An Algorithm for Improving Graph Partitions

Reid Andersen [*] Kevin J. Lang [†]

Abstract

We present an algorithm called `Improve` that improves a proposed partition of a graph, taking as input a subset of vertices and returning a new subset of vertices with a smaller quotient cut score. The most powerful previously known method for improving quotient cuts, which is based on parametric flow, returns a partition whose quotient cut score is at least as small as any set contained within the proposed set. For our algorithm, we can prove a stronger guarantee: the quotient score of the set returned is nearly as small as any set in the graph with which the proposed set has a larger-than-expected intersection. The algorithm finds such a set by solving a sequence of polynomially many $s-t$ minimum cut problems, a sequence that cannot be cast as a single parametric flow problem. We demonstrate empirically that applying `Improve` to the output of various graph partitioning algorithms greatly improves the quality of cuts produced without significantly impacting the running time.

1 Introduction

The minimum quotient cut problem is a fundamental and well-studied graph partitioning problem. The goal is to find, within an input graph with weighted vertices, a subset of vertices whose weight is at most half the total weight of the graph, and whose quotient score is as small as possible. The quotient score measures the quality of the set as a partition of the graph, and is defined to be the number of edges between the set and its complement, divided by the weight of vertices in the set. The problem is NP-hard [16], and includes as special cases the minimum ratio cut problem and minimum conductance cut problem.

Approximation algorithms for the minimum quotient cut problem have numerous practical and theoretical applications; they have been used to develop divide-and-conquer algorithms [31], to solve multiway partitioning problems [32], for VLSI layout [4], and for clustering [6, 9, 21]. There are several families of algorithms for finding quotient cuts, including local search heuristics [23, 14], spectral partitioning and clustering algorithms [10, 29, 21], the multicommodity flow algorithm of Leighton and Rao [25], and the multilevel heuristic implemented in METIS [22]. The best approximation ratio known for the minimum quotient cut problem is $O(\sqrt{\log n})$, which is attained by the semidefinite programming algorithm of Arora, Rao, and Vazirani [2].

We consider the complementary task of taking a proposed set of vertices and improving its quotient score. There is a surprisingly powerful known algorithm for this task. Given a set A of vertices whose weight is at most half the total weight of the graph, the subset of A with the smallest quotient score can be found in polynomial time by solving a single parametric maximum flow problem. This algorithm was described in the paper of Gallo, Grigoriadis, and Tarjan [15], as an application of their fast parametric flow algorithm. This flow-based improvement technique was a key tool in several theoretical results, most notably in Feige and Krauthgamer's approximation algorithm for the minimum bisection problem [13], and in finding hierarchical oblivious routing schemes [18]. In addition to its theoretical importance, the flow-based improvement technique is a useful heuristic for improving the results of other partitioning algorithms. This was demonstrated by Lang and Rao [24], who showed that applying flow-based improvement to the output of existing partitioning algorithms, including spectral partitioning and METIS, produced the best known partitions for a variety of benchmark graphs.

In this paper, we describe and analyze an algo-

[*]Microsoft Research (reidan@microsoft.com)
[†]Yahoo! Research (langk@yahoo.com)

rithm called Improve that is strictly more powerful than the improvement method described in the previous paragraph. Like the existing method, Improve produces a set with quotient score at least as small as the best subset of the proposed set A. In addition, we prove the following stronger result, which we state now for the special case where each vertex has unit weight. Let C be a given set with small quotient score, let $F = |A \cap C|/|C|$ be the fraction of C that is contained in the proposed set A, and let $E = |A|/|V|$, which is the expected fraction of C contained in a random set of the same size as A. If $F \geq E + \epsilon$, meaning the fraction of C contained in A is better than random, then the Improve algorithm will produce a set whose quotient score is at within a $1/\epsilon$ factor of the quotient score of C. As an immediate consequence, we prove that to find a cut whose quotient score is within a constant-factor of optimal, it suffices to find a set of vertices whose intersection with the optimal quotient cut is a small constant fraction better than one would expect from a randomly chosen set.

The Improve algorithm solves a sequence of minimum cut problems to find the optimal way to simultaneously add and remove vertices from the proposed cut, subject to a carefully designed scoring system that imposes a penalty for each vertex added from outside of the proposed cut. The two key steps in our analysis are the introduction of a modified quotient score that enforces this system of penalties, and a proof that the Improve algorithm produces in polynomial time a set that minimizes this modified quotient score. Previously known improvement methods based on parametric flow can find the optimal way to remove vertices, or the optimal way to add vertices, but do not simultaneously add and remove vertices from the proposed set. The Improve algorithm solves a different sequence of $s - t$ minimum cut problems than existing flow-improvement methods, and the sequence of problems it solves cannot be cast as a single parametric flow problem.

We show that the Improve algorithm is a powerful and efficient tool for improving graph partitions in practice. We demonstrate empirically that the Improve algorithm outperforms existing flow-based improvement methods. Our experiments indicate that the results of most partitioning algorithms are significantly improved by applying Improve as a post-processing step, and that applying Improve to the output of spectral partitioning and METIS produces excellent partitions.

