Empirical Evaluation of Graph Partitioning Using Spectral Embeddings and Flow

Kevin J. Lang[1], Michael W. Mahoney[2], and Lorenzo Orecchia[3]

[1] Yahoo! Research, Santa Clara, CA, USA
[2] Stanford University, Stanford, CA, USA
[3] University of California, Berkeley, CA, USA

Abstract. We present initial results from the first empirical evaluation of a graph partitioning algorithm inspired by the Arora-Rao-Vazirani algorithm of [5], which combines spectral and flow methods in a novel way. We have studied the parameter space of this new algorithm, *e.g.*, examining the extent to which different parameter settings interpolate between a more spectral and a more flow-based approach, and we have compared results of this algorithm to results from previously known and optimized algorithms such as Metis.

1 Introduction

Graph partitioning refers to the problem of dividing an input graph into two large pieces such that the number of edges crossing the partition is minimized. There are several standard formalizations of this bi-criterion, and in this paper we focus on minimizing expansion.[1] Given an undirected, possibly weighted, graph $G = (V, E)$, *the expansion* $\alpha(S)$ *of a set of nodes* $S \subseteq V$ is defined as:

$$\alpha(S) = \frac{|E(S, \overline{S})|}{\min\{|S|, |\overline{S}|)\}}, \tag{1}$$

where $E(S, \overline{S})$ denotes the set of edges having one end in S and one end in the complement $\overline{S}$, and where $|\cdot|$ denotes cardinality (or weight). The *expansion of the graph* G is then defined as:

$$\alpha(G) = \min_{S \subseteq V} \alpha(S). \tag{2}$$

It is well-known that solving (2) exactly is NP-hard. Graph partitioning is, however, of interest in many applications. For example, it has been used in divide-and-conquer algorithms; for load balancing in parallel computing applications; to segment images and, more generally, to cluster data; and to find clusters and communities in large social and information networks.

Graph partitioning is also a problem for which a wide range of algorithms have been developed, and the theoretical and/or empirical strengths and weaknesses of these algorithms have been extensively studied. Most algorithms that

[1] Expansion is sometimes referred to as the quotient cut objective.

J. Vahrenhold (Ed.): SEA 2009, LNCS 5526, pp. 197–208, 2009.
© Springer-Verlag Berlin Heidelberg 2009

have been designed to find good cuts, *i.e.*, low-expansion partitions, employ one or more of the following four algorithmic ideas: *spectral methods*; *flow-based methods*; *local improvement*; and *multi-resolution*. Historically, spectral methods and flow-based methods have dominated the theoretical landscape; and variants of spectral methods, as well as local improvement techniques, often in a multi-resolution setting, have dominated applications. In particular, note that the heuristic Metis [12] is often the method of choice in applications since it finds good-quality cuts in mesh-like graphs very quickly.

Researchers also noticed that spectral and flow-based methods tend to have complementary strengths—the worst-case examples for spectral algorithms are easy graphs for flow-based methods, and vice-versa—which has lead to attempts to combine spectral and flow into a better graph partitioning algorithm. This was achieved by Arora, Rao, and Vazirani (ARV) [5], who developed the concept of "expander flows" and who introduced an algorithm that achieves an $O(\sqrt{\log n})$ worst-case approximation to expansion and several related quantities. The original version of ARV [5] yields a polynomial-time algorithm, but one that is too slow to be practical, as it must solve a large semi-definite program.

The ARV breakthrough was followed by the introduction of several related algorithms exploring the running-time versus quality-of-approximation tradeoff [3,4,13]. In particular, Orecchia, Schulman, Vazirani, and Vishnoi (OSVV) [16] developed an algorithm that performs only polylogarithmic single commodity max-flow computations to achieve an $O(\log n)$ approximation. The fastest theoretical algorithm for these single commodity flow computations has time complexity $\tilde{O}(n^{3/2})$ [10], but push-relabel methods have been shown to be faster in practice [7], potentially making the OSVV algorithm useful in applications. OSVV and other methods inspired by [5] have been reviewed in [6], where the authors of ARV pose the question of how well they will perform empirically, in particular with respect to Metis.

