Fast Spectral Graph Partitioning with a Randomized Eigensolver

Heliezer J. D. Espinoza
Department of Mathematics
Cal Poly Pomona
Pomona, CA, USA
dhespinoza@cpp.edu

Jennifer A. Loe
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, USA
jloe@sandia.gov

Erik G. Boman
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, USA
egboman@sandia.gov

***Abstract*—A known problem in parallel computing is how to partition a matrix such that work can be distributed among several processors efficiently. One technique to do this is spectral graph partitioning, which uses the eigenvectors of the graph Laplacian to determine the optimal way for the matrix to be divided. This partitioning method is particularly suited for parallelization, specifically for GPUs, as it mainly relies on linear algebra operations. However, this increased parallelism may come at the cost of accuracy.**

In this work, we present a novel improvement to spectral graph partitioning by replacing the exact eigensolver (LOBPCG) with a randomized eigensolver roughly an order of magnitude faster. While the accuracy of the eigensolver is typically worse, we show that for graph partitioning this is sufficient. Our algorithm is implemented in the Sphynx spectral graph partitioner, contained in the Zoltan2 package of Trilinos. Results show this randomized method in general gives a substantial speedup with minimal loss in the quality of the edge cut. In some cases the randomized method even gives slightly better edge cuts than the LOBPCG eigensolver.

I. Introduction

Graph partitioning is a classic problem in computer science. We focus on the *balanced partitioning* problem, which aims to minimize the edge cut subject to the parts having (approximately) the same size. Since the problem is NP-hard, work in this area [1] has led to a wide variety of heuristic algorithms. Perhaps the most popular algorithm is the multilevel algorithm [2], which has been implemented in software such as Chaco, Metis, Scotch, and KaHip. Although multilevel methods are fast and generally give good quality, they have a couple of drawbacks. First, they require a lot of memory (all the levels must be kept in memory). Second, they are hard to parallelize, especially on GPUs. The spectral partitioning algorithm [3] addresses both of these concerns: it is single-level, so requires less memory, and it is suitable for GPU because it relies on linear algebra. However, it is often slower and generally gives poorer cut quality. Our goal here is to improve the speed of spectral partitioning using randomized linear algebra. We will demonstrate the usefulness of our approach by modifying the Sphynx [4], [5] partitioner. We will show that the speed can be improved significantly, at little or no expense in quality. We believe this makes randomized spectral partitioning a viable option for some applications.

Sandia is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

Part of this work was performed while HJDE was an intern at Sandia.

Our approach is general and may be adapted to similar problems, such as graph clustering [6], community detection via modularity, and embeddings in machine learning.

A. Contributions

The main contributions of this paper are:

- A parallel implementation of a randomized spectral graph partitioner that runs on multiple GPUs. We build on the Sphynx partitioner in Trilinos/Zoltan2.
- An empirical study of the time and cut quality for large, highly irregular graphs, such as web graphs and circuit simulation. We also give comparison to LOBPCG (a standard eigensolver).
- Empirical evaluation of the time vs. quality trade-off in using q steps of the power method (subspace iteration) in the randomized eigensolver, for various q.

B. Related Work

Spectral methods have long been used both for (balanced) partitioning and clustering. Using a randomized method was first proposed in [7] and later described and analyzed [8], but there was only one experiment for a single eigenvector on a small example. We use a more modern randomized method and compare against a state-of-the-art eigensolver (LOBPCG) on large graphs. We also compute multiple eigenvectors to partition into $k > 2$ parts. Espinoza [9] outlined our algorithm but used a combination of Matlab and Sphynx to produce results. Here, we present an HPC implementation based on Trilinos [10] that is performance portable via Kokkos so it can run on multiple platforms (CPU and GPUs).

Graph partitioning is a well-studied problem [1]. Generally, connectivity-based methods that use the graph structure give best quality, but may be slow. Geometric methods that only use the geometry and not the connectivity are fast but typically yield lower quality cuts. Our focus is on irregular graphs from web networks, social networks, etc., where there is no geometry. In that case, the multilevel method [2] and the spectral method are the most effective ones.

We focus on spectral partitioning since it is more suitable for GPUs. Naumov [11] first implemented spectral partitioning on GPU, but the focus was on clustering, so there is no option for balanced partitioning. Also, the code is limited to a single GPU. Acer et al [4], [5] developed the Sphynx partitioner in Trilinos/Zoltan2. That code base was chosen for the current work because it both supports balanced partitioning and runs on multiple GPUs.

Randomized numerical linear algebra [12], [13] has gained much popularity in the last decade or two. The key advantage is that randomized methods can quickly find approximate solutions to some linear algebra problems, such as low-rank approximations, range-space, and SVD. The randomization allows one to approximate certain steps of the algorithm while saving the expense of computing those values exactly.

