Parallel Spectral Graph Partitioning

Maxim Naumov and Timothy Moon
NVIDIA, 2701 San Tomas Expressway, Santa Clara, CA 95050

Abstract

In this paper we develop a novel parallel spectral partitioning method that takes advantage of an efficient implementation of a preconditioned eigenvalue solver and a k-means algorithm on the GPU. We showcase the performance of our novel scheme against standard spectral techniques. Also, we use it to compare the ratio and normalized cut cost functions often used to measure the quality of graph partitioning. Finally, we show that the behavior of our spectral scheme and popular multi-level schemes is starkly different for two classes of problems: (i) social network graphs that often have power law-like distribution of edges per node and (ii) meshes arising from discretization of partial differential equations. Although in our numerical experiments the multi-level schemes are almost always faster, we show that our spectral scheme can achieve a significantly higher quality of partitioning for the former class of problems.

1 Introduction

The graph partitioning problem arises in many areas. For example, it can be used to minimize communication and perform load balancing in parallel computing [32, 35]. It can also improve and facilitate hierarchical design, layout and testing of electronic circuits [7, 15, 44]. Further, it can be used to identify clusters of elements which represent different communities in social networks [26], among many other applications.

NVIDIA Technical Report NVR-2016-001, March 2016.

This research was developed, in part, with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

There are many strategies to perform graph partitioning, starting from local heuristics, such as Kernighan-Lin [23] and Fiduccia-Mattheyses [12], to global spectral and multi-level techniques [6, 22].

In this paper we focus on the global spectral graph partitioning technique. It constructs the Laplacian matrix and uses its smallest eigenpairs to perform the partitioning. The relationship between the eigenpairs of a Laplacian matrix and the connectivity of a graph was first noted by Donath and Hoffman [11] and Fiedler [13]. The technique was later improved and used in many different research areas, with some examples given in the first paragraph.

A comprehensive scientific literature review of spectral graph partitioning is given in [8, 28]. We avoid repeating it and rather focus on reviewing the mathematical theory behind the Laplacian matrix, spectral graph bisection and spectral partitioning into an arbitrary number of partitions. We use this theory to setup the eigenvalue problem and show how to use the k-means algorithm to transform a continuous solution into a discrete one. The concise review of this material unifies many different approaches used in mathematics, computer science and engineering communities for it.

Also, we review different approaches available for the solution of the eigenvalue problem that arises in the spectral scheme. In particular, we propose the use of a preconditioned eigenvalue solver [25, 24] that results in a significantly better performance and quality of the graph partitioning. We take great care in its implementation to preserve parallelism and maintain numerical stability of the algorithm. We point out that Laplacian matrix is singular and therefore it is relatively easy to obtain incorrect results when working with it.

Moreover, we use our spectral scheme to compare two different cost functions often used to measure the quality of graph partitioning. We empirically show that in our numerical experiments normalized cut cost function [36] often requires less iterations to convergence than the ratio cut [44].

Finally, we compare our spectral scheme with other the state-of-the-art implementations of spectral [17] and multi-level techniques [21]. In our numerical experiments we point to trends that indicate that the behavior of spectral and multi-level schemes is starkly different for two classes of problems: (i) social network graphs that often have power law-like distribution of edges per node and (ii) meshes arising from discretization of partial differential equations. Even though multi-level schemes are considered part of global techniques, they often rely on local information to construct a graph hierarchy that is used to partition the graph. Therefore, we conjecture that this different behavior is driven by the type of information used by the schemes to partition the graph.

Let us now formulate the graph partitioning problem.

2 Graph Partitioning

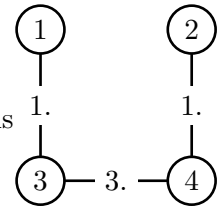
Let a graph $G = (V, E)$ be defined by its vertex V and edge E sets. The vertex set $V = \{1, ..., n\}$ represents n nodes in a graph, with each node identified by a unique integer number $i \in V$. The edge set $E = \{(i_1, j_1, w_1), ..., (i_e, j_e, w_e)\}$ represents e weighted edges in a graph, with each directed edge from node i to j of positive weight $w > 0 \in \mathbb{R}$ identified by a tuple $(i, j, w) \in E$.

Also, let the weighted adjacency matrix $A = [a_{i,j}]$ of a graph $G = (V, E)$ be defined through its elements

$$a_{i,j} = \begin{cases} w & \text{if } (i, j, w) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Let us assume that if $(i, j, w) \in E$ then $(j, i, w) \in E$, in other words, the matrix A is symmetric. If it is not, we can always work with $\tilde{G}$ induced by $A + A^T$. An example of a graph G and weighted adjacency matrix A is shown below.

$$A = \begin{pmatrix} 0.0 & & 1.0 & \\ & 0.0 & & 1.0 \\ 1.0 & & 0.0 & 3.0 \\ & 1.0 & 3.0 & 0.0 \end{pmatrix}, G = (V, E) \text{ is}$$



$V = \{1, 2, 3, 4\}$
$E = \{(1, 3, 1.0),$
$(2, 4, 1.0), (3, 4, 3.0)\}$

Let a set of vertices $S \subseteq V$ then its boundary is a set of edges $\partial(S) \subset E$ such that only one end point vertex of each edge is in S, in other words,

$$\partial(S) = \{(i, j, w) : i \in S \wedge j \notin S\} \tag{2}$$

and let a cut $C = (S, T)$ of a graph be a partition of vertices V into two disjoint sets S and $T \subseteq V$.

In graph partitioning we are often interested in finding a minimum balanced cut $C = (S, \bar{S})$ of a graph $G = (V, E)$ in the sense that S is such that either

$$\rho(S) = \min_S \frac{\text{vol}(\partial(S))}{|S||\bar{S}|} \tag{3}$$

or

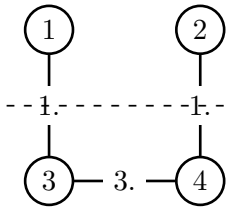
$$\eta(S) = \min_S \frac{\text{vol}(\partial(S))}{\text{vol}(S)\text{vol}(\bar{S})} \tag{4}$$

where $\bar{S}$ is the complement of set S with respect to V, $|.|$ denotes the cardinality

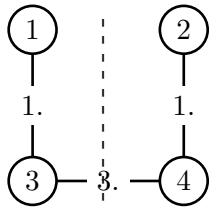
(number of elements) of a set, and

$$\begin{aligned} \mathrm{vol}(\partial(S)) &= \sum_{(i,j,w)\in\partial(S)} w \\ \mathrm{vol}(S) &= \sum_{i\in S \,\wedge\, (i,j,w)\in E} w \end{aligned} \tag{5}$$

The cost function $\rho(S)$ is often referred to as *ratio cut* [44], while in the $\eta(S)$ is often referred to as *normalized cut* [36]. A sample graph partitioning indicated by dashed line for ratio and normalized cuts is shown below.



(a) ratio cut
$\rho(S) = \frac{(1.0+1.0)}{(2)(2)} = \frac{1}{2}$



(b) normalized cut
$\eta(S) = \frac{(3.0)}{(1.0+4.0)(1.0+4.0)} = \frac{3}{25}$

Notice that in parallel computing these cost functions have a direct relationship to minimizing communication (numerator), while keeping load balancing (denominator). For example, in distributed sparse matrix-vector multiplication, we can interpret the numerator as # of elements that needs to be sent between partitions and denominator as the work, measured in terms of # of rows in (3) or # of non-zero elements in (4), done per partition.

Also, notice that for unweighted graphs, the edge weights can be considered to be all one ($w = 1$). Letting $d(i)$ be the degree (# of edges) of node i, then

$$\begin{aligned} \mathrm{vol}(\partial(S)) &= |\partial(S)| \\ \mathrm{vol}(S) &= \sum_{i\in S} \mathrm{d}(i) \end{aligned} \tag{6}$$

In this case $\eta(S)$ is related to the *conductance* $\phi(S) = \mathrm{vol}(V)\eta(S)$, *Cheeger constant* and *isoperimetric number* of a graph [38, 40].

The minimum balanced cut problem stated in (3) and (4) is NP-complete [28]. However, there are algorithms that can often produce good enough approximation of the minimum balanced cut in a reasonable amount of time [22, 39].

3 Laplacian

We will once again assume that graph $G = (V, E)$ has edges only with positive weights $w > 0 : \forall (i, j, w) \in E$ and that if edge $(i, j, w) \in E$ then $(j, i, w) \in E$. Therefore, the graph has a symmetric weighted adjacency matrix A with nonnegative elements $a_{ij} \geq 0$. The treatment of graphs with negative weights or that correspond to nonsymmetric matrices is beyond the scope of this paper.

Let us define Laplacian matrix for graph $G = (V, E)$ as

$$L = D - A \tag{7}$$

where $A \in \mathbb{R}^{n \times n}$ is the weighted adjacency matrix of the graph and diagonal matrix $D = \text{diag}(A\mathbf{e})$, where vector $\mathbf{e} = (1, ..., 1)^T$.

Notice that D is simply the diagonal matrix, with each i-th diagonal element being the sum of all the elements in i-th row of A. Therefore, the Laplacian matrix remains the same independent of whether or not we include self-edges, diagonal elements, in the definition of the weighted adjacency matrix A.

The Laplacian matrix L is symmetric positive semi-definite. Notice that for vector $\mathbf{e}$ we have

$$L\mathbf{e} = \mathbf{0} \tag{8}$$

Therefore, if the graph $G = (V, E)$ is connected (there is a path between every pair of vertices in V) then L has the smallest eigenvalue 0 with the corresponding eigenvector being $\mathbf{e}$.

