set instead of the whole graph? Start from the groundbreaking work by Spielman and Teng [4], several algorithms with run time nearly linear with respect to the output set, exploiting some variants of random walks have been proposed, including the widely-used pagerank algorithm [1]. However, all of these algorithms use Lovasz-Simonovits Theorem [3] to prove their validity, which has a Cheeger type bound [2] on the quality of cut they found. Recently, Wang et al. [5] has proposed a flow-based algorithm that claims to overcome the Cheeger's barrier. Our problems are in two-folds: 1). Wang et al.'s algorithm does not have a direct evidence of why they break Cheeger's bound, they just don't use the technology that causes the problem. Can we verify, at least through experiments, that their algorithm really solves the problem? 2). Are the other random-walk based algorithms really subject to Cheeger's bound, or is that just a consequence of the tools they used?

Terminology & Notations

Let $G=(V,E)$ be a graph, and $S\subseteq V$, the *volume* of S is defined as $\text{vol}(S)=\sum_{v\in S}\text{degree}(v)$, and *conductance* of S is defined as

$$\Phi(S)=\frac{E(S,V\setminus S)}{\min(\text{vol}(S),\text{vol}(V\setminus S))}$$

where $E(S,V\setminus S)$ denote number of edges with one endpoint in S and the other in $V\setminus S$. Finally, the *conductance* of G is defined as $\Phi_G=\min_{S\subseteq V}\Phi(S)$.

The *graph Laplacian matrix*, L, is defined as $L=D-A$, where D is the degree matrix over vertices, and A is the adjacency matrix. The *normalized Laplacian*, $\tilde{L}$, is $\tilde{L}=I-D^{-1/2}AD^{-1/2}=D^{-1/2}LD^{-1/2}$. The *Cheeger* inequality introduced in [2] gives a concrete bound on the second smallest eigenvalue of $\tilde{L}$, λ_2:

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

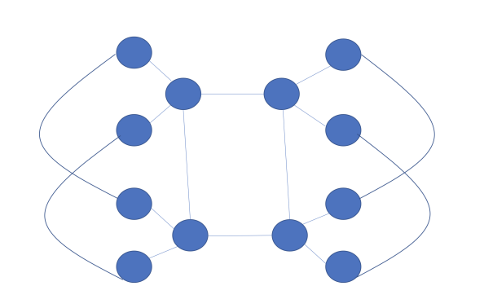
Together with this inequality, [2] also introduces a canonical algorithm that finds a cut of quality $O\left(\sqrt{\lambda_2}\right)$, however, this might differ Φ_G by a factor of $\sqrt{\lambda_2}$. We call algorithms that has such a problem *subject to Cheeger's bound*.
The *Cartesian product* of two graphs G,H, denoted by $G\otimes H$can be defined in terms adjacency matrix: suppose $P=G\otimes H$, then

$$A_P=A_G\oplus A_H=A_G\otimes I_H+I_G\otimes A_H$$

where $\oplus$ is the Kronecker sum and $\otimes$ is the Kronecker product. We remark that eigenvalues of L_P are direct sums of L_G,L_H, i.e., if $\lambda(L_G)=\{\lambda_1,\lambda_2,\dots,\lambda_n\},\lambda(L_H)=\{\mu_1,\mu_2,\dots,\mu_m\}$, then $\lambda(L_P)=\{\lambda_i+\mu_j:i\in[n],j\in[m]\}$.

Main Problem

Let T_n be a balanced binary tree of size n and P_m be a path of length m, consider the product graph $G=T_n\otimes P_m$, where $m=N^{1/3},n=N^{2/3}$, and $\lambda_2(P_m)<\lambda_2(T_n)$



If we use standard Cheeger algorithm on this graph, it will give rise to a horizontal cut that cuts all paths between two trees at center, yielding a cut of size $O\left(\frac{N^{2/3}}{\text{vol}(G)}\right)$, on the other hand, if we cut vertically, i.e., cut the middle edge of each tree, this gives rise to a cut of size $O\left(\frac{N^{1/3}}{\text{vol}(G)}\right)$, therefore, this is a bad example for algorithms subject to Cheeger's bound. We call this graph *tree-cross-line graph*.