II. Eigensolvers and Spectral Graph Partitioning

A. Spectral partitioning

Spectral graph partitioning is based on computing a few eigenvectors of the *graph Laplacian*. Given a graph $G = (V, E)$, the *combinatorial* Laplacian is

$$L = D - A,$$

where A is the adjacency matrix and D is a diagonal matrix of vertex degrees. For weighted graphs, the off-diagonals will correspond to edge weights. A closely related variation is the *normalized* Laplacian:

$$L_N = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}.$$

The edge cut in graph partitioning is then $\frac{1}{4}x^T Lx$, where each $x_i \in \pm 1$. This gives an integer optimization problem, which is NP-hard. Donath et al. [14] observed that a relaxation of the integer optimization problem gives an eigenproblem, where the eigenvector of interest is the second smallest (aka the Fiedler vector [15]). This observation opened up practical (polynomial-time) algorithms. Pothen et. al [3] first studied the spectral method to partition graphs and sparse matrices. Later, this method has been extended [16] and implemented in software [17]–[19].

Spectral methods are also popular for the related problem of *graph clustering* [6], [7], [20]. In clustering, the parts do not need to be similar size; rather, the goal is to identify closely related information in the graph.

B. The Sphynx partitioner

We review the approach used in the Sphynx [4], [5] partitioner (a subpackage of Zoltan2 in Trilinos), into which our new randomized method is incorporated. Figure 1 illustrates the paritioning process at a high level: First, we will take a graph Laplacian (combinatorial or normalized) of a test graph/matrix. From there, an eigensolver is used to obtain approximate eigenvectors. The number of eigenvectors d required to get k parts is $d = log_2 k + 1$. Note that graph Laplacians have an eigenvector corresponding to the zero eigenvalue that is not used for partitioning (and thus discarded). For Figure 1, the graph is partitioned into 4 parts, so 3 eigenvectors are needed. Sphynx will then use the eigenvectors as coordinates to map the graph's vertices into (d - 1)-space. In the figure, Sphynx uses the 2nd and 3rd eigenvectors as coordinates to map the vertices into 2D space. Lastly, Sphynx will call the multi-jagged (MJ) algorithm in the Zoltan2 package to partition the graph into k parts (4 parts for this example figure).

The three main steps in Sphynx are summarized below:

1) Form the graph Laplacian L from the graph G.
2) Compute the $d + 1$ smallest eigenvectors of L.
3) Partition the vertices of G in $\mathbb{R}^d$ using the spectral embedding and a fast geometric method (Multi-jagged).

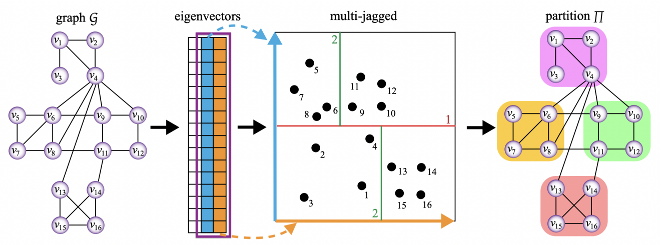

Fig. 1: Illustration of Sphynx partitioning a graph with 16 unit-weight vertices into 4 parts. [4]

C. LOBPCG Eigensolver

It is well-known that most of the computational time in spectral partitioning is spent in the eigensolver. Any eigensolver can be used, but LOBPCG [21] has been shown to work well on many graphs, so is generally preferred. LOBPCG has the added advantage it supports preconditioning, which is not possible with many other methods, such as, Lanczos or Krylov-Schur.

However, LOBPCG was designed to solve eigenproblems to high accuracy. For partitioning, fairly low accuracy is often sufficient [4]. We therefore propose to use a *randomized eigensolver*. Randomized eigensolvers are typically fast when low accuracy is sufficient, which is precisely our use case. For the experiments that follow, Sphynx utilizes the LOBPCG solver implemented in the Trilinos package Anasazi [22]. We also implemented a new prototype of the randomized eigensolver in Anasazi, which we used for comparison.

D. Randomized Eigensolver

There are several variations of randomized eigensolvers. Of the different techniques to solve the eigenvalue-eigenvector problem that we can utilize for our randomized method, we are most interested in a direct numerical method. The Rayleigh-Ritz method [23, p. 98] uses a projection of a smaller dense matrix to approximate the eigenvalues and eigenvectors of our original matrix. This method is of great interest because solving for the eigenvectors of a small projected matrix, rather than a much larger matrix, can significantly reduce our computation cost.

