Graph Partitioning using Single Commodity Flows

Rohit Khandekar*
University of Waterloo, Canada
rkhandekar@gmail.com

Satish Rao
University of California, Berkeley
satishr@cs.berkeley.edu

Umesh Vazirani
University of California, Berkeley
vazirani@cs.berkeley.edu

ABSTRACT

We show that the sparsest cut in graphs can be approximated within $O(\log^2 n)$ factor in $\tilde{O}(n^{3/2})$ time using polylogarithmic single commodity max-flow computations. Previous algorithms are based on multicommodity flows which take time $\tilde{O}(n^2)$. Our algorithm iteratively employs max-flow computations to embed an expander flow, thus providing a certificate of expansion. Our technique can also be extended to yield an $O(\log^2 n)$ (pseudo) approximation algorithm for the edge-separator problem with a similar running time.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

edge-separator, single commodity max-flow, sparse cut, spectral method

1. INTRODUCTION

Graph partitioning and clustering are fundamental problems that have been extensively studied both in theory and practice. Heuristics for graph partitioning constitute a central tool for coping with NP-completeness, and are used in applications of clustering ranging from computer vision, to data analysis, to learning. One version of graph partitioning is the sparsest cut problem. That is, given a graph $G = (V, E)$, find a set $S \subset V, |S| \leq |V|/2$ that minimizes the sparsity of S: $E(S, \overline{S})/|S|$.

There are two principal themes in theoretical work on graph partitioning. The first is based on spectral methods. The performance of these methods depends upon the sparsity of the graph: they are very fast and have good performance guarantees when the sparsity is very large [2, 6, 17]. The second theme dates back to the paper by Leighton and Rao [14] and uses linear programs related to multicommodity flow to approximate the sparsity to within $O(\log n)$. Until very recently this was the best approximation bound known.

More recently, the spectral and linear programming based approaches were combined by Arora, Rao, and Vazirani [4] to give a polynomial time semidefinite programming based algorithm that produces a cut whose sparsity is within a factor of $O(\sqrt{\log n})$ of optimal. The paper also introduced the framework of expander flows. The basic idea here is that the sparsity of the given graph G can be closely approximated by finding an expander that can be embedded into G with minimum congestion. Arora, Hazan, and Kale [3] gave an efficient multicommodity flow based implementation using the expander flow framework. They achieved running time $\tilde{O}(n^2)$ for an $O(\sqrt{\log n})$ approximation.

In this paper we give a new approach for finding graph separators that uses single commodity flows. We show how to find a cut whose sparsity is within an $O(\log^2 n)$ factor of optimal in $\tilde{O}(n^{3/2})$ time. While the approximation is worse than previous results, we continue to get a polylogarithmic approximation and break the $O(n^2)$ multicommodity flow barrier as well as get by the typically slow convergence rates of spectral methods. Moreover, we achieve this by computing only polylogarithmic single commodity flows thus achieving a speedup in theory, and perhaps even more dramatically in practice.

Finding a sparse cut using max-flow is easy if you can identify many vertices on each side of a good cut. The difficulty, of course, is in identifying these vertices. The way around this dilemma starts with the observation that the spectral method can efficiently find cuts that do not expand (i.e., constant expansion). Combining this with the expander flow formalism, our main task is to iteratively construct an expander that can be embedded in G with small congestion. The key to our algorithm is that once we identify a small cut in the currently embedded graph, improving its expansion while maintaining embeddability can be cast as a max-flow problem in the original graph G. The point being that now the two sides of the cut are specified and so the max-flow computation either routes many edges in G that cross this bad cut in the embedded graph or failing that it identifies a small cut in G.

To find a non-expanding cut, we give a new algorithm inspired by the spectral method. We note that we could also have used the spectral method directly, but this would not yield any running time improvement over multicommodity based algorithms. We could also use the algorithm of Spielman-Teng [16]. This would work but give worse approximation bounds. In fact, our resulting algorithm improves upon the algorithm of Spielman-Teng for finding balanced cuts.

*This work was done when the author was visiting UC Berkeley.

STOC'06, May21–23, 2006, Seattle, Washington, USA.