In this paper, we report initial results from the first empirical evaluation of an algorithm from this novel family. We have implemented the algorithm of OSVV [16], and we have compared it with several implementations of traditional graph partitioning algorithms, including Metis, on a suite of graphs designed to highlight the strengths and weaknesses of previously existing algorithms. We demonstrate that the algorithmic ideas underlying the ARV method can be implemented on medium-sized graphs to find cuts that are competitive with those returned by existing algorithms; and we demonstrate a manner in which different parameter settings interpolate between spectral and flow methods.

2 Algorithms

In this section, we describe the algorithms we used in our empirical evaluation. We compared the algorithm of OSVV [16] with two versions of Metis, two versions of the spectral method, and one purely flow-based method:

– Metis is a fast heuristic that combines a multi-resolution approach with local improvement techniques [12].

- MetisRand is a randomized variation of the basic Metis algorithm that achieves much better results.
- Spectral is the classical spectral method of [1], which uses a sweep cut to round the eigenvector solution.
- SpecFlow is a variation of Spectral in which the standard sweep-cut rounding is replaced by a flow-based rounding which is guaranteed to obtain a better or equally good cut [2].
- LR is a simplified version [14] of the flow-based algorithm by Leighton and Rao [15].
- OSVV is our implementation of the the algorithm of OSVV [16], which uses ideas related to the original ARV algorithm [5]. Our implementation closely follows the theory, and the approximation guarantees of [16] still apply.

Note that our comparison includes both standard versions of the traditional algorithms (Metis and Spectral), as well as modified versions (MetisRand and SpecFlow) which in practice find much better cuts.

2.1 The Improve Algorithm

We start by describing a flow-based "improvement" procedure that will be an important building block for SpecFlow and OSVV. The Improve algorithm was originally introduced as a post-processing procedure to improve cuts returned by other methods [2]. This algorithm takes as input a bisection $(A, \overline{A})$, and it looks for a cut which optimizes a combination of low expansion and correlation with the starting bisection. The algorithm outputs a cut $(T, \overline{T})$ and a perfect matching M between A and $\overline{A}$. If the expansion of $(T, \overline{T})$ is α, then M can be routed[2] in G with congestion $1/\alpha$. This matching M can then be used as a certificate that $(T, \overline{T})$ has better expansion than all cuts strictly contained in $(A, \overline{A})$. This algorithm can be implemented by a small number of single commodity max-flow computations. We used the C++ implementation of [2], which is based on the max-flow push-relabel program `hi_pr` v.3.4, described in [8].

2.2 The Metis and MetisRand Algorithms

Metis is a heuristic developed by Karypis and Kumar [12] to find good balanced partitions in graphs. Although it has no theoretical guarantees, in practice it runs extremely fast. Metis makes many random choices during its execution, but the standard version of the code has a fixed random seed which makes the algorithm deterministic; our Metis results were obtained by running the unmodified pmetis program (version 4.0.1). MetisRand is a modified version of the basic program in which the random seed is left as an input to the program

[2] A weighted graph H can be *routed as a flow* in a graph G if every edge $e = \{u, v\} \in E(H)$ in H with weight w_e can be routed on a path from u to v in G such that the total congestion on every edge of G, *i.e.*, the total weight of H routed across that edge in G, is less or equal to 1. In this case, we can use H as a certificate of expansion, since one can show that $\alpha(H) \leq \alpha(G)$.

and the best of 10,000 runs is returned; in addition to small code changes to allow different seeds, we changed the matching method from the default "Sorted Heavy Edge Matching" to "Random Matching."

2.3 The Spectral and SpecFlow Algorithms

Spectral algorithms compute (exactly or approximately) the second eigenvector x of the Laplacian[3] of the graph, and then approximate the best cut in the graph by a cut defined by this vector. Recall that this eigenvector assigns to each vertex v of the graph a value x_v, and if we assume that these have been ordered, this allows one to define $n-1$ "sweep cuts" $(S_i, \overline{S_i})$, for $1 \leq i < n$, as: $S_i = \{v \in V : x_v < x_i\}$. The eigenvector can be rounded to a cut by picking the best of these sweep cuts. Our SPECTRAL algorithm computes the smallest second, third and fourth eigenvectors of the Laplacian, applies the sweep cut rounding to each of them, and then returns the best cut found.