Also, notice that we can associate a vector $\mathbf{z} = (z_1, ..., z_n)^T$ with set S, by letting elements

$$z_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Then, using (1), (2), (5) and the definition of Laplacian in (7) we obtain

$$\mathbf{z}^T L \mathbf{z} = \sum_{i \in S \wedge j \notin S} a_{ij} = \sum_{(i,j,w) \in \partial(S)} w = \text{vol}(\partial(S)) \tag{10}$$

$$\mathbf{z}^T \mathbf{z} = |S| \tag{11}$$

Notice that (10) corresponds to the numerator, while (11) is related to the denominators of the minimization problems (3) and (4). Therefore, we may be able to express these minimization problems as a combination of vector $\mathbf{z}$ and another term, which does not affect (10). Fortunately, such term is readily available based on the result in (8).

Finally, we can state that the Laplacian matrix has the following property. Let vector $\mathbf{u} = \mathbf{z} - \alpha\mathbf{e}$, where $\alpha \in \mathbb{R}$ is some constant. Then, using (8) and (10) we obtain

$$\mathbf{u}^T L\mathbf{u} = \mathbf{z}^T L\mathbf{z} \quad = \quad \text{vol}(\partial(S)) \tag{12}$$

and consequently using (6) for unweighted graphs $\mathbf{u}^T L\mathbf{u} = |\partial(S)|$.

Also, notice that for $\alpha = |S|/|V|$ we have

$$\begin{aligned} \mathbf{u}^T\mathbf{u} &= \mathbf{z}^T\mathbf{z} - 2\alpha\mathbf{z}^T\mathbf{e} + \alpha^2\mathbf{e}^T\mathbf{e} = |S| - 2\alpha|S| + \alpha^2|V| = |S|(1-\alpha) = \\ &= \frac{|S|(|V| - |S|)}{|V|} = \frac{|S||\bar{S}|}{|V|} \end{aligned} \tag{13}$$

$$\mathbf{u}^T\mathbf{e} = \mathbf{z}^T\mathbf{e} - \alpha\mathbf{e}^T\mathbf{e} = |S| - \alpha|V| = 0 \tag{14}$$

and for $\alpha = \text{vol}(S)/\text{vol}(V)$ we have

$$\begin{aligned} \mathbf{u}^T D\mathbf{u} &= \mathbf{z}^T D\mathbf{z} - 2\alpha\mathbf{z}^T D\mathbf{e} + \alpha^2\mathbf{e}^T D\mathbf{e} = \text{vol}(S) - 2\alpha\text{vol}(S) + \alpha^2\text{vol}(V) = \\ &= \text{vol}(S)(1-\alpha) = \frac{\text{vol}(S)(\text{vol}(V) - \text{vol}(S))}{\text{vol}(V)} = \frac{\text{vol}(S)\text{vol}(\bar{S})}{\text{vol}(V)} \end{aligned} \tag{15}$$

$$\mathbf{u}^T D\mathbf{e} = \mathbf{z}^T D\mathbf{e} - \alpha\mathbf{e}^T D\mathbf{e} = \text{vol}(S) - \alpha\text{vol}(V) = 0 \tag{16}$$

Therefore, we may conclude that minimization problem (3) is equivalent to

$$\tilde{\rho}(S) = |V|\rho(S) = \min_{\mathbf{u}^T\mathbf{e}=0} \frac{\mathbf{u}^T L\mathbf{u}}{\mathbf{u}^T\mathbf{u}} \tag{17}$$

and minimization problem (4) is equivalent to

$$\tilde{\eta}(S) = \text{vol}(V)\eta(S) = \min_{\mathbf{u}^T D\mathbf{e}=0} \frac{\mathbf{u}^T L\mathbf{u}}{\mathbf{u}^T D\mathbf{u}} \tag{18}$$

Notice that since the vector $\mathbf{u} = \mathbf{z} - \alpha\mathbf{e}$ and values of $\mathbf{z}$ are constrained to be either 0 or 1 the integer programming problems (17) and (18) are still NP-complete. The main idea of spectral partitioning is to relax the constraint on vector $\mathbf{u}$ and let it take arbitrary real values, finding the solution of the relaxed problem and then extracting the approximation to the discrete solution from it.

Let us now assume that we are working with the relaxed version of the above optimization problems, where vector $\mathbf{u} \in \mathbb{R}^n$. Recall that $(0, \mathbf{e})$ is the smallest eigenpair of a Laplacian matrix L corresponding to a connected graph. Then, using Courant-Fischer theorem in [19], we can conclude that the solution to

the optimization problems (17) and (18) is the eigenvector associated with the second smallest eigenvalue of the eigenvalue problem

$$L\mathbf{u} = \lambda\mathbf{u} \qquad \text{and} \tag{19}$$
$$L\mathbf{u} = \lambda D\mathbf{u}, \tag{20}$$

respectively.

Once we find the eigenvector of interest we can convert the real values to discrete 0 and 1 using heuristics, such as scaling and looking for a sign change, or sorting and looking for a gap in the values [32, 35]. There is no theoretical guarantee that the obtained approximate solution will closely match the optimal discrete solution, in fact there are known examples where this will not be the case [20], but in practice we often do obtain a good approximation [39].

4 Multiple Partitions

Let us now generalize the spectral partitioning from bisection to multiple partitions. We will closely follow the generalization proposed in [28], and we include its derivation here mostly for completeness.

First, let us slightly reformulate the optimization problem (3) and (4) in terms of partitions rather than cuts. Notice that $\mathrm{vol}(\partial(S)) = \mathrm{vol}(\partial(\bar{S}))$, then using $|V| = |S| + |\bar{S}|$ we obtain

$$\tilde{\rho}(S) = \min_S \frac{|V|\mathrm{vol}(\partial(S))}{|S||\bar{S}|} = \min_S \left(\frac{\mathrm{vol}(\partial(S))}{|S|} + \frac{\mathrm{vol}(\partial(\bar{S}))}{|\bar{S}|} \right) \tag{21}$$

and using $\mathrm{vol}(V) = \mathrm{vol}(S) + \mathrm{vol}(\bar{S})$ we have

$$\tilde{\eta}(S) = \min_S \frac{\mathrm{vol}(V)\mathrm{vol}(\partial(S))}{\mathrm{vol}(S)\mathrm{vol}(\bar{S})} = \min_S \left(\frac{\mathrm{vol}(\partial(S))}{\mathrm{vol}(S)} + \frac{\mathrm{vol}(\partial(\bar{S}))}{\mathrm{vol}(\bar{S})} \right) \tag{22}$$

Therefore, it is natural to generalize the spectral partitioning for p partitions as

$$\tilde{\rho}(S_1, ..., S_p) = \min_{S_1,...,S_p} \sum_{k=1}^{p} \frac{\mathrm{vol}(\partial(S_k))}{|S_k|} \tag{23}$$

and

$$\tilde{\eta}(S_1, ..., S_p) = \min_{S_1,...,S_p} \sum_{k=1}^{p} \frac{\mathrm{vol}(\partial(S_k))}{\mathrm{vol}(S_k)} \tag{24}$$

Let us define a tall matrix $U = [u_{i,k}]$, that can be interpreted as a set of vectors $U = [\mathbf{u}_1, ..., \mathbf{u}_p]$ where each vector $\mathbf{u}_k$ corresponds to set S_k, with elements

$$u_{i,k} = \begin{cases} \beta & \text{if } i \in S_k \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

where $\beta \in \mathbb{R}$ is some constant. Notice that

$$\mathbf{u}_k^T L \mathbf{u}_k = \beta^2 \sum_{i \in S \wedge j \notin S} a_{ij} = \beta^2 \sum_{(i,j,w) \in \partial(S_k)} w = \beta^2 \text{vol}(\partial(S_k)) \tag{26}$$

then for $\beta = \frac{1}{\sqrt{|S_k|}}$ we have

$$\begin{aligned} \mathbf{u}_k^T L \mathbf{u}_k &= \frac{\text{vol}(\partial(S_k))}{|S_k|} \qquad \text{for } k = 1, ..., p \\ U^T U &= I \end{aligned} \tag{27}$$

and for $\beta = \frac{1}{\sqrt{vol(S_k)}}$ we have

$$\begin{aligned} \mathbf{u}_k^T L \mathbf{u}_k &= \frac{\text{vol}(\partial(S_k))}{vol(S_k)} \qquad \text{for } k = 1, ..., p \\ U^T D U &= I \end{aligned} \tag{28}$$

where $I \in \mathbb{R}^{p \times p}$ is the identity matrix. Therefore using (27) and (28) we have

$$\tilde{\rho}(S_1, ..., S_p) = \min_{S_1, ..., S_p} \sum_{k=1}^{p} \mathbf{u}_k^T L \mathbf{u}_k = \min_{U^T U = I} \text{Tr}(U^T L U) \tag{29}$$

and

$$\tilde{\eta}(S_1, ..., S_p) = \min_{S_1, ..., S_p} \sum_{k=1}^{p} \mathbf{u}_k^T L \mathbf{u}_k = \min_{U^T D U = I} \text{Tr}(U^T L U) \tag{30}$$

where Tr(.) is the trace of a matrix (sum of its diagonal elements) [19].