Algorithm	Output sparsity	Running time
Spectral [2]	$\phi^{1/2}$	n/ϕ^2
Spielman-Teng [16]	$\phi^{1/3}\log^3 n$	n/ϕ^3
Leighton-Rao [14]	$\phi \log n$	n^2
Arora-Rao-Vazirani [4]	$\phi\sqrt{\log n}$	poly(n)
Arora-Hazan-Kale [3]	$\phi\sqrt{\log n}$	n^2
this paper	$\phi \log^2 n$	$\min\{n^{3/2}, n/\phi\}$

Note: Here ϕ denotes the optimum sparsity and the running times ignore logarithmic factors. The spectral algorithm only yields a sparse cut, while the Spielman-Teng algorithm only yields a balanced separator.

Figure 1: Output sparsity and running time comparison of various algorithms for computing sparse cuts and balanced separators.

We summarize our results and previous results with regard to computing sparse and balanced cuts in Figure 1.

1.1 Heuristics

There is an extensive history of practical heuristics for graph partitioning. The earliest attempts dating back to the seventies were based on local search, with the Kernighan-Lin heuristic [11] achieving the best results in this class. Spectral methods (see for example [8, 9, 15]), which found widespread use in the eighties, compute eigenvectors, which are embeddings of the graph onto the real line. This extra information is often more useful than just the cut, since it can be easily combined with other criteria in clustering problems. Another advantage of this approach in practice was the easy availability of highly optimized code for computing eigenvectors. In the early nineties, the linear programming methods were shown to find better cuts [13] than Kernighan-Lin and the spectral method. But they were not adopted in practice since they were slow compared to Kernighan-Lin and did not provide the additional embedding information contained in the eigenvectors.

In the mid-nineties, a multilevel heuristic embodied in the package METIS was introduced [10]. This method is very fast and produces cuts that are almost as good as those obtained by linear programming based methods in practice. The heuristic proceeds by collapsing random edges until the resulting graph is quite small. It finds a good partition in this collapsed graph, and successively induce it up to the original graph, using local search. More recently [8] this multilevel approach has been used to speed up spectral methods, thereby obtaining the useful embedding information. We speculate that the random collapsing process is akin to process of adding matchings in our algorithm. This connection might provide further insight into such multilevel heuristics, and perhaps lead to a performance guarantee for some variant of the basic approach.

Very recently, a combination of spectral methods and single commodity flows have been used effectively in experiments [12]. Indeed, this combination seems to be quite effective for a large range of graphs – the resulting algorithms are slightly slower than METIS, but appear to give better cuts. One view of these algorithms is that they partially implement one iteration of an algorithm like ours, i.e., a phase of the spectral algorithm plus a max-flow computation to isolate a cut. It would be interesting to see if an iterative version of these methods would improve their performance in practice. The results of this paper suggest that this might be the case. Additionally, we note that our algorithm produces embeddings of the graph on the real line.

2. THE SPARSEST CUT PROBLEM

Let $G = (V, E, w)$ be an undirected graph with edge-weights $w_e \geq 0$. For a cut $(S, \overline{S})$ in G, where $\overline{S} = V \setminus S$), let $\delta(S)$ denote the set of edges with exactly one end-point in S and let $w(S, \overline{S}) = \sum_{e\in\delta(S)} w_e$ denote the total weight of such edges. The *expansion* (or *sparsity*) of a cut $(S, \overline{S})$ is defined as $\phi(S) = \frac{w(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$. The *sparsest cut problem* is to find a cut with minimum expansion. Let the expansion of G be $\phi(G) = \min_{(S,\overline{S})} \phi(S)$.

We call a graph an expander if its expansion is at least a constant. A comparison method of establishing a lower bound on the expansion of G is by "embedding" an expander in G with low congestion. More precisely, let H be an undirected graph with edge-weights $c_f \geq 0$ on the same set of vertices as G. The graph H can be thought of as an instance of the multicommodity flow problem in G with a demand c_f between the end-points of each edge $f \in H$. We say that H can be *embedded* (or *routed*) in G with congestion C if a flow of c_f units can be routed in G between the end-points of f *simultaneously* for all $f \in H$ without violating the edge-capacities w_e by a factor more than C. Observe that if an expander can be embedded in G with congestion C, then every cut in G has expansion at least $\Omega(1/C)$. Thus such an embedding can be thought of as a certificate of a lower bound on the expansion of G.

The main result of the paper is summarized in Theorem 2.1.