SPECFLOW is a randomized variant of SPECTRAL that was developed by Andersen and Lang [2] and that differs from SPECTRAL in two respects. First, rather than using each eigenvector separately, SPECFLOW uses a random combination of the lowest three non-zero eigenvectors of the Laplacian of G. This makes SPECFLOW more robust against cuts which may be hidden from a single eigenvector. Second, SPECFLOW replaces the rounding by a sweep cut with a call to IMPROVE on the bisection $(S_{n/2}, \overline{S_{n/2}})$. In [2], it is proven that the IMPROVE rounding procedure is strictly no worse than rounding by a sweep cut, and it is shown that SPECFLOW outperforms SPECTRAL on most graphs.

The computation of the second eigenvector of the Laplacian can be performed in a number of ways (we computed "exact" eigenvectors with ARPACK), but most relevant for the subsequent discussion is that it can be performed by considering exact or approximate random walks on the instance graph G. The idea underlying this approach is that random walks will mix slowly across cuts containing few edges, and conversely that if a random walk mixes slowly then there must be some cut which is constraining the spreading of the probability mass. Moreover, the second eigenvector of the Laplacian defines the slowest mixing direction, and the second smallest eigenvalue, *i.e.*, the spectral gap λ_2, characterizes the mixing time [9]. This highlights the weakness of the spectral method at finding good cuts: spectral algorithms are sensitive not only to sparse cuts but also to "large distances" in the graph—a random walk may fail to mix rapidly either because it takes a long time to overcome a sparse cut or because the graph has very long paths along which the random walk makes slow progress. The worst examples for spectral methods are based on this idea [11].

2.4 The LR Algorithm

Flow-based methods provide a very different way to find good cuts in a graph. They route a certificate graph H in G, and use this routing to provide both

[3] The *Laplacian* of a graph K is $L(K) = D(K) - A(K)$, where $D(K)$ is a diagonal matrix containing the degree of each vertex and $A(K)$ is the adjacency matrix.

a lower bound on expansion as well as an approximate cut by showing that if no better lower bound can be proved then by duality a good cut must exist. Leighton and Rao [15] chose H to be a scaled version of the complete graph on n vertices, and they chose a linear programming relaxation of the problem based on multi-commodity flows. The best implementations of this lead to a theoretical running time of $\tilde{O}(n^2)$, but in our empirical evaluation we used the implementation of [14], which runs faster since it only approximately solves the flow problem. Graphs on which LR is known to perform poorly include constant-degree expanders [15] and expanders with planted cuts [2].

2.5 The OSVV Algorithm

OSVV and other related algorithms subsequent to that of ARV [5] have a simple interpretation based on a modification of flow-based ideas: they strengthen the flow-based approach of Leighton and Rao [15] by removing the limitation that the graph to be routed in G be a complete graph by instead allowing it to be any graph with large spectral gap [6]. Here, we are going to give a dual interpretation of these algorithms as based on a modification of spectral ideas: OSVV strengthens the standard spectral approach by using flow-based ideas to modify the instance graph to make it more amenable to spectral methods. It does so by using the matching returned by IMPROVE to add edges to the input graph to fix the oversensitivity of spectral methods to large distances.

Before describing the OSVV algorithm, recall that the *heat kernel* of a graph $K = (V, E)$ is defined as: $H_K^\eta = \exp(-\eta L(K))$, where $L = L(K)$ is the Laplacian of K and where $\eta \geq 0$ is a learning rate. The heat kernel can be used in an alternative version of the standard spectral method to produce an approximate eigenvector of the Laplacian as follows: take a vector v picked uniformly at random from $\{+1, -1\}^V$ and consider $x = H_K^\eta v = \exp(-\eta L(K))v$. As η varies between 0 and infinity, the vector x becomes a better and better approximation to the second eigenvector of the Laplacian. Replacing the exact computation of the second eigenvector of the Laplacian with an approximation based on the heat kernel with $\eta \neq \infty$ has two potential advantages. First, the heat kernel is more robust against cuts hidden from the second eigenvector. Second, the computation of $H_K^\eta v$ is faster than that of the second eigenvector, especially for graphs with a small spectral gap.