1.1 Related Work

Here we survey previous algorithms for improving graph partitions.

The basic greedy method for improving a quotient cut is to swap vertices from one side of the cut to the other when this improves the quotient score. Variations of this swapping procedure are used in the local search heuristics of Fiduccia-Mattheyses [14] and Kernighan-Lin [23], and within the multilevel algorithm METIS [22].

Vertex-swapping procedures have been used in various algorithms for finding the optimal bisection in random graphs with planted bisections, including Jerrum and Sorkin's algorithm based on simulated annealing [19], Boppana's algorithm based on eigenvector computation [5], and the local search heuristic "Go with the winners" analyzed by Impagliazzo and Dimitriou [12]. The algorithm of Dasgupta et al. [11] combines recursive spectral partitioning with greedy vertex-swapping to find hierarchical planted partitions. Bui et al. [8] find planted bisections using an algorithm based on network flow.

Cut improvement procedures based on network flow are often stronger than greedy methods. Alon and Milman used a flow-based algorithm for finding a vertex separator from an eigenvector [1]. Boykov et al. [7] used parametric minimum cut computations to minimize energy functions that have applications to computer vision and image analysis. The algorithm of Patkar and Narayanan [27] improves the quality of a proposed cut by computing the principal partition of a submodular function. In the special case of quotient cuts, their method reduces to the flow-based algorithm for finding the optimal subset of a proposed cut that we described earlier, but their method applies to more general objective functions. Narasimhan and Bilmes [26] introduced an algorithm that uses submodular function optimization to find the optimal set of vertices to add to a proposed cut.

The main computation performed by the Improve algorithm of this paper is to optimize a certain objective function (the modified quotient cut score) by constructing and solving a sequence of minimum cut problems. It is known that many objective functions can be optimized by constructing similar cut problems. Examples include solving the densest subgraph problem [17], solving the more general *selection problem* introduced by Balinski [3] and Rhys [30], and solving fractional programming problems in the framework of Picard and Queyranne [28]. Numerous applications of parametric flow are given in the paper of Gallo et al. [15].

1.2 Preliminaries Let $G = (V, E)$ be an undirected graph with an edge weight function $w(u, v)$. We define the edge border $\partial(S)$ of a set $S \subseteq V$ to be the sum of the weights of the edges with one endpoint in S and one endpoint in the complement $\bar{S}$.

Let $\pi(v)$ be a function that assigns a positive weight to each vertex in V. We assign a weight to each subset $S \subseteq V$ by letting $\pi(S) = \sum_{v \in S} \pi(v)$. The minimum quotient cut problem is to find a set $S \subseteq V$ that minimizes the quotient score $Q(S)$,

$$Q(S) = \frac{\partial(S)}{\min(\pi(S), \pi(V \setminus S))}.$$

We will assume that the weights $\pi(v)$ are integers, and that the total weight $\pi(V)$ is bounded by a polynomial in the number of vertices in the graph. Two useful weight functions are $\pi(v) = 1$ and $\pi(v) = \text{degree}(v)$, which turn the minimum quotient cut problem into the minimum ratio cut problem and the minimum conductance cut problem.

2 The quotient cut improvement algorithm

In this main section we state and analyze the `Improve` algorithm. The `Improve` algorithm takes as input a set A and outputs a new set S. Our main theorem, stated below, gives upper bounds on the quotient score $Q(S)$ of the improved cut. The first part of the theorem shows that $Q(S)$ is at least as small as the smallest quotient score of any subset of A. This implies that `Improve` is at least as powerful as the existing flow-based improvement algorithm. The second part shows that for any set C, if the fraction of C contained in A is larger by an additive term ϵ than one would expect for a randomly chosen set, then $Q(S)$ is within a $1/\epsilon$ factor of $Q(C)$.

Theorem 2.1. *The algorithm* `Improve` *runs in polynomial time. Let A be a set satisfying $\pi(A) \leq \pi(\bar{A})$, and let $S =$* `Improve`*(A).*

1. *For any set $C \subseteq A$, we have $Q(S) \leq Q(C)$.*
2. *If C is a set for which the intersection of A with C satisfies*

$$\frac{\pi(A \cap C)}{\pi(C)} \geq \frac{\pi(A)}{\pi(V)} + \epsilon \frac{\pi(\bar{A})}{\pi(V)},$$

for some $\epsilon > 0$, then the set S output by the algorithm satisfies

$$Q(S) \leq \frac{1}{\epsilon} Q(C).$$

The proof of this theorem, and the description of the `Improve` algorithm, are given later in this section.

2.1 Overview of the proof of the main theorem In this section we prove the main theorem, after stating the key definitions and lemmas needed for the proof. The cornerstone of the analysis is the definition of a modified quotient score $\tilde{Q}_A$, which we call the quotient score relative to the proposed cut A. This modified quotient score has a denominator $D_A(S)$ that is always smaller than the denominator $\pi(S)$ of the original quotient score $Q(S)$. In effect, this new denominator penalizes S for containing vertices outside of A.