Results

Pick $N=15625$, we experiment a variation of `Nibble` algorithm [4] and local pagerank algorithm [1] on tree-cross-line graph on N vertices. As a byproduct, we also provide a fast combinatorial algorithm that simulates random walk process on tree-cross-line graph. Standard random walk involves doing matrix-vector product at each step, where in our case, matrix is of size $N\times N$, so we need to pay $O(t_N)$ time, where t_N is the run time of a matrix-vector product algorithm of size $N\times N$. When N gets larger, this can be prohibitive to run. In contrast, our proposed algorithm has its run time depends on the tree size, i.e., $N^{2/3}$.

Theorem 1. *There exists an algorithm that performs random walk on tree-cross-line graph for k steps in time $O\left(kt_{N^{2/3}}+N\right)$.*

Proof Sketch. At each time stamp, we just need to keep track of constant number of vectors associated with the tree instead of $N^{1/3}$ of them. Let W be the transition matrix, and random walk starts at a tree that has two neighbor trees. Let v_i^t denote the vector of at time t, for the tree that *first gets mass* at time i, and let $*$ denote the Hadamard product, then

$$v_0^{t+1}=W\left(\ell*v_0^t\right)+(\mathbf{1}-\ell)*v_1^t \qquad (1)$$

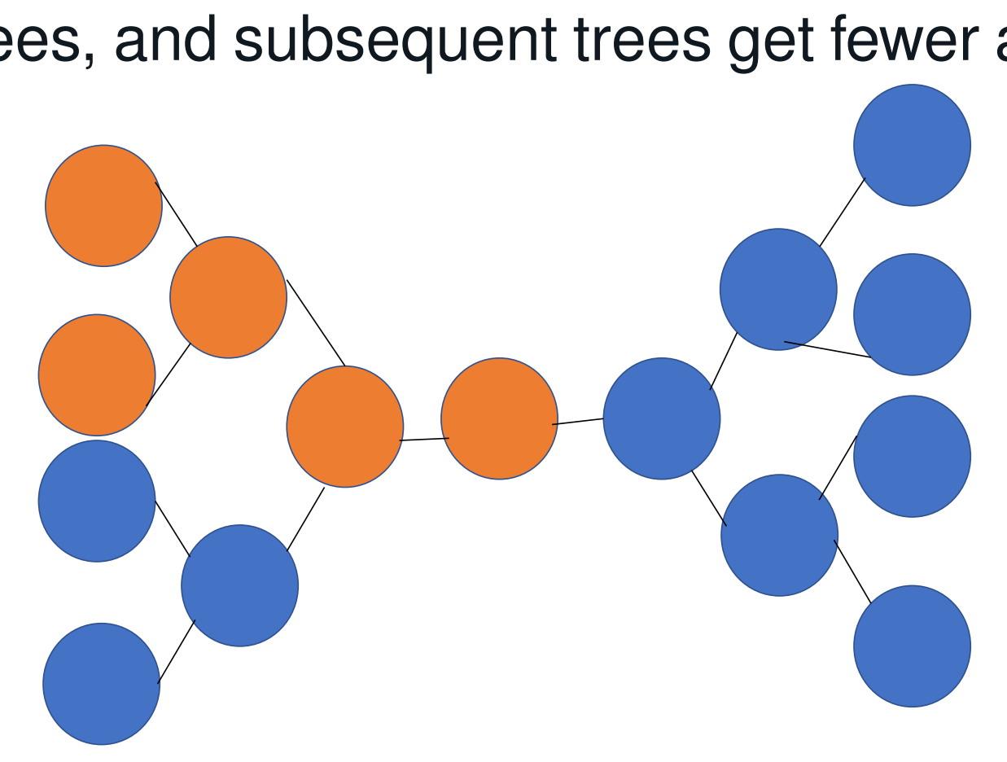
$$v_i^{t+1}=W\left(\ell*v_i^t\right)+\frac{1}{2}(\mathbf{1}-\ell)*\left(v_{i-1}^t+v_{i+1}^t\right)\quad i\neq 0,\text{ and it's not the top/bottom tree} \qquad (2)$$

$$v_{\text{top}}^{t+1}=W\left(\ell*v_{\text{top}}^{t+1}\right)+\frac{1}{2}(\mathbf{1}-\ell)*v_i^t\quad v_i\text{ is the neighbor tree of top} \qquad (3)$$