THEOREM 2.1. *Given an undirected graph $G = (V, E, w)$ with n vertices and m edges and an expansion parameter $\alpha > 0$, there is a randomized algorithm that with high probability*[1] *outputs*

- *either a cut $(S, \overline{S})$ of G with $\phi(S) \leq \alpha$,*
- *or an expander H on V that can be embedded in G with congestion at most $O((\log^2 n)/\alpha)$.*

Furthermore the algorithm can be implemented in $\tilde{O}(m + n^{3/2})$ time[2] *using $O(\log^2 n)$ single commodity max-flow computations.*

By doing a binary search on α, we can find a cut with expansion at most α_0 *and* an expander that can be embedded in G with congestion at most $O((\log^2 n)/\alpha_0)$ for some α_0.

3. THE ALGORITHM

At a high level our algorithm may be viewed as iteratively building an expander H to embed into G with congestion $O((\log^2 n)/\alpha)$. The notion of expander we have to use is special, and we start by defining it. Let $\{M_1, \ldots, M_t\}$ be a sequence of perfect matchings on a set of vertices V. The natural t-step random walk on V associated with this sequence of matchings proceeds as follows: in step i the particle stays put with probability $1/2$ and traverses the incident matched edge in M_i with probability $1/2$. The sequence $\{M_1, \ldots, M_t\}$ is *mixing* if for any starting position of the particle the probability that the particle reaches any vertex v is at least $1/2n$. This definition implies that the union of the matchings in a mixing sequence forms a graph (in which we give an edge a weight equal to the number of matchings it appears in) with edge expansion at least $1/2$.

We measure the progress of our algorithm by defining a potential function on a sequence of matchings $\{M_1, \ldots, M_t\}$, which

[1] Throughout the paper, we use the term "high probability" to mean probability at least $1 - n^{-C}$ for any given constant $C > 0$.

[2] $\tilde{O}(f)$ stands for $O(f \operatorname{polylog} f)$

is a measure of how far from uniform the resulting distribution of the associated random walk is when starting from a random vertex. Formally $\psi(t) = \sum_{i,j}(P_{ij}(t) - 1/n)^2$, where $P_{ij}(t)$ is the probability that a particle that starts at j reaches i in the walk associated with $\{M_1, \ldots, M_t\}$. Observe that for the empty sequence, $\psi(0) = n - 1$. If $\psi(t) \leq 1/4n^2$ then the sequence $\{M_1, \ldots, M_t\}$ is mixing. Our algorithm will start with the empty sequence, and while the sequence is not mixing, it tries to find a new matching to add to the sequence, which is embeddable in G with congestion $1/\alpha$ and which reduces the potential by a factor of $(1 - \Omega(1/\log n))$. If it succeeds in doing this, $O(\log^2 n)$ iterations suffice to produce a mixing sequence of matchings. In this case, the union of the matchings which forms a $1/2$-edge expander can be embedded in G with congestion $O((\log^2 n)/\alpha)$. In case the algorithm does not succeed in some iteration, it outputs a cut in G with expansion at most α.

We proceed by describing the implementation of an iteration in this procedure. In more detail, if the current sequence of matchings is $\{M_1, \ldots, M_t\}$, an iteration proceeds as follows.

1. We find a bisection $(S, \overline{S})$ of V, such that adding any prefect matching M_{t+1} between S and $\overline{S}$ to $\{M_1, \ldots, M_t\}$ reduces the potential function, in expectation, by a factor of $1 - \Omega(1/\log n)$. That is, $\mathbb{E}[\psi(t+1)] \leq (1 - \Omega(1/\log n))\psi(t)$.
2. We give a maximum flow based procedure that either
   (a) produces a perfect matching M_{t+1} between S and $\overline{S}$ which can be embedded in G with congestion $1/\alpha$,
   (b) or finds a cut in G of expansion at most α.

The procedure for step 1 is the following natural variant of the spectral method.

FINDBISECTION: We choose a random unit n-dimensional vector r orthogonal to $\mathbf{1}$, and compute $u = \mathcal{M}_t(\mathcal{M}_{t-1}(\cdots(\mathcal{M}_1 r)))$. We form S by choosing the $n/2$ smallest values of u.