Building on the idea of the Rayleigh-Ritz method, Halko, et. al. [12] proposed a randomized projection to be used for eigensolvers and SVD. We adopt this method due to its simplicity. The method is detailed in Alg. 4.3 and 5.1 of [12]. (Their work does contain an example that applies the method to graph Laplacians, but it was not used for partitioning.) One caveat is that the method is intended to find the largest (rightmost) eigenvalues of the matrix, not the smallest (leftmost), as it uses a variation of subspace iteration (the power method). We address this by applying a simple spectral transformation to the matrix to find the desired eigenvectors.

For completeness, we describe our method in detail here. First recall that that the (symmetrically) normalized Laplacian is defined as $L_N = I - D^{-1/2}AD^{-1/2}$. We observe that L_N has ones on the diagonal, and by Gerschgorin's circle theorem, all the eigenvalues are in the interval $(0, 2)$. Let $\hat{L}$ be defined by

$$\hat{L} = 2I - L_N = I + D^{-1/2}AD^{-1/2}. \tag{1}$$

Observe that, like L_N, $\hat{L}$ is symmetric positive semidefinite. The largest eigenvalue of $\hat{L}$ is the smallest of L_N, and vice versa, so we apply our method to $\hat{L}$. Our randomized eigensolver is:

- **Part A:** Compute orthogonal basis Q of the range of $\hat{L}$.
  1) Create a random Gaussian matrix, $\Omega \in \mathbb{R}^{nxl}$, where l is chosen to be greater than the number of desired eigenvectors.
  2) Form $Y = \hat{L}^q\Omega$, where q is a small integer (input parameter)
  3) Compute the tall and skinny QR factors: $QR = Y$.
- **Part B:** Solve projected (small) eigenvalue system; project back.
  1) Compute projection $B = Q^T\hat{L}Q$.
  2) Solve eigenproblem $B = V\Lambda V^T$.
  3) Project back to large system $U = QV$.

Thus, the columns of U approximate eigenvectors of $\hat{L}$. Part A corresponds to *Algorithm 4.3: Randomized Power Iteration* from Halko, et. al. [12], and Part B corresponds to *Algorithm 5.1 - Direct SVD.*

Note the parameter l must be larger than the desired number of eigenvectors d, which again depends on the number of parts. The difference $l - d$ is known as the oversampling parameter, and is typically a small integer in the range 2 to 5.

We implement a preliminary version of the randomized eigensolver in Anasazi to use in conjunction with Sphynx. The eigensolver implementation uses MPI distributed parallelism and can run on GPUs when available (via Kokkos and KokkosKernels).

Complexity: First suppose L is dense (which is rarely the case since most graphs are sparse). Then the most expensive part is actually the matrix multiplication $\hat{L}^q\Omega$, which requires $O(qn^2l)$ flops. The QR factorization (orthogonalization) is $O(nl^2)$, which is less since $l < n$.

Now suppose L is sparse with nnz nonzeros. Then a single sparse matrix-vector multiply is $O(nnz)$ flops, so the dominating matrix multiply is now $O(q(nnz)l)$. Since Y remains dense, it is possible the tall skinny QR cost could dominate for very sparse Laplacians.

An open question is what is a good choice of q. Although there are bounds on the convergence rate of subspace iteration, this does not tell us much about the partitioning quality. Therefore, we perform an empirical study.

III. Experiments

Our experiments will compare the Sphynx spectral partitioner using two different eigensolvers, LOBPCG and the randomized method, to obtain approximate eigenvectors for the graph Laplacians L_N of the test matrices. We also compare Sphynx on GPU with XtraPuLP [24], a CPU-only partitioner known to work well with irregular graphs. Prior to computing the graph Laplacian of our test matrices, we performed preprocessing calculations by following the procedure laid out in Acer, et. al [4]. This was done by symmetrizing the matrix, setting all nonzero entries equal to 1 (equivalent to assuming that all edge weights are one), and extracting the largest connected component. The (normalized) graph Laplacian L_N of the preprocessed test matrices was then computed. It should also be noted that obtaining the largest connected component is important for this experimental run as Sphynx cannot handle multiple connected components.

All experiments are run on a cluster equipped with IBM Power9 processors (dual-socket, 20 cores per socket) that have dual NVIDIA V100 GPUs per socket. In other words, each compute node has 40 cores and 4 V100 GPUs. The Sphynx code is built using OpenMPI 4.1.1 and CUDA 11.2.2. For calling XtraPuLP from Zoltan2, we use OpenMPI 4.1.1 with OpenMP enabled. Unless stated otherwise, Sphynx experiments are run on a single node using 4 MPI ranks (one rank for each GPU). The XtraPuLP experiments are also run on one node with 4 MPI ranks, where each MPI rank gets 10 OpenMP threads, thus using all CPU cores available. We claim this gives as fair comparison of Sphynx and XtraPuLP performance as possible, given that XtraPuLP is not GPU-enabled.