Definition 2.1. *Given a set $A \subseteq V$ that satisfies $\pi(A) \leq \pi(\bar{A})$, we define*

$$D_A(S) = \pi(S \cap A) - \pi(S \cap \bar{A}) \left(\pi(A)/\pi(\bar{A})\right).$$

We remark that $D_A(S)$ may be zero or may be negative. In particular, $D_A(S)$ is zero when $S = \emptyset$ and when $S = V$, and it may be zero in other cases as well. $D_A(S)$ is negative when $S = \bar{A}$. Keeping this in mind, we define with caution the ratio between $\partial(S)$ and $D_A(S)$.

Definition 2.2. *Given a set $A \subseteq V$ that satisfies $\pi(A) \leq \pi(\bar{A})$, we define*

$$\tilde{Q}_A(S) = \begin{cases} \partial(S)/D_A(S) & \text{if } D_A(S) > 0. \\ +\infty & \text{if } D_A(S) \leq 0. \end{cases}$$

We refer to $\tilde{Q}_A$ as the quotient score relative to A.

We prove that a set S achieving the minimum value of $\tilde{Q}_A$ over all subsets of V can be found in polynomial time by solving a sequence of minimum cut problems. This is the task performed by the `Improve` algorithm. The proof of the following lemma, and the description of the `Improve` algorithm, are given in section 2.2.

LEMMA 2.1. *The algorithm* `Improve`(A) *outputs a set* S *that minimizes* $\tilde{Q}_A(S)$. *If the vertex weights* $\pi(v)$ *are integers, the algorithm halts after at most* $\pi(V)^2$ *iterations. If the edges of the graph are unweighted, the algorithm halts after at most* m *iterations, where* m *is the number of edges in the graph.*

We prove a pair of crucial facts relating the modified quotient score $\tilde{Q}_A$ to the actual quotient score Q. The proof of the following lemma is given in section 2.2.

LEMMA 2.2. *Assume that* $\pi(A) \leq \pi(\bar{A})$*, Then,*

1. *For any set* $S \subseteq V$*, we have* $\tilde{Q}_A(S) \geq Q(S)$.
2. *If* C *is a set for which the intersection of* A *with* C *satisfies*

$$\frac{\pi(A \cap C)}{\pi(C)} \geq \frac{\pi(A)}{\pi(V)} + \epsilon \frac{\pi(\bar{A})}{\pi(V)},$$

for some $\epsilon > 0$*, then* $\tilde{Q}_A(C) \leq \frac{1}{\epsilon} Q(C)$.

We can now prove the main theorem by combining the two previous lemmas.

Proof. [Proof of Theorem 2.1] Lemma 2.1 shows that the `Improve` algorithm runs in polynomial time and produces a set S minimizing $\tilde{Q}_A$. We use Lemma 2.2 to show that S satisfies the two assertions of the main theorem.

If $C \subseteq A$, then $\tilde{Q}_A(C) = Q(C)$, and so

$$Q(S) \leq \tilde{Q}_A(S) \leq \tilde{Q}_A(C) = Q(C).$$

If C is a set for which the intersection of A with C satisfies

$$\frac{\pi(A \cap C)}{\pi(C)} \geq \frac{\pi(A)}{\pi(V)} + \epsilon \frac{\pi(\bar{A})}{\pi(V)},$$

for some $\epsilon > 0$, then $\tilde{Q}_A(C) \leq \frac{1}{\epsilon} Q(C)$, and so

$$Q(S) \leq \tilde{Q}_A(S) \leq \tilde{Q}_A(C) \leq \frac{1}{\epsilon} Q(C).$$

2.2 The `Improve` algorithm

In this section, we state the `Improve` algorithm and show that it outputs in polynomial time a set minimizing the relative quotient score $\tilde{Q}_A$. The algorithm works by constructing and solving a sequence of $s - t$ minimum cut problems.

We now describe the $s - t$ minimum cut problems the algorithm will construct. We construct an augmented graph $G_A(\alpha)$ that depends on the input graph G, on the proposed set A and on a parameter $\alpha \in [0, \infty)$ to be determined later. The vertex set of $G_A(\alpha)$ contains the vertex set of G, plus two new nodes $\mathtt{s}$ and $\mathtt{t}$ that serve as the source and sink of the minimum cut problem. The edge set of $G_A(\alpha)$ contains the edges of G, with their original weights, plus the following additional edges from the source and sink: the source node $\mathtt{s}$ is connected to each node v in A by an edge of weight $\alpha\pi(v)$, and the sink node $\mathtt{t}$ is connected to each node v in $V \setminus A$ by an edge of weight $\alpha\pi(v)f(A)$, where $f(A) = \pi(A)/\pi(\bar{A}) \leq 1$. The list below summarizes these edge weights.

$$\begin{aligned} w(\mathtt{s}, v) &= \alpha \cdot 1_{v \in A} \pi(v). \\ w(v, \mathtt{t}) &= \alpha \cdot 1_{v \notin A} \pi(v) f(A). \\ w(u, v) &= w_G(u, v). \end{aligned}$$

Notice that $f(A)$ has been chosen so that the total weight of edges connected to $\mathtt{s}$ is equal to the total weight of edges connected to $\mathtt{t}$.