Here $\mathbf{1}$ denotes the vector of all 1s and an $n \times n$ matrix

$$\mathcal{M}_p(i,j) = \begin{cases} 1/2 & \text{if } i = j \text{ or } (i,j) \in M_p, \\ 0 & \text{otherwise,} \end{cases}$$

represents the probability transition matrix of the pth step of the natural random walk.

To get some intuition about the above procedure consider the following discrete version (which can also be analyzed): assign to each vertex an initial weight of $\pm 1/n$, where the signs are chosen by the flip of a fair coin. Now average these weights according to the natural random walk associated with $\{M_1, \ldots, M_t\}$ defined above. Intuitively, the weights rapidly achieve their average within each "well-connected" component. These average weights for each component are typically non-zero (either positive or negative) because of the random initial weight assignments. Now selecting the $n/2$ smallest weights should help separate some of these "well-connected" components from the rest (say the negative from the positive). Our actual analysis follows a different line, as is based on the properties of random projections.

The procedure for Step 2 above is quite standard and is described in Section 3.2.

3.1 Analysis of FINDBISECTION

It is tempting to analyze FINDBISECTION by following the evolution of the random vector as matchings are successively applied to it. The key to our analysis is to lift this process by following random walks starting from each vertex, and then projecting the resulting distributions onto the randomly chosen initial vector.

To describe the lifting, consider a random walk from each vertex j, and recall that $P_{ij}(t)$ is the probability of going from j to i in the natural random walk associated with $\{M_1, \ldots, M_t\}$. In the lifted process, the vector $(P_{i1}, \ldots, P_{in})$ denotes the probability of ending up at i starting from each vertex. We denote this vector by $P_i(t)$. Observe that the entries in $P_i(t)$ sum up to 1. This follows by induction since the natural random walk averages the vectors $P_i(t-1)$ and $P_j(t-1)$ to obtain $P_i(t) = P_j(t)$ for each $(i,j) \in M_t$.

Next we observe that the vector u produced by FINDBISECTION on $\{M_1, \ldots, M_t\}$ is the projection of the vectors $P_i(t)$ onto the randomly chosen vector $r \perp \mathbf{1}$, i.e., $u_i = P_i(t) \cdot r$. FINDBISECTION partitions the vertices i into two sets according to whether the corresponding u_i's are large or small. Thus in any matching respecting this partition, if vertices i and j are matched, $|u_i - u_j|$ will tend to be large. Note that $u_i - u_j$ is the projection of $P_i(t) - P_j(t)$ on a random vector r. Since the vector $P_i(t) - P_j(t)$ lies in the $(n-1)$-dimensional space orthogonal to $\mathbf{1}$ and r is a unit vector chosen uniformly at random from this space, we have $\mathbb{E}[|u_i - u_j|^2] = \|P_i(t) - P_j(t)\|^2/(n-1)$. Indeed, using the fact that the length of a vector has a Gaussian distribution under random projection, we will show, with high probability, that for any pair of vertices i, j, we have $\|P_i(t) - P_j(t)\|^2 \geq |u_i - u_j|^2 \cdot (n-1)/C \log n$ for a constant C. Thus for any perfect matching M between S and $\overline{S}$,

$$\sum_{(i,j)\in M} \|P_i(t) - P_j(t)\|^2 \geq \frac{n-1}{C \log n} \sum_{(i,j)\in M} |u_i - u_j|^2.$$

We can thus establish the desired bound by noting that the left hand side above corresponds to the reduction in potential and that $(n-1)\sum_{(i,j)\in M} |u_i - u_j|^2$ corresponds to the total potential. The former follows from the fact that the natural random walk averages the vectors $P_i(t)$ and $P_j(t)$ to obtain $P_i(t+1) = P_j(t+1)$. The latter follows from the fact that $\sum_i \mathbb{E}[|u_i|^2]$ is proportional to $\psi(t)$.

We establish the facts used in the above outline in the following lemmas.

LEMMA 3.1. *The reduction in potential for a perfect matching M is*

$$\frac{1}{2} \sum_{(i,j)\in M} \|P_i(t) - P_j(t)\|^2.$$

LEMMA 3.2. *With high probability, for all pairs i, j,*

$$\|P_i(t) - P_j(t)\|^2 \geq \frac{(n-1)}{C \log n} \cdot |u_i - u_j|^2$$

for a constant $C > 0$.