These integer optimization problems are NP-hard, but as before this requirement can be relaxed in order to find an approximate solution. If we let tall matrix $U \in \mathbb{R}^{n \times p}$ then following Courant-Fischer theorem in [19], we again can conclude that the solution to the optimization problems (29) and (30) are the eigenvectors associated with p smallest eigenvalues of the eigenvalue problem

$$LU = U\Sigma \qquad \text{and} \tag{31}$$

$$LU = DU\Sigma, \tag{32}$$

where $\Sigma = \text{diag}([\lambda_1, ..., \lambda_p]) \in \mathbb{R}^{p \times p}$, respectively.

When finding multiple partitions it is no longer clear how we could use straightforward heuristics to identify the partitions of interest. Notice that we are now looking for a pattern of 0 and 1 in two dimensions. Therefore, it is difficult to find a single element from the row of matrix U to represent a graph node. For example, if we pick the largest number in the row and make partition assignments based on it, we neglect the interaction of numbers within columns. The smaller number within a row could still be the largest within its column and vice-versa.

Since we are working in multiple dimensions, it is natural to use the distance between points as a metric of how to group them. In this case, if we interpret each row of the matrix U as a point in a p-dimensional space then it becomes natural to use a clustering algorithm, such as k-means [1, 27] or Voronoi diagrams [5, 43] to identify the p distinct partitions.

Notice that the number of partitions identified by the clustering algorithm does not necessarily need to match the number of computed eigenvectors. Although in theory they should be the same, recall that we are working with an approximation to the discrete problem, therefore the real question is how good is our approximation and if it has any particular meaning. For example, if we compute $\log_2(p)$ eigenvectors and later identify p partitions, the result can be interpreted as an approximation to the problem of minimizing number of hops performed during communication on hypercube network [16].

5 Spectral Graph Partitioning

We are now ready to describe an outline of the spectral graph partitioning technique, shown in Alg. 1.

Algorithm 1 Spectral Graph Partitioning

1: Let $G = (V, E)$ be an input graph and A be its weighted adjacency matrix.
2: Let $B = I$ (ratio cut) or $B = D$ (normalized cut), where $D = \text{diag}(A\mathbf{e})$.
3: Let p be the number of desired partitions.
4: Set the Laplacian matrix $L = D - A$.
5: Find p smallest eigenpairs of the eigenvalue problem $LU = BU\Sigma$.
6: Scale the p eigenvectors U by row or by column (optional).
7: Run clustering algorithm, such as k-means, on points defined by rows of U.

In our numerical experiments we have found that the eigenvalue solver is a critical part of this technique. First, it is the most time consuming part of the

computation, often taking up to 80% of the total time. Second, the accuracy of the solution has a significant impact on the quality of the partitioning, with insufficient accuracy usually resulting in poor approximation. Finally, a failure in convergence of the eigenvalue solver results in failure of the entire algorithm.

Therefore, it is important to carefully analyse and implement the solution of the eigenvalue problem (on line 5), which will be the focus of the next section.

5.1 Eigenvalue Problem

Let the Laplacian $L \in \mathbb{R}^{n \times n}$ be symmetric positive semi-definite and $B \in \mathbb{R}^{n \times n}$ be symmetric positive definite diagonal matrix. We are interested in computing p smallest eigenpairs of the generalized eigenvalue problem

$$L\mathbf{u} = \lambda B\mathbf{u} \tag{33}$$

which corresponds to pencil (L, B) [41].

Let us assume that the eigenvalues of the Laplacian are ordered as shown in

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n \tag{34}$$

and examine a few approaches.

First, we can use a few iteration of the power method [2] to compute an estimate of the largest eigenvalue $\tilde{\lambda}_n$ and then solve the shifted problem

$$(\tilde{\lambda}_n I - \bar{L})\bar{\mathbf{u}} = \mu\bar{\mathbf{u}} \tag{35}$$

where $\bar{L} = B^{-\frac{1}{2}} L B^{-\frac{1}{2}}$, $\bar{\mathbf{u}} = B^{\frac{1}{2}}\mathbf{u}$ and $\mu_i = \tilde{\lambda}_n - \lambda_i$ for $i = 1, ..., n$. Notice that $\tilde{\lambda}_n I - \bar{L}$ is symmetric positive semi-definite and that the smallest eigenvalues of (33) correspond to the largest eigenvalues of (35). Therefore, we are able to use subspace iteration [10, 42] method which converges well for the largest eigenvalues and require only matrix-vector multiplication per iteration. Unfortunately, the convergence of this method is often governed by the ratio between the eigenvalues, which becomes very poor after the shift.

Second, we can keep the original formulation in (33), but then in order to find the solution quickly we need to solve a linear system per iteration. Solving linear systems with Laplacian matrix L is both expensive and complicated because it is singular. One could avoid some of the issues related to singularity by working with the projected system of the form $(I - \mathbf{u}\mathbf{u}^T)L(I - \mathbf{u}\mathbf{u}^T)$, which is roughly the approach followed by more advanced eigenvalue solvers, such as Tracemin [34] and Jacobi-Davidson [37], but we have still found it to be too expensive.

To illustrate the tradeoffs consider an eigenvalue distribution where $\lambda_2 = 0.1$, $\lambda_{p+1} = 1.0$ and $\lambda_n = 10.0$, which is not atypical for the Laplacian matrix.

Consider the subspace iteration method with p vectors, where theoretically the error in the second eigenvector is reduced by λ_{p+1}/λ_2 per iteration. Then, $\lambda_{p+1}/\lambda_2 = 10$, while $\mu_{p+1}/\mu_2 = 1.1$. In this case, the first approach will take 97 iterations (with each requiring a matrix-vector multiplication with L), while the second will take 4 iterations (with each requiring a linear system solve with L) to reduce the error by 10^{-4}. In the first approach we perform many cheap steps, while in the second we perform few computational intensive steps.

Third, is the approach taken by most people who have worked with spectral graph partitioning. It is simply to run the Lanczos method [14] on the original formulation (33) and extract the approximation to the smallest eigenvalues from the Krylov subspace generated by it. In general, the Lanczos method will indeed converge to both the largest and smallest eigenvalues, but it does so much more slowly for the latter. Also, Lanczos can not handle eigenvalues with multiple multiplicity and in some cases may converge to spurious eigenvalues [9]. These characteristics make it dangerous to use for this particular problem.

To solve the above challenges we propose to use the LOBPCG method [4, 24]. It is in our opinion the preferred way to solve this problem for the following reasons: (i) it works with the original eigenvalue problem and does not need to solve a linear system with L, (ii) its convergence is similar to Lanczos, but the algorithm can handle eigenvalues with multiple multiplicity and is more numerically stable, and (iii) perhaps most import of all, it allows for preconditioning of the eigenvalue problem [25]. The Laplacian matrix has a very particular structure and properties and is well suited for preconditioning. These unique characteristics allow LOBPCG to address all the challenges in solving the eigenvalue problem arising in spectral partitioning.

There exist several variations of the LOBPCG algorithm proposed by Knyazev. We present a scheme that was refined to specifically address the Laplacian (generalized) eigenvalue problem in Alg. 2. An experienced computational scientist will immediately recognize the Rayleigh-Ritz method and formation of sections of pencil (L, B) as some of the building blocks of the algorithm, but he will also notice a particular restart strategy that is crucial to the numerical stability of the algorithm. These will be discussed next.

For completeness, we recall that a section of a pencil (A, B) is a tall matrix X that satisfies

$$\begin{aligned} X^T A X &= \Sigma \\ X^T B X &= I \end{aligned} \tag{36}$$

where Σ is a diagonal matrix. The reader will see the computations of sections on lines 7-11 and 27-30 in Alg. 2.

Algorithm 2 LOBPCG Eigenvalue Solver (Detailed Algorithm)

Let $L \in \mathbb{R}^{n \times n}$ be a symmetric positive semi-definite Laplacian matrix
Let $B \in \mathbb{R}^{n \times n}$ be an arbitrary symmetric positive definite matrix
We are interested in p eigenpairs of the eigenvalue problem $L\mathbf{u} = \lambda B\mathbf{u}$