There is a simple bijective correspondence between subsets $S \subseteq V$ and cuts separating $\mathtt{s}$ and $\mathtt{t}$ in the augmented graph $G_A(\alpha)$. Specifically, we associate the subset $S \subseteq V$ with the cut $(\mathtt{s} \cup S, \mathtt{t} \cup V \setminus S)$. We define the cost of a set S, written $\mathrm{cost}_{A,\alpha}(S)$, to be the cost of the corresponding cut. By the construction of the augmented graph, the cost of a cut is given by the following equation.

$$\mathrm{cost}_{A,\alpha}(S) = \partial(S) + \alpha\pi(A \cap \bar{S}) + \alpha\pi(S \cap \bar{A}) f(A).$$

The cost can be rewritten in terms of $D_A(S)$, as follows.

$$\begin{aligned} \mathrm{cost}_{A,\alpha}(S) &= \partial(S) + \alpha\pi(A \cap \bar{S}) + \alpha\pi(S \cap \bar{A}) f(A) \\ &= \partial(S) + \alpha\pi(A) \\ &\qquad - \alpha\pi(S \cap A) + \alpha\pi(S \cap \bar{A}) f(A) \\ &= \alpha\pi(A) + (\partial(S) - \alpha D_A(S)). \end{aligned}$$

We can now state the `Improve` algorithm.

```
Improve($A$)

  Input and Output. The input is a set $A \subseteq V$
      satisfying $\pi(A) \leq \pi(\bar{A})$, and the output is a
      new set $S \subseteq V$.

  Initialization. Let $S_0 = A$, let $i = 0$, and let
      $\alpha_0 = \tilde{Q}_A(A) = Q(A) < \infty$.

  Main loop.
      1. Compute the minimum $\mathtt{s} - \mathtt{t}$ cut in
         the graph $G_A(\alpha_i)$, and let $S_{i+1}$ be the
         subset of $V$ corresponding to this cut.
      2. Let $\alpha_{i+1} = \tilde{Q}_A(S_{i+1})$.
      3. If $\alpha_{i+1} < \alpha_i$, let $i \leftarrow i + 1$ and repeat
         the loop. Otherwise halt and output $S_i$.
```

We can now prove Lemma 2.1 and Lemma 2.2.

Proof. [Proof of Lemma 2.1] Given the proposed set A, let

$$\alpha_* = \min_{X \subseteq V} \tilde{Q}_A(X).$$

We will prove the set $S = \texttt{Improve}(A)$ satisfies $\tilde{Q}_A(S) = \alpha_*$.

We first show that if there exists a set of vertices X that satisfies $\tilde{Q}_A(X) < \alpha$, then we can find such a set by solving an $s - t$ minimum cut problem in $G_A(\alpha)$. Fix a value of $\alpha \in (0, \infty)$, and let S_α be a fixed set that minimizes $\mathrm{cost}_{A,\alpha}$, satisfying

$$\mathrm{cost}_{A,\alpha}(S_\alpha) = \min_{X \subseteq V} \mathrm{cost}_{A,\alpha}(X).$$

We will show that if any set $X \subset V$ satisfies $\tilde{Q}_A(X) < \alpha$, then S_α also satisfies $\tilde{Q}_A(S_\alpha) < \alpha$. To see this, observe that for any set X, we have $\tilde{Q}_A(X) < \alpha$ if and only if $\partial(X) - \alpha D_A(X) < 0$. If some set $X \subseteq V$ satisfies $\tilde{Q}_A(X) < \alpha$, then

$$\begin{aligned}\mathrm{cost}_{A,\alpha}(S_\alpha) &\leq \mathrm{cost}_{A,\alpha}(X) \\ &= \alpha\pi(A) + (\partial(X) - \alpha D_A(X)) \\ &< \alpha\pi(A).\end{aligned}$$

Then, we have

$$\alpha\pi(A) + (\partial(S_\alpha) - \alpha D_A(S_\alpha)) = \mathrm{cost}_{A,\alpha}(S_\alpha) < \alpha\pi(A).$$

The inequality above implies $\partial(S_\alpha) - \alpha D_A(S_\alpha) < 0$, and hence $\tilde{Q}_A(S_\alpha) < \alpha$.