A. Graphs/Matrices

Results in [9] indicate that spectral partitioning with the randomized eigensolver works better for irregular graphs (those with a high ratio of max/average vertex degree) than regular graphs (which often come from meshes). Therefore we focus our tests on 14 irregular graphs available in the SuiteSparse collection [25], mostly coming from web graphs, social networks, and circuit simulation. These matrices and their properties are listed in Table I.

B. Randomized Sphynx with increasing q

We partition each graph into 64 parts, setting Sphynx to balance by rows. Thus, for spectral partitioning, we need to compute $\log_2(64) + 1 = 7$ eigenvectors (including the trivial vector of all ones). Since results from all the tested partitioners are non-deterministic, results shown use the median edgecut of five runs and the corresponding timing. Timings for Sphynx include the eigenvector solve and the multi-jagged partitioning

TABLE I: Test matrices (graphs) used for the experiments. Vertices correspond to rows/columns in the matrix.

SuiteSparse Matrices			degree	
name	vertices	edges	max	average
hollywood-2009	1,069,126	113,682,432	11,468	106
sx-stackoverflow	2,601,977	36,233,450	44,065	22
FullChip	2,986,914	26,621,906	2,312,481	9
com-Orkut	3,072,441	237,442,607	33,314	77
wikipedia-2007	3,512,462	88,261,228	187,672	25
cit-Patents	3,764,117	36,787,597	794	10
com-LiveJournal	3,997,962	73,360,340	14,816	18
circuit5M	5,555,791	59,519,031	1,290,501	11
wb-edu	8,863,287	97,233,789	25,782	11
uk-2005	39,252,879	1,602,132,663	1,776,859	41
it-2004	41,290,577	2,096,240,367	1,326,745	51
twitter7	41,652,230	2,446,678,322	2,997,488	59
com-Friendster	65,608,366	3,677,742,636	5,215	56
webbase-2001	118,142,155	1,019,903,190	816,127	15

phase. Timings for pre-processing operations (symmetrizing and finding the largest connected component) are not included.

First we study the Randomized Sphynx graph partitioner using increasing values of q, the number of times we multiply by the matrix $\hat{L}$. We use a block size of $\ell = 10$; larger block sizes did not seem to give any significant improvement in the edgecuts. Table II shows edgecuts and timings for $q = 1, 2, 4, 8, 16$. Recall that smaller edgecuts are preferable in graph partitioning. Typically, edgecuts decrease with larger values of q; some examples are plotted in Figure 2. This is expected because q corresponds to the number of iterations in subspace iteration, so larger q values will give more accurate eigenvector approximations with the randomized eigensolver. On average, the edgecuts decrease by over 40% from $q = 1$ to $q = 16$ and by over 30% from $q = 8$ to $q = 16$. However, the improvements lessen as q increases, eventually reaching a plateau. Further, large q values may lead to numerical instability and breakdown in the orthogonalization step. This happened for a small number of our test matrices with $q = 32$; thus, we stopped our experiments at $q = 16$ in this work. We should be able to use higher q for those matrices if we added re-orthogonalization to the randomized eigensolver.

We observe that timings only increase slightly as q increases. This is expected due to additional sparse matrix-vector products (SpMVs) in the randomized eigensolver. More specifically, timings from $q = 1$ to $q = 16$ increase, on average, by a little over 2 times. From $q = 8$ to $q = 16$, the timings increase, on average, by only about 35%. Improvements in edgecut with higher q seem to strongly outweigh the small increase in solve time.

C. *Randomized Sphynx vs known methods*

Next we compare the Randomized Sphynx partitioner with two other existing methods: a) Sphynx using an LOBPCG solve and b) the XtraPuLP partitioner [26]. For Sphynx with the LOBPCG solve, we use a Jacobi preconditioner (which is actually equivalent to no preconditioner in the normalized case), and we run LOBPCG until the eigenvectors converge to a tolerance of $1e-2$. We compare with XtraPuLP because it is a non-spectral partitioner known to be effective with irregular graphs.

Timings and edgecuts for both methods are shown in Table III. For comparison, we also include results for the randomized partitioner using $q = 16$, with edgecuts and timings normalized against those of Sphynx with LOBPCG and XtraPuLP. On the whole, the randomized method is typically much faster than the other two methods but provides a worse edgecut than XtraPuLP.