Let $M \in \mathbb{R}^{n \times n}$ be a preconditioner matrix and constant $\bar{\kappa}_1 = -\log_{10}(\epsilon)/2$
Let $U_0 \in \mathbb{R}^{n \times p}$ be a random initial guess and start with $P \in \mathbb{R}^{n \times p}$ to 0
Compute $G = U_0^T B U_0$ ▷ B-orthonormalize U
Compute $\bar{U} = U_0 S^{-1}$, where Cholesky factorization of $G = S^T S$
Compute $H = \bar{U}^T A \bar{U}$ ▷ Compute section U of pencil (L, B)
Compute eigenvalue decomposition $H = W \Sigma W^T$
Compute $U = \bar{U} W$ ▷ Check $U^T B U = I$ and $U^T A U = \Sigma$
for $j = 1, 2, ...$convergence **do**
    Compute residual $R = LU - BU\Sigma$ ▷ Check convergence $||R||_2 <$ tol.
    Solve linear system $M\bar{R} = R$ ▷ Preconditioning
    Compute $G = \bar{R}^T B \bar{R}$ ▷ B-orthonormalize R
    Compute $R = \bar{R} S^{-1}$, where Cholesky factorization of $G = S^T S$
    Compute $G = Y^T B Y$, with $Y = [U, R, P]$ ▷ B-orthonormalize Y
    Compute condition number $\bar{\kappa}_{j+1} = \log_{10}(\kappa(G)) + 1$ ▷ Restart strategy
    Compute average $\gamma_j = \frac{(\sum_{l=i+1}^{j} \bar{\kappa}_l)}{(j-i)}$, with $i = \max(0, j - 9 - \log(p))$
    if $((j = 1)$ **or** $(\bar{\kappa}_{j+1} > 8)$ **or** $((\bar{\kappa}_{j+1}/\gamma_j > 2)$ **and** $(\bar{\kappa}_{j+1} > 2)))$ **then**
        Compute $G = Y^T B Y$, with $Y = [U, R]$ ▷ Remove P from Y and G
        Compute $\bar{\kappa}_{j+1} = \log_{10}(\kappa(G)) + 1$ ▷ Recompute condition number
        Set restart= j
    else
        Set restart= -1
    end if
    Compute $\bar{Y} = Y S^{-1}$, where Cholesky factorization of $G = S^T S$
    Compute $H = \bar{Y}^T A \bar{Y}$ ▷ Compute section Y of pencil (L, B)
    Compute eigenvalue decomposition $H = W \Sigma W^T$
    Compute $U = \bar{Y} W(1 : 3p, 1 : p)$ ▷ Check $U^T B U = I$ and $U^T A U = \Sigma$
    if j = restart **then**
        Compute $P = RW(p + 1 : 2p, 1 : p)$
    else
        Compute $P = RW(p + 1 : 2p, 1 : p) + PW(2p + 1 : 3p, 1 : p)$
    end if
    Compute $G = P^T B P$; ▷ B-orthonormalize P
    Compute $P = P S^{-1}$, where Cholesky factorization of $G = S^T S$
end for

The algorithm implements a restarting strategy on lines 18-26 that was proposed by Knyazev in his MATLAB code. This restarting strategy keeps track of conditioning of matrix G through $\bar{\kappa}$ and will restart the algorithm if it is either very high or if it jumps significantly compared to an average γ obtained from previous iterations. Notice that $\bar{\kappa}$ is based on the condition number $\kappa(G)$ and machine precisio ϵ. The condition number $\kappa(G)$ is defined as ratio of largest and smallest singular values of G and is equivalent to the ratio of its largest and smallest eigenvalues because G is symmetric positive semi-definite [19]. Therefore, in our case $\kappa(G)$ can be computed using an eigenvalue decomposition. We point out that there exist other restarting strategies, such as the ones suggested in [18].

The matrix $Y \in \mathbb{R}^{n\times 3p}$ on lines 17 and 21 need not be formed explicitly, as it is sufficient to organize $X \in \mathbb{R}^{n\times p}$, $R \in \mathbb{R}^{n\times p}$ and $P \in \mathbb{R}^{n\times p}$ consecutively in memory an then alias them with Y. Finally, the notation $W(i:k, 1:p)$ on lines 30-34 means that we are selecting a submatrix of W that is located in rows i through k and columns 1 through p.

Also, it is implicitly assumed in the algorithm that the eigenvalue decomposition $H = W\Sigma W^T$ on lines 10 and 29 is such that eigenvalues are ordered from smallest to largest in $\Sigma = \text{diag}(\lambda_1, ..., \lambda_{3p})$, therefore by selecting the submatrix $\Sigma(1:p, 1:p)$ we would obtain p smallest eigenvalues. Not all numerical software packages return results ordered in this fashion, therefore additional sorting might be needed to conform to this implicit assumption.

Finally, we point out that we do not implement filtering of eigenpairs that have already converged to the required tolerance also proposed by Knyazev. The implementation of this filtering would either require shuffling of the data so that the active eigenvectors are located next to each other in memory or masking of the computation for the inactive eigenvectors, which would prevent us from using the standard libraries, such as CUBLAS [45]. The filtering might become useful if one is looking for hundreds of eigenvectors, but in our case we have experimented with less than 32 partitions. Therefore, given the computational resource of the GPU, we have found that simply running with all eigenvectors being active to completion is easier and has no adverse effects on numerical stability of the algorithm.

5.2 Clustering Problem

At this point we have obtained the p smallest eigenvalues $\Sigma = \text{diag}(\lambda_1, ..., \lambda_n)$ and corresponding eigenvectors $U \in \mathbb{R}^{n\times p}$ of generalize eigenvalue problem (33). In order, to solve solve the discrete problems (29) and (30) let us now find the assignment of nodes into partitions.

Let us interpret each row of U as a point $\mathbf{x}_i$ in p-dimensional space, so that

$$U = \begin{pmatrix} u_{11} & \dots & u_{1p} \\ u_{21} & \dots & u_{2p} \\ \vdots & & \vdots \\ u_{n1} & \dots & u_{np} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} \tag{37}$$

Then, we will need to find cluster sets S_k for $k = 1, ..., p$, each with a centroid (point in the center) $\mathbf{c}_k$, such that

$$\min_{S_k} \sum_{k=1}^{p} \sum_{i \in S_k} ||\mathbf{x}_i - \mathbf{c}_k||_2^2 \tag{38}$$

The exact solution of this problem is NP-complete, but we can find an approximation to its solution using many variations of the k-means clustering algorithm [1, 27]. The outline of the k-means Lloyd algorithm is shown in Alg. 3.

Algorithm 3 K-means Lloyd Algorithm

1: Let centroids $\mathbf{c}_k$ for $k = 1, ..., p$ be an initial guess.
2: **for** $j = 1, ..., p$ **do** ▷ Assign points $\mathbf{x}_i$ to clusters S_k
3: Compute distance $d_{ij} = ||\mathbf{x}_i - \mathbf{c}_j||_2$ for $i = 1, ..., n$
4: **end for**
5: Assign points $\mathbf{x}_i$, so that cluster $S_k = \{i : \ d_{ik} \leq d_{ij}\}$ for $j = 1, ..., p$
6: **for** $l = 1, 2, ...$convergence **do**
7: Compute error $\epsilon_l = \sum_{k=1}^{p} \sum_{i \in S_k} d_{ik}^2$ ▷ Check convergence $\frac{\epsilon_l}{\epsilon_{l-1}} <$ tol.
8: **for** $k = 1, ..., p$ **do** ▷ Update centroids $\mathbf{c}_k$ of clusters S_k
9: Compute centroid $\mathbf{c}_k = \left(\sum_{i \in S_k} \mathbf{x}_i\right) / |S_k|$
10: **end for**
11: **for** $j = 1, ..., p$ **do** ▷ Assign points $\mathbf{x}_i$ to clusters S_k
12: Compute distance $d_{ij} = ||\mathbf{x}_i - \mathbf{c}_j||_2$ for $i = 1, ..., n$
13: **end for**
14: Assign points $\mathbf{x}_i$, so that cluster $S_k = \{i : \ d_{ik} \leq d_{ij}\}$ for $j = 1, ..., p$
15: **end for**

Finally, we point out that in practice we avoid performing scaling corresponding to line 6 in Alg. 1 as suggested in [28, 31]. We have found them to be detrimental to the quality of partitioning of graphs corresponding to social networks. Instead we use k-means++ clustering algorithm where starting points for centroids are chosen based on a particular probability distribution [1].

6 Numerical Experiments

Let us now study the performance and quality of the partitioning obtained by the proposed spectral graph partitioning algorithm on a sample of graphs from the `DIMACS10` graph collection [46, 47], shown in Tab. 1. All numerical experiments are performed on a workstation with Ubuntu 14.04 operating system, gcc 4.8.4 compiler, CUDA Toolkit 8.0 software and Intel Core i7-3930K CPU 3.2 GHz and Nvidia Tesla K40c GPU hardware.

In order to avoid discrepancies between different software packages, we always set compiler optimization flags to `-O3` and when possible perform all computations with 32-bit integer and double precision floating point types. In Alg. 1, on line 5 we let the stopping criteria for the eigenvalue solver be residual $||L\mathbf{u}_1 - \lambda B\mathbf{u}_1||_2 \leq 10^{-2}$ and max. # of iterations ≤ 512 (with 128 restart for Lanczos). Also, on line 6 we do not use any scalings of the obtained eigenvectors. As mentioned earlier, we found them to be detrimental to partition quality. Finally, on line 7 we let the stopping criteria for k-means++ algorithm be error ratio $\frac{\epsilon_l}{\epsilon_{l-1}} \leq 10^{-2}$ and max. # of iterations ≤ 16.

Also, we always perform a reordering of the original matrix before starting the computation. The reordering is based on the coloring of the associated graph. It is performed using JPL algorithm implemented in the `csrcolor` routine in CUSPARSE [45]. It improves the parallelism available in the incomplete-LU factorization (ILU0), which we use as a preconditioner for the LOBPCG [30]. We point out that this reordering does not change the structure of our graph, it simply relabels its vertices. It is performed in all cases for consistency.

Let us now look at the effect of using different eigenvalue solvers in Alg. 1. We compare the # of iterations (taken to convergence), performance (in terms of time) as well as quality of the partitioning (as measured by cost function $\tilde{\eta}$) for Lanczos, LOBPCG and LOBPCG preconditioned with ILU0 on Fig. 1 - 3. Notice that Lanczos is the fastest algorithm, while unpreconditioned LOBPCG almost always obtains superior quality. The LOBPCG(ILU0) algorithm is often as fast as Lanczos and usually obtains the solution with the same or better quality as its unpreconditioned version. In fact, the only reason for using the latter might be the occasional failure to compute the incomplete-LU due to singularity of the Laplacian, denoted by † in Tab. 2 - 5 in the Appendix.