The Improve algorithm repeatedly applies the procedure described above to produce a set that satisfies $\tilde{Q}_A(S) = \alpha_*$. The algorithm maintains the following loop invariant: At the beginning of the ith step of the main loop, we have $\tilde{Q}_A(S_i) = \alpha_i < \infty$. The initialization of the algorithm ensures this is true for step 0. During step i, the algorithm performs a minimum cut computation in $G_A(\alpha_i)$ to find a set S_{i+1} that minimizes $\mathrm{cost}_{A,\alpha_i}(S)$, and then sets $\alpha_{i+i} = \tilde{Q}_A(S_{i+1})$. The previous paragraph tells us that either $\alpha_{i+1} < \alpha_i$, or else $\alpha_i = \alpha_*$. In the case where $\alpha_{i+1} < \alpha_i$, the algorithm proceeds to step $i + 1$ with a set S_{i+1} and value α_{i+1} that satisfy the loop invariant. In the case where $\alpha_i = \alpha_*$, the algorithm halts and outputs the set S_i, which satisfies $\tilde{Q}_A(S_i) = \alpha_i = \alpha_*$.

We have shown that if the algorithm halts it produces the correct output, but we still need to show that the algorithm halts after a reasonable number of steps. We will do this by showing that the quantity $D_A(S_i)$ strictly decreases during any step in which the algorithm does not halt. For any step $i > 0$ during which the algorithm does not halt, we know α_{i+1} and α_i are not equal to ∞, and so both $D_A(S_{i+1})$ and $D_A(S_i)$ are strictly greater than zero. Then, we may write

$$\begin{aligned}\mathrm{cost}_{A,\alpha_{i-1}}(S_i) &= \alpha_{i-1}\pi(A) + (\partial(S_i) - \alpha_{i-1}D_A(S_i)) \\ &= \alpha_{i-1}\pi(A) \\ &\quad + D_A(S_i)(\tilde{Q}_A(S_i) - \alpha_{i-1}) \\ &= \alpha_{i-1}\pi(A) + D_A(S_i)\left(\alpha_i - \alpha_{i-1}\right).\end{aligned}$$

Similarly,

$$\begin{aligned}\mathrm{cost}_{A,\alpha_{i-1}}(S_{i+1}) = \alpha_{i-1}\pi(A) \\ + D_A(S_{i+1})\left(\alpha_{i+1} - \alpha_{i-1}\right).\end{aligned}$$

Because S_i minimizes $\mathrm{cost}_{A,\alpha_{i-1}}$, we have $\mathrm{cost}_{A,\alpha_{i-1}}(S_i) \leq \mathrm{cost}_{A,\alpha_{i-1}}(S_{i+1})$, which implies

$$D_A(S_i)\left(\alpha_i - \alpha_{i-1}\right) \leq D_A(S_{i+1})\left(\alpha_{i+1} - \alpha_{i-1}\right).$$

Since $\alpha_{i+1} < \alpha_i < \alpha_{i-1} < \infty$, we have $D_A(S_{i+1}) \leq D_A(S_i)$, as desired.

If the vertex weights $\pi(v)$ are positive integers, the quantities $\pi(S \cap A)$ and $\pi(S \cap \bar{A})$ can each take at most $\pi(V)$ possible values. Therefore, the quantity $D_A(S) = \pi(S \cap A) - \pi(S \cap \bar{A})f(A)$ can take at most

$\pi(V)^2$ values for a fixed value of A. Since $D_A(S)$ strictly decreases at each step, the algorithm halts after at most $\pi(V)^2$ steps.

Since both $\tilde{Q}_A(S_i)$ and $D_A(S_i)$ strictly decrease at each step, $\partial(S_i)$ also strictly decreases at each step. If the edges of the graph are unweighted, then $\partial(S_i)$ can take on at most m possible values, so the algorithm halts after at most m steps.

Although our upper bound on the number of steps required for `Improve` to converge is large, in practice the algorithm often converges after a much smaller number of steps. In our preliminary experiments, the algorithm typically halted after fewer than 10 steps on graphs with a few hundred thousand nodes. We remark that is it also possible to perform binary search over α to compute α_*, and then produce a set S attaining $\tilde{Q}_A(S) = \alpha_*$.

Proof. [Proof of Lemma 2.2] To prove assertion 1, it suffices to show that $D_A(S) \leq \min(\pi(S), \pi(V \setminus S))$ whenever $\pi(A) \leq \pi(\bar{A})$. It is easy to see that $D_A(S) \leq \pi(S)$, since we have

$$D_A(S) = \pi(S \cap A) - \pi(S \cap \bar{A}) f(A) \leq \pi(S),$$

so it remains to prove that $D_A(S) \leq \pi(V \setminus S)$. To prove this, notice that $D_A(S) + D_A(V \setminus S) = 0$, because

$$D_A(S)+D_A(V\setminus S) = D_A(V) = \pi(A)-\pi(\bar{A})f(A) = 0.$$

We now perform the following calculation, in which we use the fact that $f(A) = \pi(A)/\pi(\bar{A}) \leq 1$.

$$\begin{aligned} D_A(S) &= -D_A(V \setminus S) \\ &= -\pi((V \setminus S) \cap A) + \pi((V \setminus S) \cap \bar{A}) f(A) \\ &\leq \pi(V \setminus S) f(A) \\ &\leq \pi(V \setminus S). \end{aligned}$$

This completes the proof of Assertion 1. We remark that equality holds, with $\tilde{Q}_A(S) = Q(S)$, if and only if $S \subseteq A$.