The cut quality comparison between Sphynx with LOBPCG versus the randomized eigensolver is interesting. In 6 of the test cases, the randomized solver actually gives better edge cuts, while in 8 of the cases, it did worse. The two best edgecuts are for the matrices FullChip and circuit5M, which interestingly are both circuit simulation matrices. Both edgecuts are about 21% smaller than those of LOBPCG. Three matrices have edgecuts that are less than 5% larger than those of Sphynx with LOBPCG. This is not a significant difference. Table IV lists eigenvalues and corresponding residuals for both the LOBPCG solves and the randomized eigensolver. Recall that LOBPCG was set to stop when all eigenvalues had converged to a residual of at least $1e-2$. The table shows that many eigenvalues converged to less than that. With $q = 16$, however, most of the residuals are slightly higher than $1e-2$. This demonstrates that in many cases quite low eigenvalue accuracy is sufficient to achieve good partitioning.

As for timings, the randomized partitioner (with $q = 16$) was much faster than traditional Sphynx for every graph tested. The most dramatic improvement is seen with webbase-2001, where the randomized partitioner ($q = 16$) is 658 times faster than Sphynx with LOBPCG. (Though this does come at a price of a twice as large an edgecut.) Even the solve with the smallest improvement, with com-Orkut, is still roughly two times faster than with traditional Sphynx. (This time the faster partitioner gives a comparable edgecut.) On average, the randomized solver is about 65 times faster than Sphynx with LOBPCG.

We now compare the randomized partitioner with results from XtraPuLP. All edgecuts from PuLP were smaller than corresponding edgecuts from Randomized Sphynx. Eight of the fifteen randomized edgecuts were less than two times the size of the PuLP edgecuts. The worst case is with the graph it-2004 which had an edgecut 3.7X larger with Randomized Sphynx than with XtraPuLP. At this time, we have not identified a reason for the large discrepancy in edgecuts. We leave this as a subject for future research.

Again, the randomized solver proves to typically be faster than other methods. Eleven of the fourteen randomized timings were faster than XtraPuLP. Our comparison used four GPUs (and 4 MPI ranks) for Randomized Sphynx, versus 40 cores (4 MPI ranks with 10 threads each) for XtraPuLP.

IV. Conclusions and Future Work

We have presented and evaluated a randomized eigensolver for spectral partitioning. We showed the randomized method is roughly an order of magnitude faster than the LOBPCG

Matrix	q=1 Edgecut	q=1 Time	q=2 Edgecut	q=2 Time	q=4 Edgecut	q=4 Time	q=8 Edgecut	q=8 Time	q=16 Edgecut	q=16 Time
hollywood-2009	1.107e8	.9708	1.104e8	1.010	1.107e8	1.082	9.400e7	1.224	**6.980e7**	**1.512**
sx-stackoverflow	5.532e7	1.446	5.511e7	1.529	5.473e7	1.679	5.388e7	2.020	**5.022e7**	**2.683**
FullChip	2.289e7	5.589	2.220e7	7.147	2.051e7	10.33	1.815e7	16.67	**1.620e7**	**29.28**
com-Orkut	2.304e8	2.134	2.300e8	2.334	2.285e8	2.626	2.158e8	3.268	**1.647e8**	**4.577**
wikipedia-20070206	8.316e7	1.923	8.280e7	2.088	8.189e7	2.463	7.923e7	3.128	**7.006e7**	**4.547**
cit-Patents	3.200e7	1.635	3.136e7	1.648	2.987e7	1.752	2.705e7	1.955	**2.307e7**	**2.315**
com-LiveJournal	6.777e7	1.813	6.698e7	1.873	6.439e7	2.005	5.669e7	2.342	**4.829e7**	**2.903**
circuit5M	5.295e7	3.530	5.275e7	4.115	5.215e7	5.129	4.816e7	7.293	**3.218e7**	**11.75**
wb-edu	8.593e7	2.682	8.395e7	2.843	7.471e7	2.904	4.257e7	2.977	**2.113e7**	**3.316**
uk-2005	1.535e9	14.54	1.525e9	15.40	1.432e9	17.17	7.168e8	20.59	**3.545e8**	**27.23**
it-2004	2.020e9	33.79	2.013e9	35.58	1.961e9	38.48	1.282e9	40.23	**4.161e8**	**45.93**
twitter7	2.365e9	435.7	2.363e9	473.8	2.356e9	559.1	2.331e9	766.8	**1.864e9**	**1,040**
com-Friendster	3.550e9	800.7	3.544e9	934.3	3.527e9	1,262	3.484e9	2,059	**3.067e9**	**3,741**
webbase-2001	1.651e9	74.45	1.627e9	74.93	1.492e9	75.06	8.175e8	79.10	**3.329e8**	**84.84**

TABLE II: Results for partitioning into $k = 64$ parts with Randomized Spectral Partitioning with various q values ($l = 10$). Timings are in seconds. The best edgecut for each matrix and corresponding timings are in bold.