LEMMA 3.3. *For any perfect matching M that respects the partition found by* FINDBISECTION, *we have*

$$(n-1)\mathbb{E}\left[\sum_{(i,j)\in M} |u_i - u_j|^2\right] \geq \psi(t).$$

The inequality in Lemma 3.2 holds with probability at least $1 - n^{-\Omega(1)}$. Therefore these three lemmas together imply that the expected reduction in the potential in any iteration is $\Omega(1/\log n)$ fraction of the current potential. This implies that in $O(\log n)$ iterations the potential reduces by half with high probability. Thus, with

high probability, in $O(\log^2 n)$ iterations, the potential drops below $1/4n^2$ and the union of the matchings gives the desired expander. We proceed by proving the lemmas above.

PROOF OF LEMMA 3.1. Recall that $P_{ik}(t)$ denotes the probability of going from k to i in the natural random walk associated with $\{M_1, \ldots, M_t\}$. Suppose now that we add a matching M to this sequence. Let $(i,j) \in M$ be a matched edge. If the particle starting at k is currently present at i (resp. j), it stays put with probability $1/2$ and jumps to j (resp. i) with probability $1/2$. Thus the resulting probability of reaching i or j is given by $(P_{ik}(t) + P_{jk}(t))/2$ and the resulting distributions are given by

$$P_i = P_j = \frac{P_i(t) + P_j(t)}{2}.$$

Recall that the potential is given by $\psi = \sum_i \|P_i - \mathbf{1}/n\|^2$. Since for any two vectors u and v, we have $\|u\|^2 + \|v\|^2 - 2\|\frac{u+v}{2}\|^2 = \frac{1}{2}\|u - v\|^2$, the reduction in the potential contributed by i and j where $(i,j) \in M$ is given by

$$\frac{1}{2}\|(P_i(t) - \mathbf{1}/n) - (P_j(t) - \mathbf{1}/n)\|^2 = \frac{1}{2}\|P_i(t) - P_j(t)\|^2.$$

Thus the overall reduction is $\frac{1}{2}\sum_{(i,j)\in M} \|P_i(t) - P_j(t)\|^2$. □

PROOF OF LEMMA 3.2. In the tth step of the random walk, for $(i,j) \in M_t$, the vectors $P_i(t)$ and $P_j(t)$ both take the average value $(P_i(t-1) + P_j(t-1))/2$. By induction, it is easy to see that the entries in $P_i(t)$ add up to 1. Thus the vectors $P_i(t) - P_j(t)$ lie in the $(n-1)$-dimensional space orthogonal to $\mathbf{1}$. Now in the next iteration, we pick a unit random vector r in this space and let $u_i = P_i(t) \cdot r$ be the projection of $P_i(t)$ onto r. Clearly $u_i - u_j$ denotes the projection of $P_i(t) - P_j(t)$ onto r. We now use the following lemma about the random projections.

LEMMA 3.4 (GAUSSIAN BEHAVIOR OF PROJECTIONS). *If v is a vector of length l in $\mathbb{R}^d$ and r is a random unit vector in $\mathbb{R}^d$, then*

- $\mathbb{E}[(v^T r)^2] = l^2/d$, *and*
- *for $x \le d/16$, we have $Pr[(v^T r)^2 \ge x l^2/d] \le e^{-x/4}$.*

By setting $x = C \log n$ for a sufficiently large constant $C > 0$, we get that for any pair i, j,

$$Pr\left[(u_i - u_j)^2 \ge C \log n \cdot \frac{\|P_i(t) - P_j(t)\|^2}{n-1}\right] \le n^{-C/4}.$$

Therefore by union bound, with high probability,

$$\|P_i(t) - P_j(t)\|^2 \ge \frac{(n-1)}{C \log n} \cdot |u_i - u_j|^2$$

holds for all pairs i, j. □

PROOF OF LEMMA 3.3. Since the perfect matching M respects the partition found by FINDBISECTION, it matches the vertices in S to those in $\overline{S}$. Recall that S is the set of $n/2$ vertices i with smallest values of u_i. Let η be a real number so that $u_i \le \eta \le u_j$ for any $i \in S$ and $j \in \overline{S}$. We then have

$$\begin{aligned}
\sum_{(i,j)\in M} |u_i - u_j|^2 &\ge \sum_{(i,j)\in M} \left((u_i - \eta)^2 + (\eta - u_j)^2\right) \\
&= \sum_{i \in V} (u_i - \eta)^2 \\
&= \sum_{i \in V} u_i^2 - 2\eta \sum_{i \in V} u_i + n\eta^2 \\
&\ge \sum_i u_i^2.
\end{aligned}$$