Our OSVV algorithm takes as input a graph G, as well as parameters η, γ, and stopping condition σ. It then does the following:

1. Let $G' = \gamma G$; and $t = 0$.
2. Approximate the second eigenvector of the Laplacian of G' by performing a heat kernel computation on G'.
3. Using the bisection $(S_{n/2,t}, \overline{S_{n/2,t}})$ from the sweep cut along this approximate eigenvector, call IMPROVE with G to get a cut $(T_t, \overline{T_t})$ and a matching M_t.
4. Let $G' = G' + M_t$; and $t{+}{+}$. Until the stopping rule is satisfied, goto Step 2.
5. Return as output the cut $(T_t, \overline{T_t})$ of minimum expansion found in Step 3.

In [16], it is shown that the algorithm takes at most $O(\log^2 n)$ rounds to achieve an $O(\log n)$ approximation. Our implementation closely follows the theory [16] and has the following three parameters:

- The learning rate, η, determines how much the spectral computation of the heat kernel on G'_t is allowed to converge towards the second eigenvector of G'_t. (For higher values of η, the spectral part of the algorithm is more global, but it is also the more susceptible to the errors caused by long paths.)
- The initialization coefficient, γ, determines the weight of the instance graph G in G'. (For higher values of γ, G' depends relatively more on G, and less on the feedback matchings output by the IMPROVE algorithm, and thus the more similar the spectral computation on G' is to that of G.)
- The stopping condition, σ, is the number of iteration after which, if no improvement in the best cut found has occurred, the algorithm aborts. (A higher stopping condition can yield a better solution at the expense of time, while a lower stopping condition can make the algorithm faster but may prevent it from achieving its best expansion scores.)

Note that in addition to the instance graph G, OSVV maintains a graph G', which starts off equal to a scaled version of G and is progressively modified to be more suited to spectral methods. At every iteration t, the approximate spectral computation is performed on the current G'_t and a sweep bisection $(S_{n/2,t}, \overline{S_{n/2,t}})$ is obtained from the resulting vector. The IMPROVE algorithm is then applied to this bisection *on the input graph* G (since we are interested in cuts on G), and this will yield a cut $(T_t, \overline{T_t})$. Now, either there is a good sweep cut in the original eigenvector, which would have been found by IMPROVE, or the spectral method has been fooled by some long paths in G. To fix this problem, OSVV considers the matching M_t returned by IMPROVE at iteration t. Since the endpoints of the edges of M_t lie on opposite sides of the bisection $(S_{n/2,t}, \overline{S_{n/2,t}})$, M_t can be used to "shortcut" the long paths, and so the algorithm sets $G'_{t+1} = G'_t + M_t$. Clearly, the matching M_t can be thought as providing *iterative feedback* to the spectral method about the quality of the cut found and how to modify G'_t to explore different cuts and identify better cuts.

Note also that for large values of γ and η, OSVV becomes very spectral in flavor, as G' becomes dominated by G and the heat kernel is allowed to converge closer to the second eigenvector of G. For example, in the first iteration, the spectral computation performed by OSVV is the multiplication of a random vector by $\exp^{-\eta\gamma G}$. Hence, the higher the product $\gamma\eta$, the more the first iteration of OSVV will look like a second eigenvector computation; and similarly for subsequent iterations. Conversely, as η and γ decrease, the algorithm performs a more localized spectral computation and the feedback matchings increase in weight with respect to G, yielding an algorithm with a stronger flow-based flavor.

3 Graphs Used in Our Empirical Evaluation

In this section, we describe the graphs we used to perform our empirical evaluation of the algorithms described in Section 2. Our main testbed consists of

Table 1. Basic statistics for our main testbed of graphs, including information about the best quotient cut found during our empirical evaluation. The "cutsize" is the number of edges cut in the best quotient cut found by any method, and "smallside" is the number of nodes on the small side of the cut, in which case the balance is smallside / nodes and the quotient cut score (not displayed) would be cutsize / smallside.

graph	GM.100.6	PLANT5K	PLANT6K	WING	TOOTH	RND-A	A1.I2	A3.I4	A6.I3	A9.I0
nodes	12600	20000	20000	62032	78136	10000	10000	10000	10000	10000
edges	24974	45000	46000	121544	452591	59372	47778	52024	62561	71332
cutsize	100	5000	7055	791	3827	181	622	1723	3508	5301
smallside	6300	10000	9955	31008	39030	4945	4982	4950	4959	5000
balance	0.5000	0.5000	0.4978	0.4999	0.4995	0.4945	0.4982	0.4950	0.4959	0.5000