	LOBPCG w/ Jacobi			q=16 w.r.t. LOBPCG		XtraPuLP		q=16 w.r.t XtraPuLP	
Matrix	Iters	Edgecut	Time	Edgecut	Time	Edgecut	Time	Edgecut	Time
hollywood-2009	155	6.840e7	8.260	1.020	0.183	6.027e7	6.486	1.158	0.233
sx-stackoverflow	68	5.071e7	6.455	**0.990**	**0.416**	3.864e7	10.99	1.300	0.244
FullChip	96	2.062e7	56.90	**0.785**	**0.515**	9.946e6	5.376	1.628	5.446
com-Orkut	44	1.629e8	7.619	1.011	0.601	1.227e8	23.90	1.343	0.192
wikipedia-20070206	95	7.074e7	12.38	**0.990**	**0.367**	4.338e7	19.24	1.615	0.236
cit-Patents	101	1.707e7	7.767	1.352	0.298	8.958e6	10.69	2.575	0.217
com-LiveJournal	140	4.984e7	12.37	**0.969**	**0.235**	2.917e7	13.55	1.655	0.214
circuit5M	144	4.110e7	35.88	**0.783**	**0.327**	1.936e7	15.44	1.662	0.761
wb-edu	103	6.335e6	12.02	3.335	0.276	4.390e6	11.22	4.813	0.296
uk-2005	179	1.802e8	176.7	1.967	0.154	1.515e8	82.50	2.340	0.330
it-2004	187	2.359e8	3,384	1.764	0.014	1.112e8	93.38	3.740	0.492
twitter7	284	2.100e9	36,860	**0.888**	**0.028**	1.695e9	595.4	1.100	1.747
com-Friendster	173	2.926e9	45,380	1.048	0.082	1.759e9	2,304	1.743	1.624
webbase-2001	137	1.541e8	55,800	2.161	0.002	1.086e8	243.8	3.067	0.348

TABLE III: Results for Spectral Partitioning into $k = 64$ parts using LOBPCG with Jacobi Preconditioning as eigensolver, and with Xtra-PuLP as partitioner. "Iters" indicates the number of iterations used by LOBPCG. Values are shown for the randomized partitioner with $q = 16$ and normalized with respect to LOBPCG and XtraPuLP. Edgecuts/timings from the randomized solver are divided by those of other methods. Edgecut values less than 1 (in bold) indicate the randomized solver performed better.

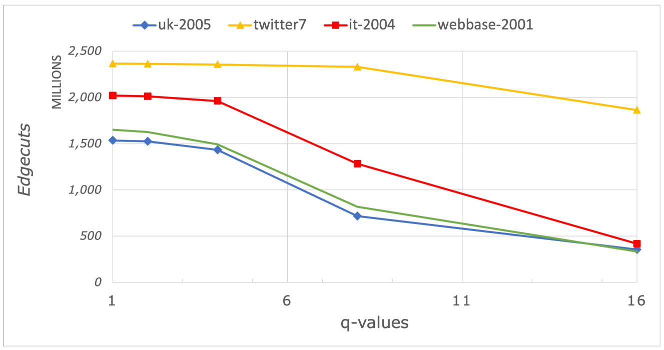

Fig. 2: Randomized eigensolver spectral partitioning edgecuts results, using the uk-2005, twitter7, it-2004, and webbase-2001 matrices. Values are taken from Table II. The x-axis q values corresponds to the equation $Y = A^q\Omega$, the second equation listed in the Part A of the randomized eigensolver method.

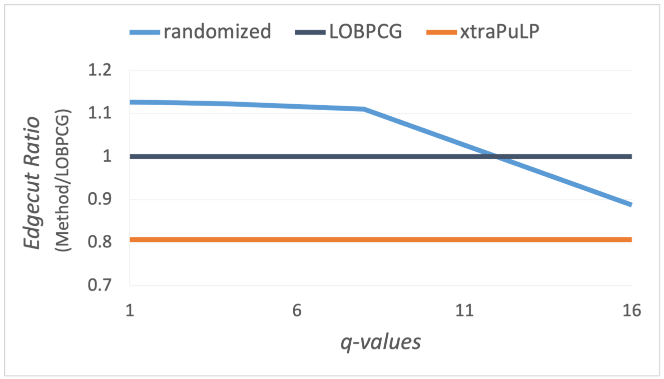

Fig. 3: Comparison of spectral partitioning edgecut results for the twitter7 matrix. Values are taken from Table II and III. The x-axis q values corresponds to the equation $Y = A^q\Omega$ The y-axis correspond to the ratio of an eigensolver's edgecut, (randomized, LOBPCG, or XtraPuLP) normalized by LOBPCG's values.