To prove assertion 2, it suffices to show that $D_A(C) \geq \epsilon\pi(C)$ when A and C satisfy the assumption of the lemma.

$$\begin{aligned} \frac{D_A(C)}{\pi(C)} &= \frac{\pi(C \cap A)}{\pi(C)} - f(A)\frac{\pi(C \cap \bar{A})}{\pi(C)} \\ &\geq \left(\frac{\pi(A)}{\pi(V)} + \epsilon\frac{\pi(\bar{A})}{\pi(V)}\right) \\ &\quad - f(A)\left(1 - \frac{\pi(A)}{\pi(V)} - \epsilon\frac{\pi(\bar{A})}{\pi(V)}\right) \\ &= \epsilon\left(\frac{\pi(\bar{A})}{\pi(V)} + f(A)\frac{\pi(\bar{A})}{\pi(V)}\right) \\ &\quad + \left(\frac{\pi(A)}{\pi(V)} - \frac{\pi(A)}{\pi(\bar{A})} + \frac{\pi(A)}{\pi(\bar{A})}\frac{\pi(A)}{\pi(V)}\right) \\ &= \epsilon \cdot 1 + \frac{\pi(A)}{\pi(V)}\left(1 - \frac{\pi(V)}{\pi(\bar{A})} + \frac{\pi(A)}{\pi(\bar{A})}\right) \\ &= \epsilon. \end{aligned}$$

3 Experiments

Improve is provably at least as strong as the MQI algorithm (MQI is the parametric flow-based quotient cut improvement method discussed earlier [24]), and empirically it costs only a small constant factor more to run. For example, in the "hard graphs" experiment below, Improve required solving an average of four ordinary flow problems on the whole graph rather than the two parametric flow problems on half the graph which MQI requires. In the following experiments we focus on graphs whose best quotient cut has close to 50:50 balance, because that is the situation where MQI is weakest since it cannot move nodes in both directions, so paying the extra constant factor would be most justified.

We study 3 classes of graphs. First there are seven classic meshlike graphs from the Graph Partitioning Archive, so we can compare with the best solutions generated by many researchers. Second, there is a set of 1000 modified random geometric graphs that are hard enough to produce a wide spread between the performance of different algorithms. Finally, we study a small set of random graphs with planted cuts, a class which has been heavily studied by theorists.

3.1 Meshlike benchmark graphs Here we report results on seven meshlike benchmark graphs

		Improve vs MQI		
name	n_nodes	wins	ties	losses
wing	62032	3914	2378	0
fe_tooth	78136	1970	625	0
fe_rotor	99617	1794	241	0
598a	110971	1263	181	0
144	144649	999	14	0
wave	156317	1132	0	0
m14b	214765	605	27	0

name	best qcut found with at least 475:525 balance	compared to archive cut
wing	0.025504 = 791 / 31015	1.000032
fe_tooth	0.097809 = 3821 / 39066	0.992518
fe_rotor	0.041964 = 2004 / 47755	1.036868
598a	0.043219 = 2398 / 55485	1.000000
144	0.089733 = 6488 / 72303	0.999982
wave	0.111400 = 8702 / 78115	1.001702
m14b	0.035723 = 3836 / 107382	1.000000

Figure 1: Results for real-world meshlike graphs. Improve always beats or ties MQI, and the resulting qcuts are very competitive with the best nearly balanced cuts from the "graph partitioning archive.

downloaded from the Graph Partitioning Archive,[1] a web site that was set up by Chris Walshaw as a followup to the paper [33]. This web site keeps track of the best nearly-balanced cuts ever found by any algorithm for these graphs and others. By now at least 22 algorithms are represented on the site. These benchmark graphs were actually one of the original motivations for the present work; in 2004 we used Metis+MQI plus some additional hacks to find some new record cuts to post on the web site. The additional hacks were needed because MQI isn't naturally good at finding cuts with nearly perfect balance. The Improve algorithm in this paper makes it possible to find cuts of a similar quality in a much more principled way.

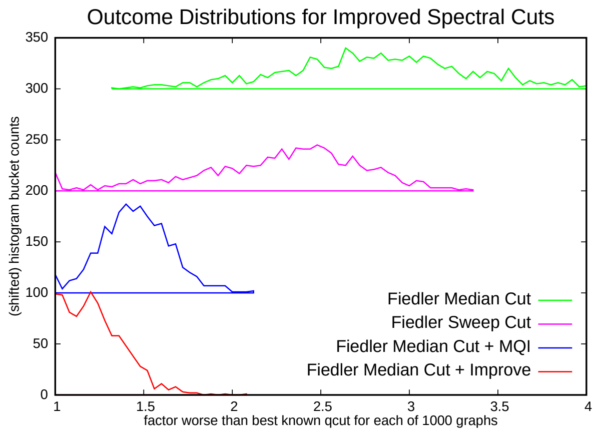
The seven graphs are listed in Figure 1. For each graph we ran a randomized version of Metis roughly a few thousand times to generate a set of initial balanced cuts. Then Improve and MQI were both run on each of these cuts. Figure 1(top) contains tabulations of the win/tie/loss scores of the two improvement methods on the various cuts for each of seven graphs. Improve always beats or ties MQI, which is consistent with Theorem 2.1, so both the theorem and our code pass this sanity check.