We emphasize that in our spectral scheme we always solve a single eigenvalue problem even when partitioning the graph into multiple partitions. We find several eigenvectors and treat them as a continuous solution of the problem, then we use k-means++ to transform it into a discrete solution. This is different from recursive spectral bisection, where multiple eigenvalue problems are solved at every level of the algorithm [35].

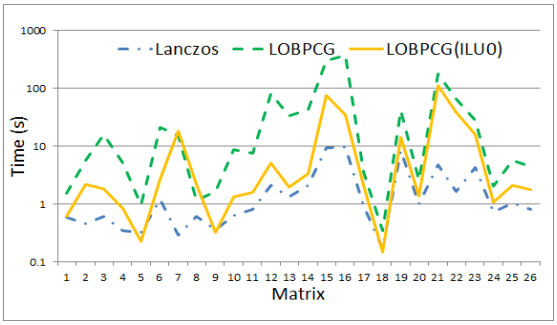


Figure 1: Time(s) consumed during spectral graph partitioning into 31 partitions

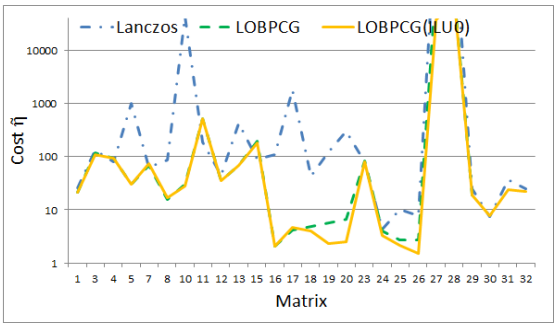


Figure 2: Cost $\tilde{\eta}$ obtained during spectral graph partitioning into 31 partitions

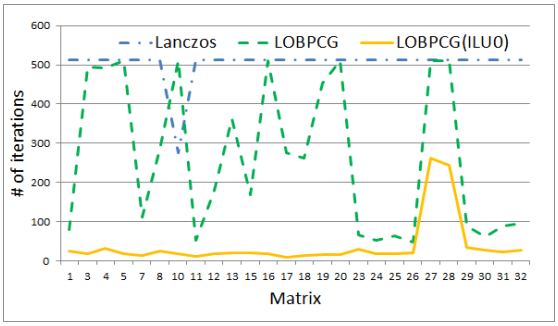


Figure 3: # of iterations performed during spectral graph partitioning into 31 partitions

We point out that we choose to partition the graph into 31 partitions arbitrarily. Similar results hold for a different number of partitions as well. Notice that since we do not recursively partition the graph, unlike multi-level graph partitioning schemes, we have no preference for choosing # of partitions to be $p = 2^k$ for some constant k and are able to find all of them at once.

Also, we choose to measure the quality of the partition using normalized cut (24) rather than ratio cut (23). The normalized and ratio cut are similar, but are formulated to solve slightly different discrete optimization problems and one might be preferable over the other depending on the application. We compare these approaches using LOBPCG(ILU0) and Lanczos algorithms for 2 partitions on Fig. 4 - 5. As a baseline we include a comparison with the spectral graph partitioning implemented in CHACO [17]. In the latter case we are mostly interested in the performance of the spectral partitioning using Lanczos, therefore we disable local improvements using Kernighan-Lin algorithm [23].

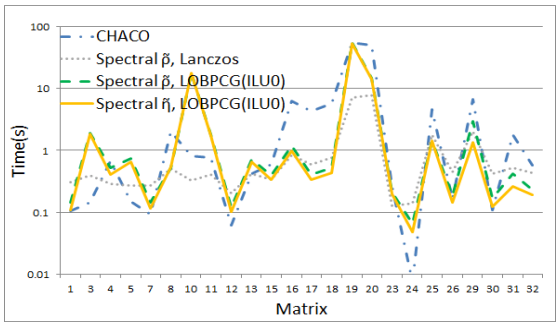


Figure 4: Time(s) consumed during spectral graph partitioning into 2 partitions

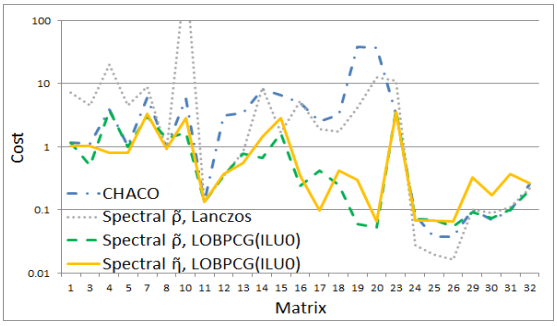


Figure 5: Cost $\tilde{\rho}$ and $\tilde{\eta}$ obtained during spectral graph partitioning into 2 partitions

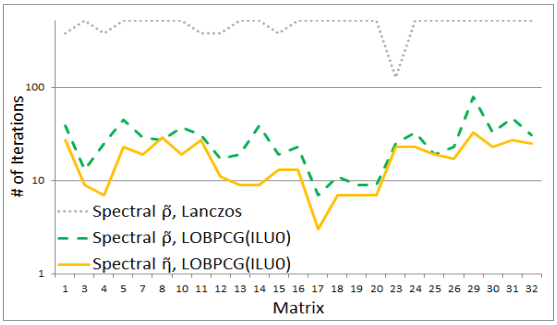
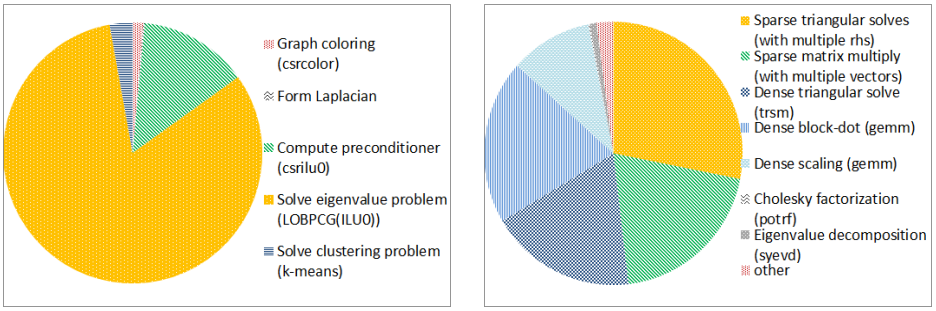


Figure 6: # of iterations performed during spectral graph partitioning into 2 partitions

Also, we point out that CHACO attempts to provide a load balanced cut within a fixed threshold ϵ, so that for instance $|S_k| \leq (1+\epsilon)|V|/p$. Therefore, CHACO's cost function is similar to ratio cut, but the clustering at the end is biased towards first of all providing a load balanced partitioning, while still minimizing the edge cuts. We use CHACO version 2.2 and do not change these default settings in our experiments.

Notice that in general our spectral graph partitioning using LOBPCG(ILU0) outperforms CHACO spectral graph partitioning using Lanczos. In particular, the shape of the curves representing the ratio cut performance (time) and quality (in terms of cost $\tilde{\rho}$) resemble the ones obtained by CHACO. This is not surprising given that they are both working on a similar formulation of the problem. On the other hand, the shape of the curves representing the normalized cut $\tilde{\eta}$ on Fig. 4 - 5 is somewhat different, but the approach still performs well. It is interesting that in our experiments the number of eigenvalue solver iterations needed to obtain normalized cut $\tilde{\eta}$ is always smaller than the ones needed to obtain ratio cut $\tilde{\rho}$, see Fig. 6. Also, we conjecture that the quality of the solution obtained by our Lanczos approach is not always as good as that of CHACO's Lanczos approach, because the latter uses selective orthogonalization to obtain more accurate eigenvectors. Finally, we point out that all the approaches could still benefit from local refinements, such as Kernighan-Lin algorithm.

It is also important to point out that our new spectral approach has been implemented on the GPU, while CHACO is implemented on the CPU. The implementation of LOBPCG eigenvalue solver in Alg. 2 relies on the CUSPARSE, CUBLAS and CUSOLVER libraries from the CUDA Toolkit [45], while k-means in Alg. 3 required custom implementation. The precise speedup obtained by using GPU is difficult to calculate because it involves comparisons between large and complex software packages, that even in the simple case implement slightly



(a) Spectral scheme (b) Eigenvalue solver LOBPCG(ILU0)

Figure 7: Profiling of time spent in spectral graph partitioning on citationCiteseer graph

different variations of the algorithm. However, we can estimate the obtained speedup. Recall that we have empirically found that up to 80% of spectral graph partitioning time is spent in the eigenvalue solver, see a sample profiling plot on Fig. 7a. Also, note that the time spent in the eigenvalue solver is often governed by the sparse matrix-vector multiplication and the preconditioning with incomplete-LU factorization, as shown in Fig. 7b. The behavior of these algorithms have been extensively studied in [29, 30]. Therefore, based on these studies, we can estimate with some confidence that we obtain a $2-4\times$ additional speedup by using GPUs.

Let us now compare the new spectral graph partitioning approach with METIS [21], one of the most popular graph partitioning software package. METIS implements a multi-level graph partitioning scheme, which works fundamentally differently from spectral graph partitioning. It agglomerates nodes of the graph in order to create a hierarchy, where the fine level represents the original graph and the coarse level represents its reduced form. The partitioning is performed on the coarse level and results are propagated back to the fine level [22]. The multi-level graph partitioning has many parameters. We use METIS version 5.1.0 and choose the default parameters in our experiments.