The last inequality follows from the fact that $\sum_i u_i = \sum_i P_i(t) \cdot r = \mathbf{1}^T r = 0$ which, in turn, is true since the vectors $P_i(t)$ add up to $\mathbf{1}$ and the random vector r was chosen to be orthogonal to $\mathbf{1}$.

Now we use Lemma 3.4 for vectors $(P_i(t) - \mathbf{1}/n)$ which lie in the $(n-1)$-dimensional space orthogonal to $\mathbf{1}$. Note that $u_i = P_i(t) \cdot r = (P_i(t) - \mathbf{1}/n) \cdot r$ denotes the projection of $(P_i(t) - \mathbf{1}/n)$ onto r. Since r is a random unit vector from the space orthogonal to $\mathbf{1}$, we have

$$\mathbb{E}[u_i^2] = \frac{\|P_i(t) - \mathbf{1}/n\|^2}{n-1}.$$

Putting the pieces together, we get

$$\begin{aligned}
\mathbb{E}\left[\sum_{(i,j)\in M} |u_i - u_j|^2\right] &\ge \mathbb{E}\left[\sum_{i \in V} u_i^2\right] \\
&= \frac{\sum_{i \in V} \|P_i(t) - \mathbf{1}/n\|^2}{n-1} \\
&= \frac{\psi(t)}{n-1}.
\end{aligned}$$

as claimed. □

3.2 Finding a Matching or a Cut

To simplify the presentation, we assume that G is unweighted. We use the following procedure for routing $n/2$ units of flow between S and $\overline{S}$.

> CUTORFLOW:
> We form a flow network from G and S as follows: assign each edge in G a capacity of $1/\alpha$ (which we assume to be integral), add a source node with an outgoing unit-capacity arc to each vertex in S, and add a sink node with an incoming unit-capacity arc from each vertex in $\overline{S}$.
> We then find the maximum flow between the source and the sink. If the flow-value is $n/2$, we produce a matching between S and $\overline{S}$ by decomposing the flow into flow-paths in a standard manner (see, e.g., [1]). If not, we find a minimum cut separating the source and the sink in the flow network and output the partition induced on V.

If a maximum flow of value $n/2$ is found, the procedure finds a matching between S and $\overline{S}$ that can be embedded in G with congestion $1/\alpha$. If not, the following lemma ensures that the procedure outputs a cut of expansion at most α.

LEMMA 3.5. *If the maximum flow between the source and the sink has value less than $n/2$, then the procedure outputs a cut in G of expansion less than α.*

PROOF. If the flow has value less than $n/2$, by the max-flow min-cut theorem for single commodity flows, the minimum cut separating the source and the sink has capacity less than $n/2$. Let the number of edges in the cut incident to the source (resp. sink) be n_s (resp. n_t). The remaining capacity of the cut is less than $n/2 - n_s - n_t$, and thus uses at most $\alpha(n/2 - n_s - n_t)$ edges in the original graph. Moreover, the cut consisting of edges in the graph separates at least $n/2 - n_s$ vertices in S from $n/2 - n_t$ vertices in $\overline{S}$. The expansion of this cut is at most $\alpha(n/2 - n_s - n_t)/\min(n/2 - n_s, n/2 - n_t)$ which is at most α. □

3.3 Running time

In the previous sections, we argued that the algorithm terminates in $O(\log^2 n)$ iterations with high probability. In each iteration,