Figure 1(bottom) lists for each graph the best quotient cut found by any of these runs of Metis+Improve, subject to the constraint that the small side must contain at least 47.5 percent of the nodes. For comparison, we computed the quotient cut score of each of the four cuts posted on the web site (which are the smallest known cuts with at least 47.5, 48.5, 49.5, and 50 percent of the nodes on the small side), then divided the qcut score of our solution by the best of those four qcut scores to obtain the ratio listed in the final column of the table. Clearly, all of these ratios are very close to 1.0.

3.2 Graphs designed to be hard for all algorithms In this section we present results for seven different methods on a set of 1000 graphs drawn from a class of graphs that was specifically designed to be difficult in order to produce a wide spread of outcomes. Our graph generator produces a modified variant of the "random geometric graphs" which have been very popular in empirical graph partitioning experiments. Early papers like [20], which were studying local improvement methods, considered these graphs to be challenging. However, we have found that Leighton-Rao and Spectral with flow-based rounding can easily produce solutions that are probably within a few percent of optimal. Therefore we add some additional random edges to make the problem harder. Specifically, each of our graphs was generated as follows. First we choose ten thousand points from a uniform distribution on the unit disk (not the unit square). We generate a set of candidate edges consisting of each point and its 50 nearest neighbors. These candidate edges are then added to the graph in order from shortest to longest, stopping when the graph becomes connected. Finally, we add 2000 additional completely random edges. The resulting graphs contain 10000 nodes and an average of 58913 edges, for an average degree of 11.7826.

We point out that Improve can be viewed as a rounding method for extracting cuts from spectral embeddings. In this experiment four of the seven studied methods fall into the Spectral Embedding + Rounding Method category. The spectral embedding is specifically the ordering of the nodes induced by the second smallest eigenvector of the

[1] `http://staffweb.cms.gre.ac.uk/~wc06/partition`



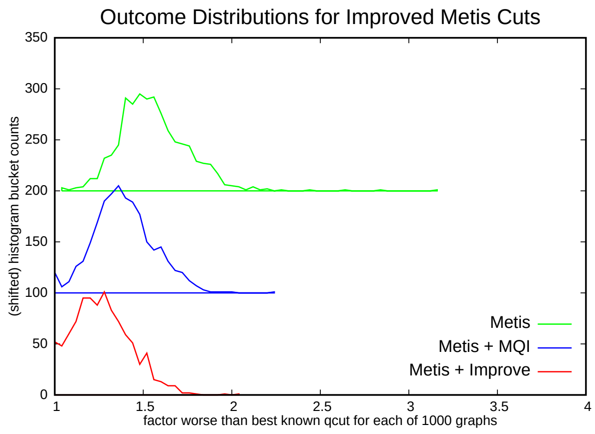


Figure 2: Solution quality distributions, for 7 algorithms on a set of 1000 hard graphs (random geometric plus random edges). The distributions for the four spectral rounding methods are strikingly different, with the new "Improve" algorithm working the best.

graph's (un-normalized) Laplacian matrix, *aka* the graph's Fiedler vector. The four rounding methods, in order of strictly increasing power, are MedianCut, SweepCut, MedianCut+MQI, and MedianCut+Improve. The remaining three methods in this experiment are, in order of strictly increasing power, Metis, Metis+MQI, and Metis+Improve. The previous two sentences contain 5 provable claims that one algorithm is strictly more powerful than another. The win-tie-loss tabulations in Figure 3 provide a sanity check on those claims and on our code. On these graphs, Improve required an average of 4 flow problems to be solved.

The histograms in Figure 2 show the distribution of solutions produced by each of the algorithms on our sample of 1000 graphs. Instead of plotting raw qcut scores, we divide the score of each solution by the best known score for each graph, which we found by running 3 additional, more expensive algorithms on each of the graphs.[2]

Evidently, there is a big difference in the power of the four rounding methods. Median cuts can be more than 4 times worse than the best known cuts, while Sweep cuts (to which most theoretical approximation guarantees apply) range from about 1 to 3 times worse than the best known. The new rounding method, Median+Improve, produces cuts ranging from about 1 to 1.5 times worse than the best known solutions.

[2]The additional algorithms included the implementation of Leighton-Rao described in [24], and the two heuristic methods 1) multi-try Metis+MQI+Midflow and 2) SDP embedding rounded with multi-try Midflow+MQI, that we previously used to find record cuts for the Graph Partitioning Archive. The SDP is the hypersphere embedding used by [2], but without the triangle inequalities on squared distances.

algorithm A	A wins	ties	B wins	algorithm B
SpecSwp	997	3	0	SpecMid
SpecMid+MQI	934	66	0	SpecSwp
SpecMid+Improve	928	72	0	SpecMid+MQI
Metis+MQI	1000	0	0	Metis
Metis+Improve	899	101	0	Metis+MQI

Figure 3: Win-tie-loss counts, for 7 algorithms on a set of 1000 hard graphs (random geometric plus random edges).