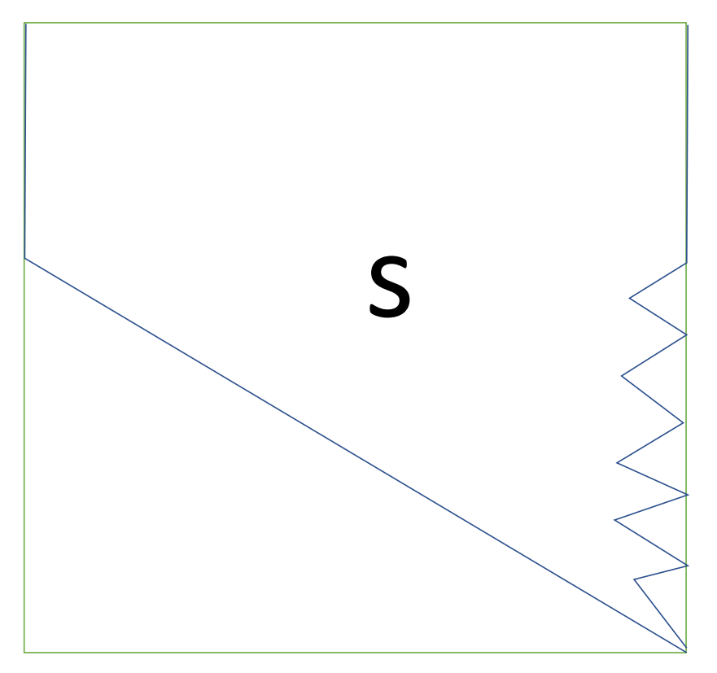
$$v_i^i=\frac{1}{2}(\mathbf{1}-\ell)*v_{i-1}^{i-1} \qquad (4)$$

Here v_{top} denote either the tree at top or at bottom, i.e., one of them must be the lastest to get mass, and they only have one neighbor tree, and ℓ is a normalization vector. Combine (2) and (4), we can rewrite the two variables recurrence into a single variable recurrence of t, and to compute vector at time $t+1$, we at most need the vector at time t and $t-1$. Also, different trees are just scaled version of their parents. Thus, we only need to keep track of constant number of vectors of size $N^{2/3}$. $\square$

Next, we present experimental results of a modified `Nibble` and local pagerank algorithm. For `Nibble`, we do not remove small probabilities but keeping them, we remark this does not affect the quality of the cut but only run time. This simplification also extends to local pagerank, where in [1], they approximate pagerank vector in order to boost run time, where we abort the approximation.
As a benchmark, the best vertical cut in this particular graph is $\frac{25}{30600}\approx 0.0008$, and the horizontal cut is of size $\frac{625}{30600}\approx 0.0204$.
For `Nibble`, we start from one of the leaves of the first tree and run it for 100 iterations, the resulting cut has conductance 0.03140, and the cut is more of a horizontal cut, it contains the first 9 trees entirely and most part of next 3 trees, and subsequent trees get fewer and fewer,