step 1 is implemented by the procedure FINDBISECTION. FINDBISECTION involves picking a random vector $r \perp \mathbf{1}$ and computing $u = \mathcal{M}_t(\mathcal{M}_{t-1}(\cdots(\mathcal{M}_1 r)))$. Multiplication by $\mathcal{M}_p$ can be done in linear time simply by averaging the values at i and j for each $(i,j) \in M_p$. Thus step 1 can be implemented in $\tilde{O}(n)$ time. In step 2, we do a single-commodity max-flow computation in a flow network with $O(m)$ edges. This can be done in $\tilde{O}(m^{3/2})$ time using an algorithm of Goldberg and Rao [7]. They show that a cut and a flow with values within a factor of $(1+\epsilon)$ of each other can be computed in time $\tilde{O}(m^{3/2}\log(1/\epsilon))$. Since we are looking for an integral flow of value $n/2$, we can set $\epsilon = \Omega(1/n)$ and still get an optimum flow and a cut. A decomposition of the flow into flow-paths can be accomplished in $\tilde{O}(m)$ time using dynamic trees.[3] The running time of step 2 is thus $\tilde{O}(m^{3/2})$. We can reduce the dependence on the number of edges by using a standard sparsification as a pre-processing step.

THEOREM 3.6 (BENCZÚR AND KARGER [5]). *Given a graph G with n vertices and m edges and an error parameter $\epsilon > 0$, there is a graph $\hat{G}$ such that*

- *$\hat{G}$ has $O((n\log n)/\epsilon^2)$ edges and*
- *the value of every cut in $\hat{G}$ is within $(1 \pm \epsilon)$ times the value of the corresponding cut in G.*

$\hat{G}$ can be constructed in $O(m\log^2 n)$ time if G is unweighted and in $O(m\log^3 n)$ time if G is weighted.

Since we are interested in an approximate sparsest cut, we can pre-process G to reduce the number of edges to $\tilde{O}(n)$ while preserving values of all the cuts within a constant factor. This step takes $\tilde{O}(m)$ time. The overall algorithm can thus be implemented in $\tilde{O}(m + n^{3/2})$ time.

Furthermore, if ϕ denotes the expansion of the graph G, the flow-paths in the max-flow computations will be of length $O((\log n)/\phi)$ and the max-flow can be computed in time $\tilde{O}(n/\phi)$.

4. THE MINIMUM SEPARATOR PROBLEM

Given a weighted graph $G = (V, E, w)$ on n vertices, the minimum edge-separator problem is to find a cut $(S, \overline{S})$ in G such that $|S|, |\overline{S}| \geq n/3$ and $w(S,\overline{S}) = \sum_{e\in\delta(S)} w_e$ is minimized. In this section, we show how the techniques developed in Section 3 can be used to get a $O(\log^2 n)$-(pseudo-)approximation for this problem in a similar running time.

We call a cut $(S, \overline{S})$ b-balanced if $|S|, |\overline{S}| \geq bn$. Clearly minimizing $w(S,\overline{S})$ over $1/3$-balanced cuts is, within a constant factor, equivalent to minimizing the expansion $\phi(S) = \frac{w(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$ over the $1/3$-balanced cuts. Following is the main theorem of this section.

THEOREM 4.1. *Given an undirected graph $G = (V, E, w)$ with n vertices and m edges, an expansion parameter $\alpha > 0$, and a constant $b < 1/2$, there is a randomized algorithm that with high probability outputs*

- *either an $\Omega(1/\log^2 n)$-balanced cut $(S, \overline{S})$ of G with $\phi(S) \leq \alpha$,*
- *or a graph H on V that can be embedded in G with congestion at most $O((\log^2 n)/\alpha)$ and in which every b-balanced cut has expansion at least a constant.*

Furthermore the algorithm can be implemented in $\tilde{O}(m + n^{3/2})$ time using $O(\log^2 n)$ single commodity max-flow computations.

The above algorithm can be used iteratively $O(\log^2 n)$ times to find either a $1/6$-balanced cut with expansion at most $O(\alpha)$ or a graph H on V that can be embedded in G with congestion at most $O((\log^2 n)/\alpha)$ and in which every $1/3$-balanced cut has expansion at least a constant.

4.1 The algorithm

Our algorithm for the minimum separator problem builds on the algorithm presented in Section 3. We run that algorithm with expansion parameter α. If it outputs an expander, this expander satisfies the conditions in Theorem 4.1. Suppose, therefore, that in step 2 of some iteration t, it computes a cut $(S, \overline{S})$ with expansion at most α. If this cut is $(1/C\log^2 n)$-balanced (for some constant C), we output this cut and stop. If on the other hand, the cut is not $(1/C\log^2 n)$-balanced, we know that the algorithm must have computed a flow of value at least $n/2 - n/C\log^2 n$ in that iteration. This flow corresponds to a matching M_t between S and $\overline{S}$ of size at least $n/2 - n/C\log^2 n$. We now *arbitrarily* match the remaining vertices in S to the remaining vertices $\overline{S}$ and call this matching M'_t. We then add the perfect matching $M_t \cup M'_t$ to our sequence of prefect matchings and go on to the next iteration. Note that although the matching M'_t need *not* be embeddable in G, its size is at most $n/C\log^2 n$.