3.3 Random graphs with planted bisections

There are numerous theoretical results for random graphs with planted bisections [8, 5, 19, 12, 11]. On the whole, they indicate that spectral methods and local improvement methods should work well. In the following experiment, we tried several algorithms on a small collection of graphs with planted bisections of various sizes. The graphs were produced as follows. First specify a cutsize $c < 50000$. Then generate two

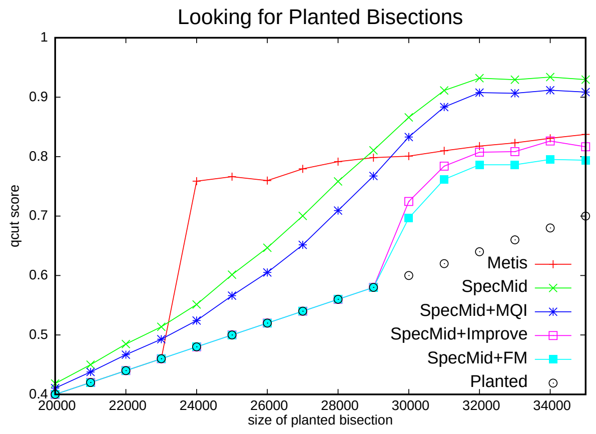


Figure 4: Results for random graphs with planted bisections. Here the new algorithm "Improve" is second best after Fidduccia-Mattheyses (local search is known to work well on random graphs).

50000-node random degree-4 graphs A and B, each by choosing 4 random matchings containing no duplicate edges. Then randomly choose a set of c nodes from graph A and c nodes from graph B, and connect them randomly. The resulting graph contains 100000 nodes and $200000 + c$ edges. We generated 16 graphs with c ranging from 20000 to 35000 in steps of 1000. In Figure 4 we plot the qcut score of the cut found by Metis, and of the Spectral Median Cut. Also plotted are the Spectral Median Cut improved by MQI, by Improve, and by Fidduccia-Mattheyses [14], a standard local improvement algorithm. The best results are SpecMid+FM, with SpecMid+Improve a close second. This outcome seems consistent with the body of theoretical work mentioned above.

References

[1] N. Alon and V. Milman. Isoperimetric inequalities for graphs and superconcentrators. *J. Combin. Theory B*, 38:73–88, 1985.

[2] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 222–231, New York, NY, USA, 2004. ACM Press.

[3] M. Balinski. On a selection problem. *Management Science*, (17):230–231, 1970.

[4] S. Bhatt and F. Leighton. A framework for solving vlsi graph layout problems. *J. Computer Systems Sciences*, 28:300–343, 1984.

[5] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis (extended abstract). In *FOCS*, pages 280–285, 1987.

[6] C. Borgs, J. T. Chayes, M. Mahdian, and A. Saberi. Exploring the community structure of newsgroups. In *KDD*, pages 783–787, 2004.

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *ICCV (1)*, pages 377–384, 1999.

[8] T. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. In *FOCS*, pages 181–192, 1984.

[9] J. Carrasco, D. Fain, K. Lang, and L. Zhukov. Clustering of bipartite advertiser-keyword graph, 2003.

[10] F. Chung. *Spectral graph theory*, volume Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[11] A. Dasgupta, J. E. Hopcroft, R. Kannan, and P. P. Mitra. Spectral clustering by recursive partitioning. In *ESA*, pages 256–267, 2006.

[12] T. Dimitriou and R. Impagliazzo. Go with the winners for graph bisection. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[13] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[14] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[15] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[17] A. Goldberg. Finding a maximum density subgraph. Technical Report UCB CSD 84/71, University of California, Berkeley, 1984.

[18] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 34–43, New York, NY, USA, 2003. ACM Press.

[19] M. Jerrum and G. B. Sorkin. Simulated annealing

for graph bisection. In *IEEE Symposium on Foundations of Computer Science*, pages 94–103, 1993.

[20] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Shevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[21] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[23] B. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–308, 1970.

[24] K. Lang and S. Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *IPCO*, pages 325–337, 2004.

[25] F. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *FOCS*, pages 422–431, 1988.

[26] M. Narasimhan and J. Bilmes. Local search for balanced submodular clusterings. In *IJCAI*, pages 981–986, 2007.

[27] S. B. Patkar and H. Narayanan. Improving graph partitions using submodular functions. *Discrete Appl. Math.*, 131(2):535–553, 2003.

[28] J.-C. Picard and M. Queyranne. Selected applications of minimum cuts in networks. *INFORM*, 20(4):394–422, 1982.

[29] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11:430–452, 1990.

[30] J. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, (17):200–207, 1970.

[31] D. Shmoys. Cut problems and their applications to divide-and-conquer, 1996.

[32] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[33] A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel approach to graph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 674–681, Las Vegas, Nevada, USA, 10-12 2000. Morgan Kaufmann.