10 graphs, summarized in Table 1, ranging in size from 10,000 up to 78,000 vertices and chosen from the following five classes. These five classes were chosen to highlight the strengths and weaknesses of existing algorithms; existing graph-partitioning testbeds are less appropriate for this empirical evaluation since they tend to be easy for spectral methods, since they consist of mesh-like graphs like WING and TOOTH. Note, in addition, that the best cuts found are all very well-balanced.

3.1 Guattery-Miller Graph

This graph is based on the construction of Guattery and Miller [11] for worst-case graphs for eigenvector-based methods. It is the outer product of a double-tree (two complete binary trees connected by an edge between their roots) and a path graph; and the minimum expansion cut is obtained by separating the two trees, but the path can be made long enough so that the slow mixing along it will cause any given number of eigenvectors to cut the path instead. We include in our testbed the graph GM.100.6, in which the path has length 100 and each of the two trees is of depth 6. These parameters have been chosen so that the first 3 eigenvectors will be given by the first three modes of vibration for the path, *i.e.*, the path folded over itself once, twice, and three times. The "right" eigenvector appears in the fourth position, where it cannot be used by SPECTRAL or SPECFLOW, which only use the first 3 non-zero eigenvectors.

3.2 Expanders with Planted Bisections

Expanders with planted bisections (expander-like graphs with a distinctly good bisection planted at a random location) are known to be a worst-case inputs for LR, while they can be solved by spectral methods and by various local improvement algorithms (see, *e.g.*, [2]). We generated a family of 8 graphs each containing 20,000 nodes, with planted bisections of size $k \cdot 1000$, for $k = 1$ through $k = 8$, by generating two 10,000-node degree-4 expanders (each is the union of 4 disjoint random matchings of the nodes) and connecting them by a random k-matching.

As the size of the planted cut increases, it becomes harder to detect it. For example, the graph with planted cut of size 1000 is solved optimally by MetisRand, LR, SpecFlow and all parameter choices of OSVV, while LR already fails to find the planted cut of size 2000. MetisRand, SpecFlow and OSVV find the planted cut up through size 5000 and then all fail to find at size 6000 and larger. Thus, we include in our results plant5k and plant6k, the graphs with planted cut of size 5000 and 6000, respectively.

3.3 Finite Element Meshes

Well-shaped meshes are classic examples of "nice" low-dimensional graphs for which many graph partitioning methods have been developed. We include the finite element mesh wing and tooth from the archive [17].

3.4 Random Geometric Graphs

Random geometric graphs have long been standard benchmark graphs [14], and LR, SpecFlow and MetisRand have been shown to perform well on them. We include one random geometric graph rnd-a generated by picking $10,000$ random points in the unit two-dimensional disk, adding edges in increasing order of length, and stopping when the graph becomes connected. We then reduced the number of edges by removing all pairs in which one end was not among the 50 nearest neighbors of the other.

3.5 Random Geometric Graphs with Random Edges Added

Given the good results obtained by the algorithms on random geometric graphs, we make them harder by adding a number of completely random edges, as was done in [2]. Our claim that these graphs are harder than ordinary random geometric graphs is based on the empirical observation of higher-variance distributions of scores from randomized partitioning algorithms [2]. We do not know of a theoretical explanation, but intuitively the extra edges seem to cause the spectral embeddings to get twisted up, making the right answer much less obvious. Similarly, perhaps by increasing the expansion, the extra random edges also cause problems for LR.

Each graph ax.iy was obtained by first running the random geometric graph generator mentioned above. Then, $x \cdot 1000$ random edges were added (avoiding the creation of duplicates); the tag y is just an instance number. In our testbed, we include a selection of random geometric graph with random edges based on which one or more of the algorithms performs well: a1.i2, on which LR does particularly well; a3.i4, on which SpecFlow does well; a6.i3, on which OSVV performs well; and a9.i0, for which MetisRand gave the best results.