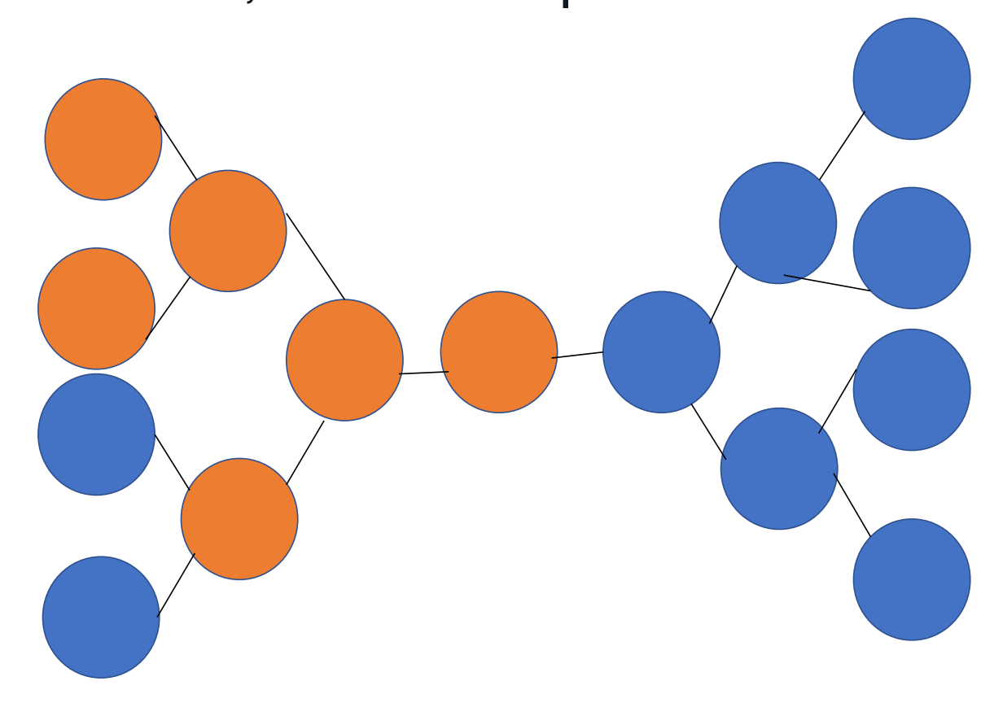
following is a figure of a tree at level 17:
Here orange nodes are in cut S, while blue nodes are not.



Induced cut has the following shape:

Results(cont.)

We run modified local pagerank algorithm starting at the same vertex, with reset probability $\alpha=0.2$, and the resulting cut has conductance 0.0323, we still present the first 4 layers of the tree at level 17:



We remark the resulting cut has very similar shape as `Nibble`, which is consistent with the natural of local pagerank algorithm, it is a faster variation of random walks.

Remarks & Open questions

According to our experiments, both `Nibble` and local pagerank are consistent with their claims — the algorithms are subject to Cheeger's bound. However, our implementation does not incorporate the "round down" feature in both of the original algorithms. In both [4] and [1], their proofs of cut quality is invariant of round down, it's just an improvement of run time, but will this make a change to practical run of the algorithm?
The next question is since we are working with Cartesian product graph, is there a notion of *product algorithm* that we can exploit? Take random walk as an example, a product algorithm of random walk might be, we perform walk separately on two graphs and combine results of both in some way to produce the result on product graph. If this is the case, then we can definitely do experiments on much larger graphs.
The last question is the flow-based algorithm due to Wang et al. [5] seems to produce satisfactory result of finding good cut on tree-cross-line graph, due to the experiment done by one of our collaborators Andy Yang. One concern is his experiment graph is not large enough, so result might be misleading. On the other hand, if the flow algorithm **does** work, how can we prove that? What can we leverage from their design and analysis?

Acknowledgements

This work is a collaboration with Andy Yang, Tian Luo and Professor Gary Miller. I sincerely appreciate the help from all of my collaborators. For experimental results related to Wang et al.'s algorithm [5], we refer audience to Andy's work, and for more theoretical parts of our work, please check out Tian's work.

References

[1] Reid Andersen, Fan Chung, and Kevin Lang. "Local Graph Partitioning using PageRank Vectors". In: Oct. 2006, pp. 475–486. DOI: 10.1109/FOCS.2006.44.

[2] Jeff Cheeger. "A lower bound for the smallest eigenvalue of the Laplacian". English (US). In: *Proceedings of the Princeton conference in honor of Professor S. Bochner*. 1969, pp. 195–199.

[3] László Lovász and Miklós Simonovits. "Random Walks in a Convex Body and an Improved Volume Algorithm". In: *Random Struct. Algorithms* 4 (1993), pp. 359–412.

[4] Daniel A. Spielman and Shang-Hua Teng. "A Local Clustering Algorithm for Massive Graphs and its Application to Nearly-Linear Time Graph Partitioning". In: *arXiv e-prints*, arXiv:0809.3232 (Sept. 2008), arXiv:0809.3232. arXiv: 0809.3232 [cs.DS].

[5] Di Wang et al. "Capacity Releasing Diffusion for Speed and Locality". In: *arXiv e-prints*, arXiv:1706.05826 (June 2017), arXiv:1706.05826. arXiv: 1706.05826 [cs.DS].