In order to better understand the results of the experiments, we will separate our graphs from Tab. 1 into two collections. The first is composed of graphs generated from different networks (matrices $1-28$). It includes graphs with low maximum degree obtained from census data or road networks (matrices $25-28$), and graphs with high maximum degree that have been generated from star nodes (matrices $1-16$) or have been created based on citations (matrices $16-20$). Notice that the latter often resemble graphs with power law-like distribution of edges per node and social networks. The second collection is composed of graphs that result from discretization of partial differential equations (PDEs), using finite-element or other similar methods (matrices $29-32$).

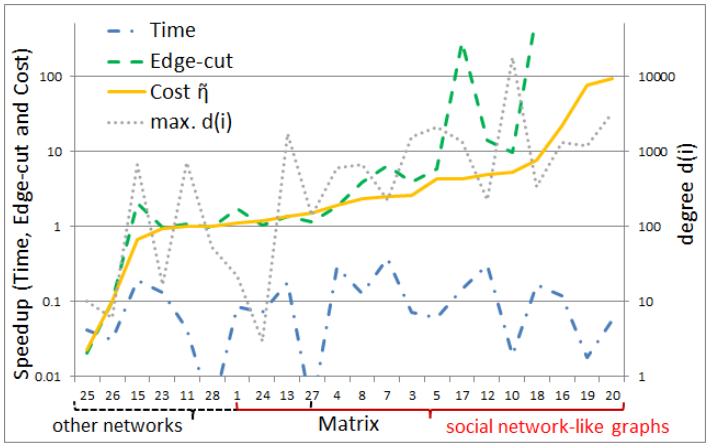


Figure 8: Comparison of Spectral vs. METIS graph partitioning into 31 partitions (Networks)

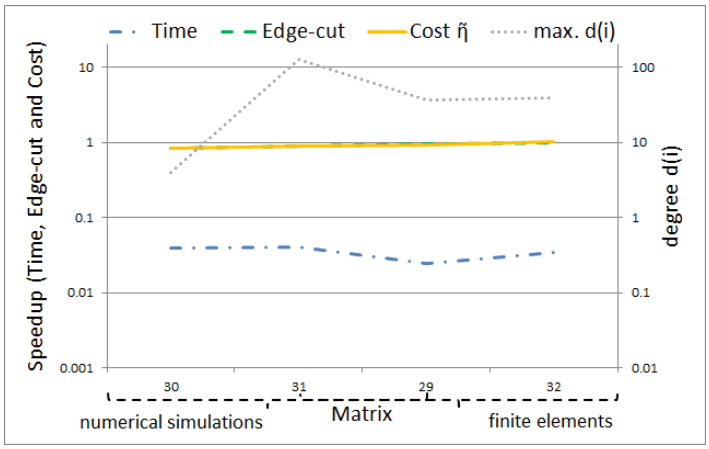


Figure 9: Comparison of Spectral vs. METIS graph partitioning into 31 partitions (PDEs)

The performance and quality of the partitions generated by the spectral graph partitioning and METIS for both collections is shown in Fig. 8 and 9. Notice that the behavior of two partitioning schemes for these types of graphs is strikingly different. The multi-level partitioning is always fast (see dash-doted blue line), and finds good partitioning for the second collection of graphs. The spectral graph partitioning is slow, but it often obtains significantly better partitioning for the first collection (see solid yellow and dashed green lines), especially when graphs resemble social networks (annotated in red in Fig. 8). Notice that for social network-like graphs the spectral partitioning can be $10\times$ slower, but can also produce partitioning that has $10\times$ higher quality (cost $\tilde{\eta}$).

Our conjecture is that the agglomeration of nodes often works well for graphs with low and more or less regular degree (# of edges per node), such as the ones arising from PDEs. In this case the graph partitioning only needs access to local information to construct a hierarchy such that the coarse level is representative of the fine graph. On the other hand for social network-like graphs, which contain star nodes with high and irregular degree, access to global information is required to perform high quality partitioning. Therefore, the spectral graph partitioning finds a better quality solution in this case.

Finally, we point out that Fig. 8 shows a trend, where the quality of partitioning (cost $\tilde{\eta}$) improves with the increase in the maximum degree of nodes in a graph, denoted by max. $d(i)$. The latter quantity is plotted on the secondary y-axis in order not to interfere with the rest of the plot, and the cost $\tilde{\eta}$ provides an almost perfect trend for it. This once again points to the potential link between degree of nodes in a graph and the quality of partitioning obtained by different partitioning schemes.

The detailed results of all numerical experiments are shown in Tab. 1 - 5 in the Appendix. In these results † denotes failure of the method, which in the case of spectral graph partitioning is due to failure of the incomplete-LU factorization.

7 Conclusion and Future Work

In this paper we have reviewed the theory of spectral graph partitioning. Also, we have developed a novel variation of the spectral technique that relies on preconditioned eigenvalue solver LOBPCG and a k-means++ algorithm on GPUs. This more numerically stable eigenvalue solver allowed us to achieve convergence on a larger number of problems, and the use of incomplete-LU preconditioning ultimately resulted in higher quality of graph partitioning.

We have compared our approach with spectral bisection in CHACO [17] and multi-level techniques implemented in METIS [21] software packages. Also, we have shown that the performance of spectral graph partitioning and multi-level schemes is starkly different for two classes of problems: (i) social network graphs that often have power law-like distribution of edges per node and (ii) meshes arising from discretization of partial differential equations. Although, in our numerical experiments it became clear that the multi-level schemes are almost always faster, we have also shown that for the former class of problems the spectral schemes can achieve a significantly higher quality of the partitioning.

An interesting characteristic of spectral graph partitioning is that we could try to automatically detect the number of clusters into which we should partition the graph. On one hand, we can look for a gap between p eigenvalues of (33). On the other hand, we can use x-means algorithm to determine the best k to

use in k-means [33]. Exploring these and other techniques for adaptive graph partitioning is a subject of future work. Also, we have only experimented with incomplete-LU as a preconditioner for the eigenvalue solver LOBPCG. However, other choices are available, such as Jacobi or more advanced support graph preconditioners [3]. In the future we would like to explore these in more detail.

Finally, many of the interesting social network-like graphs are very large and do not fit into a memory of a single GPU or even of a single node. Therefore, developing spectral graph partitioning on distributed platforms is critical for these applications and we plan to pursue it in the future.

8 Acknowledgements

The authors would like to acknowledge Erik Boman, Joe Eaton, Michael Garland, Bruce Hendrickson and Robert Leland for their useful comments and suggestions.

References

[1] D. Arthur and S. Vassilvitskii, *K-means++: The Advantages of Careful Seeding*, Proc. 18th Annual ACM-SIAM Symposium on Discrete algorithms, pp. 1027-1035, 2007.

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, PA, 2000.

[3] M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen and S. Toledo, *Support-Graph Preconditioners*, SIAM Journal on Matrix Analysis and Applications, Vol. 27, 930-951, 2006.

[4] E. Boman, K. Devine, R. Lehoucq, N. Slattengren and H. Thornquist, *Installing the Anasazi Eigensolver Package with Application to Some Graph Eigenvalue Problems*, Technical Report SAND2014-16610, Sandia National Labs, 2014.

[5] A. Bowyer, *Computing Dirichlet Tessellations*, The Computer Journal, Vol. 24, pp. 162-166, 1981.

[6] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders and C. Schulz, *Recent Advances in Graph Partitioning*, Preprint arXiv:1311.3144, 2013.

[7] P. K. Chan, M. D. F. Schlag and J. Y Zien, *Spectral K-Way Ratio-Cut Partitioning and Clustering*, IEEE Trans. Computer-Aided Design, Vol. 13, pp. 1088-1096, 1994.

[8] F. R. K. Chung, *Spectral Graph Theory*, American Mathematical Society, Providence, RI, 1997.

[9] J. K. Cullum and R. A. Willoughby *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, SIAM, Philadelphia, PA, 2002.

[10] M. Clint and A. Jennings, *The Evaluation of Eigenvalues and Eigenvectors of Real Symmetric Matrices by Simultaneous Iteration*, The Computer Journal, Vol. 13, pp. 76-80, 1970.

[11] W. E. Donath and A. J. Hoffman, *Lower Bounds for the Partitioning of Graphs*, IBM Journal of Research and Development, Vol. 17, pp. 420-425, 1973.

[12] C. M. Fiduccia and R. M. Mattheyses, *A Linear-Time Heuristic for Improving Network Partitions*, Proc. 19th Conference on Design Automation, pp. 175-181,1982.

[13] M. Fiedler, *Algebraic Connectivity of Graphs*, Czechoslovak Mathematical Journal, Vol. 23, pp. 298-305, 1973.

[14] G. H. Golub and D. P. O'Leary, *Some History of the Conjugate Gradient and Lanczos Algorithms*, SIAM Review, Vol. 31, pp. 50-102, 1989.

[15] L. Hagen, *New Spectral Methods for Ratio Cut Partitioning and Clustering*, IEEE Trans. Computer-Aided Design, Vol. 11, pp. 1074-1085, 1992.

[16] B. Hendrickson and R. Leland, *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*, Technical Report SAND92-1460, Sandia National Labs, 1992.

[17] B. Hendrickson and R. Leland, *The CHACO User's Guide 2.0*, Technical Report SAND95-2344, Sandia National Labs, 1995.

[18] U. Hetmaniuk and R. Lehoucq, *Basis Selection in LOBPCG*, Journal of Computational Physics, Vol. 218, pp. 324-332, 2006.

[19] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, NY, 1999.