4 Results

In this section, we describe the results of our initial empirical evaluation. Our computations were run on 4 64-bit AMD Opteron Processors, each running at

1795MHz, with 32GB of memory and 1024KB cache. We ran the deterministic Spectral and Metis algorithms only once, and we report the expansion of the cut found and the time required. For LR and for each of the parameter choices of OSVV, we ran 10 trials; for SpecFlow, we ran 1000 iterations; and for MetisRand, we ran 10,000 trials. For each of these algorithms, we report the best cut found over these runs and the total time taken. The number of trials for each algorithm was chosen in order to obtain total run times of the same order of magnitude in order to help focus the comparison on just the single criterion of the best cut found; we discuss this issue in more detail below. Finally, the timing data omit the time needed to load the description of the graph.

We explored a large number of settings of the parameters in preliminary empirical evaluations, and we determined that the interesting region of parameter space for our graphs is given by the 27 combinations of the following sets of parameters: $\eta \in \{1, 10, 100\}; \gamma \in \{0, 10, 100\}; \sigma \in \{2, 5, 10\}$. Many of our conclusions may be illustrated by considering only 3 setting of the parameters: $\eta = 100, \gamma = 100, \sigma = 10$; $\eta = 10, \gamma = 10, \sigma = 10$; and $\eta = 1, \gamma = 0, \sigma = 10$. (Recall that choices of the parameters for which the product $\eta\gamma$ is larger (resp. smaller) correspond to a more spectral-like (resp. flow-like) behavior of the algorithm.) Summary statistics for the best cut results and total times for these 3 parameter settings of OSVV, compared with results from each of the other algorithms, are presented in Table 2 and Table 3.

On plant5k, the more spectral-based choices of parameters find the planted cut, while it seems that the more flow-based choices encounter problems similar to (but not as severe as) that of LR and are not able to detect the right cut. On plant6k, SpecFlow gives the best cut overall (although no algorithm found the planted cut) and the more spectral-like choices of parameters for OSVV do marginally better than more flow-like choices. Note, though, that the entire OSVV method seems to be stuck at a quality around 10% worse than SpecFlow, probably as a result of the constant-degree expander strongly affecting the performance of the flow part of OSVV. For meshes and random geometric graphs, all the methods do comparably. Note, though, that the

Table 2. [Best viewed in color.] Ratio of the best expansion cut score found by multiple trials of each algorithm to the best expansion cut score found overall. (See the text or the caption of Table 3 for details on the number of trials for each algorithm, and see Table 4 for results on varying the number of trials.) First and second place for each graph are highlighted in red and blue, respectively. Ratios are given to 3 decimal digits. OSVV parameters are described as OSVV-$\eta.\gamma.\sigma$.

	gm.100.6	plant5k	plant6k	wing	tooth	rnd-a	a1.i2	a3.i4	a6.i3	a9.i0
OSVV-100.100.10	**1.000**	**1.000**	**1.098**	**1.018**	**1.003**	1.024	1.077	**1.029**	**1.009**	1.039
OSVV-10.10.10	**1.000**	**1.000**	1.102	1.069	1.033	**1.001**	**1.050**	1.039	**1.000**	**1.033**
OSVV-1.0.10	**1.000**	1.493	1.119	1.069	1.059	1.003	1.053	1.089	1.414	1.077
MetisRand	**1.000**	**1.000**	1.107	1.048	1.019	1.011	1.149	1.068	1.025	**1.000**
LR	**1.000**	2.841	2.082	1.069	1.065	1.003	**1.000**	1.163	1.072	1.075
SpecFlow	**1.260**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.149	**1.000**	1.037	1.081
Metis	1.640	1.526	1.130	1.169	1.208	1.322	1.445	1.330	1.190	1.059
Spectral	**1.260**	**1.195**	1.207	1.253	1.111	1.517	2.624	1.878	1.414	1.661

Table 3. [Best viewed in color.] Total run time in seconds for OSVV-η.γ.σ (10 trials), MetisRand (10000 trials), LR (10 trials), SpecFlow (Eigensolver + 1000 flow roundings), Metis (1 try), and Spectral (Eigensolver + 3 sweep roundings). Numbers are rounded to the nearest second, except for Metis and Spectral, where they are rounded to the second decimal.