If the algorithm computes a $(1/C\log^2 n)$-balanced cut with expansion at most α in any iteration, we output this cut and stop. Let us assume, therefore, that the algorithm never outputs such a cut. Thus after $N = O(\log^2 n)$ iterations, it outputs an expander H which is the union of the matchings $M_t \cup M'_t$ found in iterations t. Let H' be the union of the matchings M_t that are embeddable in G each with congestion at most $1/\alpha$. Thus H' can be embedded in G with congestion $O((\log^2 n)/\alpha)$. We now argue that every b-balanced cut in H' has constant expansion. Note that H' can be augmented with at most $Nn/C\log^2 n$ edges in $\bigcup_t M'_t$ to get an expander H. Since H has $\Omega(n)$ edges across any b-balanced cut for a given constant b, such a cut in H' must already have $\Omega(n) - Nn/C\log^2 n$ edges going across it. If we take C to be a sufficiently large constant (depending on b), this quantity can be made $\Omega(n)$. Thus every b-balanced cut in H' has a constant expansion.

5. CONCLUSIONS

We presented algorithms for graph partitioning problems that use single commodity flows. Our algorithms iteratively route flows across $O(\log^2 n)$ cuts in the graph and embed an expander with congestion within $O(\log^2 n)$ times the optimum. In case of the sparsest cut problem, the union of the flows, in fact, corresponds to an embedding of a *complete graph*. Thus it also computes an $O(\log^2 n)$ approximation to the uniform concurrent multicommodity flows in time $\tilde{O}(m + n^{3/2})$. This approach can also be generalized to the *product* multicommodity flows where the demand between a pair of vertices is the product of the weights of those vertices.

It will be interesting to see if this approach can yield a (tight) $O(\log n)$ approximation for embedding a complete graph or even a $O(\sqrt{\log n})$ approximation algorithm by embedding an arbitrary expander.

[3] If one wishes to avoid using dynamic trees, the algorithm can be modified in standard ways such that the total volume of the flow is linear in the number of edges, in which case a depth-first-search will decompose the flow quickly.

Even more interesting is a possible connection between our algorithm and the random collapsing process that underlies METIS, the widely used heuristic for graph partitioning. Is it possible to rigorously analyze a METIS-like heuristic based on the techniques introduced in this paper?

6. REFERENCES

[1] R. Ahuja, T. Magnati, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Eaglewood Cliffs, NJ, 1993.

[2] N. Alon and V. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38:73–88, 1985.

[3] S. Arora, E. Hazan, and S. Kale. $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 238–247, 2004.

[4] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings, and graph partitioning. In *Proceedings, ACM Symposium on Theory of Computing*, pages 222–231, 2004.

[5] A. Benczúr and D. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings, ACM Symposium on Theory of Computing*, pages 47–55, 1996.

[6] W. Donath and A. J. Hoffman. Lower bounds for partitioning of graphs. *IBM J. Res. Develop.*, 17:420–425, 1973.

[7] A. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45:783–797, 1998.

[8] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Stat. Comput.*, 16(2):452–469, 1995.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI design. In *Proc. ACM/IEEE Design Automation Conference*, pages 526–529, 1997.

[10] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359 – 392, 1999.

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

[12] K. Lang. Finding good nearly balanced cuts in power law graphs. Manuscript, 2004.

[13] K. Lang and S. Rao. Finding near-optimal cuts: An empirical evaluation. In *Symposimum on Discrete Algorithms*, pages 212–221, 1993.

[14] F. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[15] H. Simon, A. Pothen, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Theory and Appl.*, 11(3):430–452, 1990.

[16] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings, ACM Symposium on Theory of Computing*, pages 81–90, 2004.

[17] R. Tanner. Explicit concentrators from generalized n-gons. *SIAM J. Alg. Disc. Methods*, 5:287–293, 1984.