[20] S. Guattery and G. L. Miller, *On the Quality of Spectral Separators*, SIAM Journal on Matrix Analysis and Appl., Vol. 19, pp. 701-719, 2006.

[21] G. Karypis and V. Kumar, *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, 1995

[22] G. Karypis and V. Kumar, *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, SIAM Journal on Scientific Computing, Vol. 20, pp. 359-392, 1998.

[23] B. W. Kernighan and S. Lin, *An Efficient Procedure for Partitioning Graphs*, Bell Systems Technical Journal, Vol. 49, pp. 291-307, 1970.

[24] A. V. Knyazev, *Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method*, SIAM Journal on Scientific Computing, Vol. 23, pp. 517-541, 2001.

[25] A. V. Knyazev, *Preconditioned Eigensolvers - an Oxymoron?*, ETNA, Vol. 7, pp. 104-123, 1998.

[26] J. Leskoveca, K. J. Langb, A. Dasguptab and M. W. Mahoneya, *Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters*, Internet Mathematics, Vol. 6, pp. 29-123, 2009.

[27] S. P. Lloyd, *Least Square Quantization in PCM*, IEEE Trans. Information Theory, Vol. 28, pp. 129-137, 1982.

[28] U. von Luxburg, *A Tutorial on Spectral Clustering*, Technical Report No. TR-149, Max Planck Institute, 2007.

[29] M. Naumov, *Preconditioned Block-Iterative Methods on GPUs*, Proc. International Association of Applied Mathematics and Mechanics, PAMM, Vol. 12, 11-14, 2012.

[30] M. Naumov, P. Castonguay and J. Cohen, *Parallel Graph Coloring with Applications to the Incomplete-LU Factorization on the GPU*, NVIDIA Technical Report, NVR-2015-001, 2015.

[31] A. Y. Ng, M. I. Jordan and Y. Weiss, *On Spectral Clustering: Analysis and an Algorithm*, Advances in Neural Information Processing Systems, pp. 849-856, 2001.

[32] A. Pothen, H. D. Simon and K. Liou, *Partitioning Sparse Matrices with Eigenvectors of Graphs*, SIAM Journal on Matrix Analysis and Applications, Vol. 11, pp. 430-452, 1990.

[33] D. Pelleg and A. Moore, *X-means: Extending K-means with Efficient Estimation of the Number of Clusters*, Proc. 17th International Conference on Machine Learning, pp. 727-734, 2000.

[34] A. H. Sameh and J. Wisniewski, *A Trace Minimization Algorithm for the Generalized Eigenvalue Problem*, SIAM Journal on Numerical Analysis, Vol. 19, pp. 1243-1259, 1982.

[35] H. D. Simon, *Partitioning of Unstructured Problems for Parallel Processing*, Computing Systems in Engineering, Vol. 2, pp. 135-148, 1991.

[36] J. Shi and J. Malik, *Normalized Cuts and Image Segmentation*, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 22, 2000.

[37] G. L. G. Sleijpen and H. A. Van der Vorst, *A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems*, SIAM Review, Vol. 42, pp. 267-293, 2000.

[38] D. A. Spielman, *Spectral Graph Theory*, Lecture Notes (Unpublished), Yale, CT, 2009.

[39] D. A. Spielman and S. Teng, *Spectral Partitioning Works: Planar Graphs and Finite Element Meshes*, Linear Algebra and its Applications, Vol. 421, pp. 284-305, 2007.

[40] D. A. Spielman, *Spectral Graph Theory*, Combinatorial and Scientific Computing, Chapman and Hall/CRC Press, 2011.

[41] G. W. Stewart, *Matrix Algorithms: Eigensystems*, SIAM, Philadelphia, PA, 2001.

[42] G. W. Stewart, *Accelerating the Orthogonal Iteration for the Eigenvectors of a Hermitian Matrix*, Numerische Mathematik, Vol. 13, pp. 362-376, 1969.

[43] D. F. Watson, *Computing the N-dimensional Delaunay Tessellation with Application to Voronoi Polytopes*, The Computer Journal, Vol. 24, pp. 167-172, 1981.

[44] Y. Wei and C. Cheng, *Ratio Cut Partitioning for Hierarchical Designs*, IEEE Trans. Computer-Aided Design, Vol. 10, pp. 911-921, 1991.

[45] Nvidia, *CUDA Toolkit*, http://developer.nvidia.com/cuda-downloads

[46] The Georgia Institute of Technology, Center for Discrete Mathematics and Theoretical Computer Science Collection (DIMACS10), http://www.cc.gatech.edu/dimacs10/downloads.shtml

[47] The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/

9 Appendix - Numerical Experiments

#	Matrix	m,n	nnz	nnz/n	max $d(i)$	Application
1.	vsp_barth5_1Ksep ...	32212	203610	6.320	22	Star Mixtures
2.	vsp_bcsstk30_500 ...	58348	4033156	69.12	219	Star Mixtures
3.	vsp_befref_fxm_2 ...	14109	196448	13.92	1531	Star Mixtures
4.	vsp_bump2_e18_aa ...	56438	601602	10.65	604	Star Mixtures
5.	vsp_c-30_data_da ...	11023	124368	11.28	2109	Star Mixtures
6.	vsp_c-60_data_ct ...	85830	482160	5.617	2207	Star Mixtures
7.	vsp_data_and_sey ...	9167	111732	12.18	229	Star Mixtures
8.	vsp_finan512_sca ...	139752	1104040	7.899	669	Star Mixtures
9.	vsp_mod2_pgp2_sl ...	101364	778736	7.682	1901	Star Mixtures
10.	vsp_model1_crew1 ...	45101	379952	8.424	17663	Star Mixtures
11.	vsp_msc10848_300 ...	21996	2442056	111.0	722	Star Mixtures
12.	vsp_p0291_seymou ...	10498	107736	10.26	229	Star Mixtures
13.	vsp_sctap1-2b_an ...	40174	281662	7.011	1714	Star Mixtures
14.	vsp_south31_slpt ...	39668	379828	9.575	17663	Star Mixtures
15.	vsp_vibrobox_sca ...	77328	871172	11.26	669	Star Mixtures
16.	citationCiteseer	268495	2313294	8.615	1318	Citations
17.	coAuthorsCiteseer	227320	1628268	7.162	1372	Citations
18.	coAuthorsDBLP	299067	1955352	6.538	336	Citations
19.	coPapersCiteseer	434102	32073440	73.88	1188	Citations
20.	coPapersDBLP	540486	30491458	56.41	3299	Citations
21.	G_n_pin_pout	100000	1002396	10.02	25	Clustering
22.	preferentialAttachment	100000	999970	9.999	983	Clustering
23.	smallworld	100000	999996	9.999	17	Clustering
24.	uk	4824	13674	2.834	3	Clustering
25.	belgium_osm	1441295	3099940	2.150	10	Street Network
26.	luxembourg_osm	114599	239332	2.088	6	Street Network
27.	ca2010	710145	3489366	4.913	141	U.S. Census
28.	in2010	267071	1281716	4.799	53	U.S. Census
29.	auto	448695	6629222	14.77	37	Numerical Sim.
30.	wing_nodal	10937	150976	13.80	28	Numerical Sim.
31.	fe_tooth	78136	905182	11.58	39	Finite Element
32.	fe_rotor	99617	1324862	13.29	125	Finite Element

Table 1: A sample of graphs from `DIMACS10` collection

	CHACO			Spectral-Lanczos-K-means				
#	Time (s)	Edge -cut	Cost $\tilde{\rho}$	Time (s)	Edge -cut	Cost $\tilde{\rho}$	Lanczos It.	K-means It.
1.	0.099	9275	1.152	0.306	58287	7.238	380	2
2.	2.952	2852	0.196	0.831	617	0.042	512	2
3.	0.146	3920	1.111	0.390	3231	4.460	512	6
4.	0.660	54943	3.894	0.286	35689	19.70	380	2
5.	0.150	2827	1.026	0.270	12027	4.523	512	2
6.	0.740	20255	0.944	0.366	14982	0.765	512	2
7.	0.098	13697	5.977	0.271	20296	8.857	512	2
8.	2.049	35216	1.008	0.506	19836	0.883	512	2
9.	1.521	54886	2.166	0.433	4050	0.318	512	2
10.	0.801	65054	5.770	0.330	608	608.0	512	2
11.	0.770	743	0.135	0.410	742	0.135	380	2
12.	0.065	8300	3.163	0.206	833	0.353	380	2
13.	0.422	34546	3.440	0.419	5342	0.829	512	2
14.	0.748	79541	8.021	0.326	85825	8.670	512	3
15.	0.600	126433	6.540	0.337	6239	1.699	380	2
16.	6.457	328610	4.896	0.833	341844	5.244	512	2
17.	4.291	143009	2.516	0.592	105181	1.877	512	2
18.	5.672	237985	3.183	0.762	93413	1.747	512	3
19.	55.69	4.1e+6	38.30	7.112	255724	4.142	512	2
20.	49.40	5.0e+6	37.46	7.775	1.3e+6	12.70	512	2
21.	0.228	211384	8.455	0.480	0	0.000	512	7
22.	2.074	171700	6.868	0.473	212956	8.518	512	2
23.	0.259	82392	3.296	0.127	274832	10.99	128	2
24.	0.008	98	0.081	0.139	28	0.028	512	2
25.	4.655	13331	0.037	1.779	7043	0.020	512	2
26.	0.163	1076	0.038	0.446	465	0.016	512	2
27.	†	†	†	1.391	2.4e+9	13903.	512	2
28.	†	†	†	0.634	4.4e+8	6672.	512	2
29.	6.671	11364	0.101	2.027	11246	0.101	512	2
30.	0.110	1067	0.069	0.423	906	0.090	512	2
31.	1.786	2439	0.098	0.524	2702	0.112	512	2
32.	0.569	4983	0.255	0.433	4312	0.221	512	2