	gm.100.6	plant5K	plant6K	wing	tooth	rnd-a	a1.i2	a3.i4	a6.i3	a9.i0
OSVV-100.100.10	**793**	**367**	**650**	**8167**	**31847**	1956	956	**735**	**1315**	1013
OSVV-10.10.10	**363**	**304**	437	2802	9923	**881**	**401**	370	**485**	**851**
OSVV-1.0.10	**426**	2075	3030	4201	11681	602	447	441	85	423
MetisRand	**105**	**681**	700	1049	2024	110	111	189	284	**328**
LR	**187**	660	658	8521	56378	443	**509**	699	1173	1637
SpecFlow	**209**	**636**	**581**	**4887**	**13254**	**688**	639	**641**	724	798
Metis	0.01	0.06	0.07	0.09	0.21	0.01	0.01	0.02	0.02	0.03
Spectral	**7.05**	**3.22**	3.27	51.48	96.02	8.98	4.40	3.10	2.32	2.46

traditional mesh wing and tooth slightly favors spectral methods—SpecFlow does slightly better than other methods, and relatedly the best OSVV results are given by spectral-like and intermediate choices of parameters. As expected, on gm.100.6, all choices of parameters for OSVV find the optimal cut.

For random geometric graphs with added random edges, intermediate and flow-like parameter choices for OSVV tend to perform well for a1.i2 (chosen since LR performed well on it), and the more spectral-like choices perform better for a3.i4 (chosen since SpecFlow performed well). Interestingly, for a6.i3, parameters intermediate between spectral and flow decisively beat other parameter choices and all other algorithms. A similar result is seen with a9.i0, for which the intermediate parameter choices nearly tie the best cut found by MetisRand and improve on the expansion found by SpecFlow and LR by around 5%.

With respect to the stopping condition σ, (data not presented indicate that) computations behaved in expected ways, but we noticed that variations in the stopping condition seemed to impact more the quality of the score for more flow-based algorithms. Our intuition for this is that more flow-based algorithms are less aggressive in their search for sparse cuts and require more time to explore the cut space to provide their best results, while more spectral runs can achieve very good scores already in their first runs, especially if the graph is suited to the spectral method. In generally, of course, the stopping condition could be adjusted based on the relative importance of cut quality and time in the context in which the algorithm is used.

With respect to the running time, Table 3 (and data not presented) indicates that more extreme choices of parameters η and γ seem to require more running time. This is likely due to two different reasons. On the one hand, for the more spectral-like parameter settings, a larger fraction of time is spent in computing the heat kernel vector, which becomes harder as η and γ grow. On the other hand, for more flow-like parameter settings, the longer time is usually due to a larger number of iterations within the algorithm, again a result of the more conservative approach of these parameter settings. In general, if some information on the graph is available in terms of suitability to spectral or flow methods, the choice of parameters could be adjusted accordingly.

Table 4. Varying the number of trials for MetisRand and SpecFlow. Presented is the ratio of the expansion cut score for MetisRand and SpecFlow (as a function of the number of trials) to the best overall score. For MetisRand 10000 and SpecFlow 1000, the score is the minimum found over all our trials, while for other number of trials, the score is an estimate of the average best score using the empirical distribution from our experiment and assuming sampling with replacement.

	Trials	gm.100.6	plant5k	plant6k	wing	tooth	rnd-a	a1.12	a3.14	a6.13	a9.10
MetisRand	10	1.343	1.313	1.121	1.099	1.075	1.137	1.362	1.197	1.106	1.072
MetisRand	100	1.020	1.028	1.116	1.076	1.043	1.058	1.253	1.143	1.071	1.042
MetisRand	1000	1.000	1.000	1.111	1.063	1.028	1.026	1.175	1.102	1.046	1.020
MetisRand	10000	1.000	1.000	1.107	1.048	1.019	1.011	1.149	1.068	1.025	1.000
SpecFlow	1	1.913	1.402	1.100	1.082	1.161	1.278	1.325	1.150	1.108	1.171
SpecFlow	10	1.284	1.066	1.052	1.040	1.045	1.027	1.218	1.059	1.060	1.106
SpecFlow	100	1.260	1.000	1.011	1.006	1.015	1.002	1.167	1.023	1.044	1.089
SpecFlow	1000	1.260	1.000	1.000	1.000	1.000	1.000	1.149	1.000	1.037	1.081

A graph by graph inspection seems to confirm the expected behavior of the parameters of OSVV as toggling between spectral and flow methods. In addition, the results suggest that OSVV is quite robust and tends to be within 5% of the best, which is much more consistent performance than the other methods. In particular, that the performance of the intermediate choice of parameters $\eta = 10, \gamma = 10, \sigma = 10$ appears as a good global setting, performing optimally or near-optimally for all graphs except plant6k, for which more spectral-based methods are preferable.