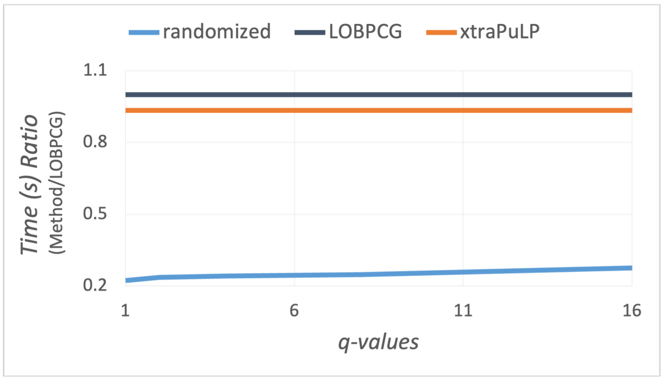

Fig. 4: Comparison of spectral partitioning timing results for the twitter7 matrix. Values are taken from Table II and III. The x-axis q values corresponds to the equation $Y = A^q\Omega$. The y-axis correspond to the ratio of an eigensolver's timings, (randomized, LOBPCG, or XtraPuLP) normalized by LOBPCG's values.

Matrix	j	LOBPCG w/ Jacobi λ_j	LOBPCG w/ Jacobi r_j	q=16 λ_j	q=16 r_j
wikipedia -20070206	1	-6.039E-15	3.48E-11	0.1286	5.832E-02
	2	3.720E-03	2.35E-05	0.1389	5.722E-02
	3	4.724E-03	7.14E-05	0.1418	5.735E-02
	4	6.210E-03	2.65E-04	0.1426	5.896E-02
	5	8.252E-03	1.52E-03	0.1432	5.768E-02
	6	1.023E-02	8.56E-03	0.1482	5.770E-02
	7	1.105E-02	1.32E-03	0.1519	5.893E-02
circuit5M	1	-1.674E-12	3.26E-09	0.0740	3.642E-02
	2	1.929E-04	1.66E-07	0.0777	3.671E-02
	3	1.929E-04	2.90E-08	0.0787	3.633E-02
	4	2.965E-04	1.31E-04	0.0802	3.815E-02
	5	4.261E-04	7.10E-03	0.0818	3.749E-02
	6	4.440E-04	6.77E-05	0.0824	3.784E-02
	7	6.136E-04	6.53E-04	0.0843	3.756E-02
twitter7	1	-6.395E-14	2.58E-09	0.1454	5.829E-02
	2	5.099E-04	1.83E-09	0.1480	5.934E-02
	3	5.105E-04	6.25E-08	0.1503	5.870E-02
	4	1.039E-03	2.84E-07	0.1508	5.882E-02
	5	1.099E-03	1.19E-06	0.1515	5.914E-02
	6	2.348E-03	2.37E-05	0.1541	5.819E-02
	7	2.755E-03	9.04E-03	0.1546	5.856E-02

TABLE IV: Eigenvalues and corresponding residuals of the six matrices that produced better edgecuts values for the randomized eigensolver than LOBPCG

eigensolver. While LOBPCG usually needs over 100 iterations to converge to our tolerance, the randomized solver (subspace iteration) usually gives good partition quality after only 16 matrix-vector multiplies. The accuracy of the eigenvalues is generally lower, which only slightly affects the cut quality of the partitioning in most cases. To improve the quality, we introduce a parameter q used in the randomized method. Generally, larger q gives better partitioning quality but takes more time. Empirically, we found $q = 16$ to be a good value. For web graphs and social networks we found the cut quality is similar to or worse than with LOBPCG. However, for circuit graphs, the randomized method can, surprisingly, give better cut quality than LOBPCG. This merits future study.

We believe our method is most useful in applications where partitioning time is a major concern, for example, simulations with dynamic load balancing.

There are several directions for future work:

- We only evaluated q values up to $q = 16$, due to numerical issues. With reorthogonalization (commonly used in subspace iteration), higher q values would be possible, and perhaps the edge cuts would improve further. It might be possible to automatically estimate a good q value from the R in the QR factorization.
- A natural way to improve our results is to add a refinement phase. Currently, the multi-jagged method only considers the geometry and does not directly try minimize the edge cut. A refinement phase would improve the edge cuts, but is hard to do in parallel, especially on GPU.
- We used the multi-jagged partitioner in Zoltan2 for the geometric partitioning in low dimension. We do not claim this is the best method. Any geometric partitioning method could be used here, such as, k-means.
- For parallel computation, we assumed the matrix is partitioned by rows. However, the sparse matrix-vector multiply (SpMV) will likely be faster for a 2D (nonzero-based) matrix decomposition [27]. This would speed up the spectral partitioner (both LOBPCG and the randomized eigensolver).
- Finally, we remark that our approach likely extends to spectral clustering (as opposed to partitioning). Since low-accuracy eigenvectors are sufficient for partitioning, we believe the same approach will work for clustering. Only the last phase (multi-jagged) would need to be changed. We leave this as future work, as clustering poses other challenges.