Table 2: Results for graph partitioning into 2 partitions (part I)

	ILU0	Spectral-LOBPCG(ILU0)-K-means					Spectral-LOBPCG(ILU0)-K-means				
#	Time (s)	Time (s)	Edge -cut	Cost $\tilde{\rho}$	LOBPCG It.	K-means It.	Time (s)	Edge -cut	Cost $\tilde{\eta}$	LOBPCG It.	K-means It.
1.	0.014	0.131	6370	1.160	39	1	0.094	5269	1.039	27	1
2.	†	†	†	†	†	†	†	†	†	†	†
3.	1.803	0.077	1	0.500	13	1	0.056	3640	1.033	9	1
4.	0.351	0.176	27707	3.764	25	1	0.056	6545	0.790	7	1
5.	0.569	0.173	670	0.989	45	1	0.092	807	0.794	23	1
6.	†	†	†	†	†	†	†	†	†	†	†
7.	0.056	0.090	1802	3.040	29	1	0.061	5984	3.324	19	1
8.	0.227	0.292	18458	1.353	27	1	0.314	22167	0.944	29	1
9.	†	†	†	†	†	†	†	†	†	†	†
10.	17.44	0.284	5	1.667	37	1	0.150	18030	2.806	19	1
11.	1.408	0.398	742	0.135	31	1	0.349	743	0.135	27	1
12.	0.061	0.058	833	0.353	17	1	0.040	880	0.371	11	1
13.	0.598	0.109	4899	0.783	19	1	0.057	3295	0.558	9	1
14.	†	†	†	†	†	†	†	†	†	†	†
15.	0.230	0.161	6022	1.643	19	1	0.113	11072	2.802	13	1
16.	0.688	0.503	33	0.243	23	1	0.294	37	0.349	13	1
17.	0.263	0.144	31	0.419	7	1	0.072	11	0.099	3	1
18.	0.259	0.270	19	0.250	11	1	0.179	28	0.424	7	1
19.	52.93	1.546	2	0.061	9	1	1.218	50	0.300	7	1
20.	13.56	1.106	1	0.053	9	1	0.878	7	0.066	7	1
21.	†	†	†	†	†	†	†	†	†	†	†
22.	†	†	†	†	†	†	†	†	†	†	†
23.	0.029	0.184	1508	3.702	25	1	0.171	1780	3.522	23	1
24.	0.006	0.059	87	0.072	33	1	0.043	82	0.068	23	1
25.	0.116	1.331	80	0.071	19	1	1.334	218	0.067	19	1
26.	0.020	0.167	1565	0.055	23	1	0.126	1868	0.065	17	1
27.	0.098	19.07	5.5e+7	325.2	511	1	7.247	9.7e+7	860.0	193	1
28.	0.043	7.719	3.6e+7	541.2	511	1	7.751	3.5e+7	531.3	511	1
29.	0.136	2.794	9972	0.092	79	1	1.191	35545	0.323	33	1
30.	0.015	0.153	1018	0.073	33	1	0.110	1854	0.168	23	1
31.	0.039	0.384	2516	0.102	47	1	0.227	9052	0.374	27	1
32.	0.027	0.203	4058	0.208	31	1	0.166	5050	0.260	25	1

Table 3: Results for graph partitioning into 2 partitions (part II)

	METIS			Spectral-Lanczos-K-means				
#	Time (s)	Edge -cut	Cost $\tilde{\eta}$	Time (s)	Edge -cut	Cost $\tilde{\eta}$	Lanczos It.	K-means It.
1.	0.05	12200	23.49	0.58	18287	25.85	512	16
2.	0.12	154850	164.3	1.00	100712	100.1	512	9
3.	0.15	65072	283.1	0.46	34590	131.6	512	12
4.	0.52	160453	175.4	0.61	78313	81.01	512	13
5.	0.05	23364	129.2	0.35	51633	1005.	512	16
6.	0.10	22697	16.47	0.71	205070	2096.	512	13
7.	0.08	27489	183.6	0.31	4672	64.56	512	9
8.	0.35	87194	38.84	1.18	250624	87.57	512	16
9.	0.66	125259	76.87	0.92	249934	132.8	512	16
10.	0.35	105354	143.5	0.28	165638	43913.	275	6
11.	0.09	183525	517.5	0.61	86944	185.4	512	7
12.	0.10	30360	178.1	0.35	4111	44.62	512	16
13.	0.24	60059	92.38	0.62	123874	425.7	512	16
14.	0.31	100458	155.4	0.31	154341	46838.	153	16
15.	0.30	149825	120.9	0.81	163863	89.37	512	16
16.	0.59	191124	44.28	2.12	556370	110.4	512	16
17.	0.28	73750	20.10	1.30	615237	1795.	512	10
18.	0.55	150532	31.19	2.11	205389	42.70	512	15
19.	1.33	1.1e+6	170.5	9.19	232438	123.6	512	16
20.	1.96	2.0e+6	232.4	9.58	1.0e+6	296.8	512	11
21.	0.87	241447	149.6	0.55	44	43.00	512	2
22.	1.03	339874	210.7	0.89	370696	184.6	512	14
23.	0.28	120983	75.01	0.92	192501	80.26	512	16
24.	0.01	305	3.91	0.17	340	4.33	512	8
25.	0.60	1117	0.05	8.19	239561	10.09	512	15
26.	0.04	284	0.15	1.00	14566	7.76	512	16
27.	0.38	6.4e+8	56549.	4.75	3.E+10	3534800.	512	16
28.	0.12	3.6e+8	85405.	1.62	5.9e+9	1349810.	512	12
29.	0.38	128156	17.70	4.17	179382	25.25	512	16
30.	0.04	6492	6.49	0.73	7524	7.37	512	16
31.	0.08	34122	21.21	1.01	59253	36.39	512	16
32.	0.06	28119	22.28	0.80	30298	24.74	512	15

Table 4: Results for graph partitioning into 31 partitions (part I)

	Spectral-LOBPCG-K-means					ILU0	Spectral-LOBPCG(ILU0)-K-means				
#	Time (s)	Edge -cut	Cost $\tilde{\eta}$	LOBPCG It.	K-means It.	Time (s)	Time (s)	Edge -cut	Cost $\tilde{\eta}$	LOBPCG It.	K-means It.
1.	1.50	7308	21.31	79	1	0.014	0.59	7152	21.27	25	1
2.	6.99	122688	138.28	101	1	†	†	†	†	†	†
3.	5.70	16774	117.35	495	1	1.803	0.36	16874	108.9	17	1
4.	16.03	87833	94.41	493	1	0.351	1.48	87290	92.71	31	1
5.	5.07	3934	30.68	511	1	0.569	0.26	3985	30.57	17	1
6.	8.85	12149	13.22	201	1	†	†	†	†	†	†
7.	0.94	4283	69.23	111	1	0.056	0.17	4307	72.81	13	1
8.	20.92	21824	15.88	285	1	0.227	2.49	22554	16.60	25	1
9.	28.21	18468	24.22	511	1	†	†	†	†	†	†
10.	15.01	20533	30.10	511	1	17.44	0.78	10954	27.56	17	1
11.	1.14	167285	511.8	53	1	1.408	0.74	171104	521.2	11	1
12.	1.64	2435	35.62	181	1	0.061	0.26	2180	35.89	19	1
13.	8.58	45188	68.26	363	1	0.598	0.72	43671	68.25	21	1
14.	13.67	5097	28.00	511	1	†	†	†	†	†	†
15.	7.36	48869	193.5	169	1	0.230	1.35	73905	183.5	21	1
16.	80.48	767	2.05	511	1	0.688	4.30	168	2.02	19	1
17.	33.31	472	4.10	275	1	0.263	1.65	259	4.74	9	1
18.	41.88	435	4.90	263	1	0.259	3.04	223	4.03	13	1
19.	297.9	535	5.71	453	1	52.93	21.99	203	2.27	15	1
20.	374.0	353	6.65	511	1	13.56	21.03	110	2.50	15	1
21.	4.39	269718	163.19	75	1	†	†	†	†	†	†
22.	20.81	343094	187.45	349	1	†	†	†	†	†	†
23.	3.50	111721	83.85	65	1	0.029	2.03	125925	80.12	29	1
24.	0.34	319	4.01	53	1	0.006	0.14	292	3.33	19	1
25.	40.53	64679	2.74	63	1	0.116	14.27	54751	2.11	19	1
26.	2.59	5300	2.69	47	1	0.020	1.33	2853	1.52	21	1
27.	177.7	1.3e+9	176497.	511	1	0.098	111.4	5.6e+8	37987.	261	1
28.	65.86	4.8e+8	100834.	511	1	0.043	38.01	3.7e+8	84531.	243	1
29.	27.46	134921	18.71	89	1	0.136	15.58	135212	19.13	35	1
30.	2.00	8055	8.00	61	1	0.015	1.05	7848	7.82	27	1
31.	5.62	37287	23.32	89	1	0.039	2.02	37887	23.96	23	1
32.	4.44	28726	22.27	95	1	0.027	1.69	28920	22.03	27	1

Table 5: Results for graph partitioning into 31 partitions (part II)