Finally, we should note that Table 4 demonstrates that for MetisRand and SpecFlow the number of trials (and thus the run time) can be decreased by a factor of 10, or in some cases 100, while still finding cuts that are only moderately worse than those found in the larger number of trials. A larger study (currently in progress) with a finer tuning of parameters, more comprehensively chosen graphs, and a more sophisticated implementation of OSVV will be necessary to fully explore these issues and validate our initial observations.

5 Conclusion

We have reported initial results from the first empirical evaluation of an algorithm from the novel family of algorithms inspired by the recent theoretical work of ARV [5]. It is important to emphasize that, prior to performing this empirical evaluation, we had no idea whether any algorithm from this family of algorithms would be at all practical in finding even moderately good cuts on graphs of any size. Thus, our primary conclusion is that a simple implementation of the algorithm of OSVV [16] performs competitively with state-of-the-art implementations of existing graph partitioning algorithms at finding good cuts on a suite of medium-sized graphs chosen to illustrate the strengths and weaknesses of these existing algorithms. Our secondary conclusion is that, as suggested by theory, different parameter choices in this algorithm can be interpreted as toggling between a more spectral-like approach and a more flow-like approach. Clearly, our initial results suggest that these methods might be a viable alternative to the

classical spectral and flow methods in practical applications in large-scale data analysis and machine learning, arguing for a more comprehensive evaluation on a larger suite of larger and more realistic graphs.

References

1. Alon, N., Milman, V.: λ_1, isoperimetric inequalities for graphs and superconcentrators. J. Combin. Theory B 38, 73–88 (1985)
2. Andersen, R., Lang, K.: An algorithm for improving graph partitions. In: SODA 2008: Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms, pp. 651–660 (2008)
3. Arora, S., Hazan, E., Kale, S.: $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In: FOCS 2004: Proceedings of the 45th Annual Symposium on Foundations of Computer Science, pp. 238–247 (2004)
4. Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: STOC 2007: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 227–236 (2007)
5. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: STOC 2004: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 222–231 (2004)
6. Arora, S., Rao, S., Vazirani, U.: Geometry, flows, and graph-partitioning algorithms. Communications of the ACM 51(10), 96–105 (2008)
7. Cherkassky, B., Goldberg, A., Martin, P., Setubal, J., Stolfi, J.: Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms. Journal of Experimental Algorithmics 3, Article 8 (1998)
8. Cherkassky, B., Goldberg, A.V.: On implementing push-relabel method for the maximum flow problem. Algorithmica 19, 390–410 (1997)
9. Chung, F.: Spectral graph theory. CBMS Regional Conference Series in Mathematics, vol. 92. American Mathematical Society, Providence (1997)
10. Goldberg, A., Rao, S.: Beyond the flow decomposition barrier. Journal of the ACM 45, 783–797 (1998)
11. Guattery, S., Miller, G.: On the quality of spectral separators. SIAM Journal on Matrix Analysis and Applications 19, 701–719 (1998)
12. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20, 359–392 (1998)
13. Khandekar, R., Rao, S., Vazirani, U.: Graph partitioning using single commodity flows. In: STOC 2006: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 385–390 (2006)
14. Lang, K., Rao, S.: Finding near-optimal cuts: an empirical evaluation. In: SODA 1993: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 212–221 (1993)
15. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM 46(6), 787–832 (1999)
16. Orecchia, L., Schulman, L., Vazirani, U., Vishnoi, N.: On partitioning graphs via single commodity flows. In: STOC 2008: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 461–470 (2008)
17. Walshaw, C., Cross, M.: Mesh partitioning: a multilevel balancing and refinement algorithm. SIAM Journal on Scientific Computing 22(1), 63–80 (2000)