Acknowledgments

Thanks to Heather Switzer whose comments helped improve the paper. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, “Recent advances in graph partitioning,” *Algorithm engineering*, pp. 117–158, 2016.
[2] B. Hendrickson and R. Leland, “A multilevel algorithm for partitioning graphs,” in *Proc. Supercomputing ’95*. ACM, December 1995.
[3] A. Pothen, H. Simon, and K. Liou, “Partitioning sparse matrices with eigenvectors of graphs,” *SIAM J. Matrix Anal.*, vol. 11, no. 3, pp. 430–452, July 1990.
[4] S. Acer, E. G. Boman, and S. Rajamanickam, “SPHYNX: Spectral Partitioning for HYbrid aNd aXelerator-enabled systems,” in *Proc. Int’l. Parallel and Distributed Proc. Symp. Workshops (IPDPSW)*, 2020.
[5] S. Acer, E. G. Boman, C. A. Glusa, and S. Rajamanickam, “Sphynx: A parallel multi-gpu graph partitioner for distributed-memory systems,” *Parallel Computing*, vol. 106, p. 102769, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819121000272
[6] U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, pp. 395–416, 2007.

[7] R. Kannan, S. Vempala, and A. Vetta, "On clusterings: Good, bad and spectral," *J. ACM*, vol. 51, no. 3, p. 497–515, may 2004. [Online]. Available: https://doi.org/10.1145/990308.990313

[8] D. Cheng, R. Kannan, S. Vempala, and G. Wang, "A divide-and-merge methodology for clustering," vol. 31, no. 4, p. 196–205, 2006. [Online]. Available: https://doi.org/10.1145/1065167.1065192

[9] H. J. D. Espinoza, "Randomized methods for graph partitioning applications," Master's thesis, California State Polytechnic University, Pomona, 2022.

[10] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Trans. Database Syst.*, vol. 31, no. 3, p. 1499–1525, dec 2005. [Online]. Available: https://doi.org/10.1145/1089014.1089021

[11] M. Naumov and T. Moon, "Parallel spectral graph partitioning," NVIDIA tech. rep. NVR-2016-001, Tech. Rep., 2016.

[12] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011. [Online]. Available: https://doi.org/10.1137/090771806

[13] P.-G. Martinsson and J. A. Tropp, "Randomized numerical linear algebra: Foundations and algorithms," *Acta Numerica*, vol. 29, p. 403–572, 2020.

[14] W. Donath and A. Hoffman, "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices," *IBM Technical Disclosure Bulletin*, vol. 15, pp. 938–944, 1972.

[15] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Math. J.*, vol. 23, no. 98, pp. 298–305, 1973.

[16] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel computations," *SIAM Journal on Scientific Computing*, vol. 16, no. 2, pp. 452–469, 1995.

[17] S. Barnard and H. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience*, vol. 6, no. 2, pp. 101–117, 1994. [Online]. Available: https://doi.org/10.1002/cpe.4330060203

[18] H. D. Simon, A. Sohn, and R. Biswas, "HARP: A fast spectral partitioner," *J. of Parallel and Distr. Computing*, vol. 50, pp. 83–103, 1998.

[19] B. Hendrickson and R. Leland, "The Chaco users guide. version 1.0," Sandia National Laboratories Albuquerque, Tech. Rep., 1993.

[20] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[21] A. V. Knyazev, "Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517–541, 2001. [Online]. Available: https://doi.org/10.1137/S1064827500366124

[22] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist, "Anasazi software for the numerical solution of large-scale eigenvalue problems," *ACM Trans. Math. Softw.*, vol. 36, no. 3, jul 2009. [Online]. Available: https://doi.org/10.1145/1527286.1527287

[23] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*. Manchester: Manchester University Press, 1992.

[24] G. M. Slota, K. Madduri, and S. Rajamanickam, "PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks," in *Proc. IEEE Int'l. Conf. on Big Data (Big Data)*, 2014.

[25] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. on Mathematical Software*, vol. 38, no. 1, 2011.

[26] G. M. Slota, S. Rajamanickam, K. Devine, and K. Madduri, "Partitioning trillion-edge graphs in minutes," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 646–655.

[27] E. G. Boman, K. D. Devine, and S. Rajamanickam, "Scalable matrix computations on large scale-free graphs using 2d graph partitioning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2503210.2503293