Nearly-Linear Time Algorithms for Graph Partitioning, Graph Sparsification, and Solving Linear Systems

[Extended Abstract]

Daniel A. Spielman
Department of Mathematics
M.I.T.
Cambridge, MA 02139
spielman@math.mit.edu

Shang-Hua Teng
Department of Computer Science
Boston University
Boston, MA 02215
and, Akamai Technologies Inc.
steng@cs.bu.edu

ABSTRACT

We present algorithms for solving symmetric, diagonally-dominant linear systems to accuracy ϵ in time linear in their number of non-zeros and $\log(\kappa_f(A)/\epsilon)$, where $\kappa_f(A)$ is the condition number of the matrix defining the linear system. Our algorithm applies the preconditioned Chebyshev iteration with preconditioners designed using nearly-linear time algorithms for graph sparsification and graph partitioning.

Categories and Subject Descriptors

F.2.1 [**Numerical Algorithms and Problems**]: Computations on matrices; G.1.3 [**Numerical Linear Algebra**]: Linear systems (direct and iterative methods)

General Terms

Algorithms, Theory

Keywords

Preconditioners, Graph Partitioning, Graph Sparsification

1. INTRODUCTION

We present a linear-system solver that, given an n-by-n symmetric diagonally-dominant matrix A with m non-zero entries and an n-vector $\boldsymbol{b}$, produces a vector $\tilde{\boldsymbol{x}}$ satisfying $\|A\tilde{\boldsymbol{x}} - \boldsymbol{b}\| < \epsilon$ and $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\| \leq \epsilon$, where $\boldsymbol{x}$ is the solution to $A\boldsymbol{x} = \boldsymbol{b}$, in time

$$m \log^{O(1)} m + O\left(\log(\kappa_f(A)/\epsilon)\left(m + n2^{O(\sqrt{\log n \log\log n})}\right)\right),$$

where $\kappa_f(A)$ is the ratio of the largest to the smallest non-zero eigenvalue of A.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

Our algorithm exploits two novel tools. The first is a nearly-linear time algorithm, `Partition`, for quickly computing crude graph partitions. For any graph G having a cut of sparsity ϕ and balance b, this algorithm outputs a cut of sparsity at most $O(\phi^{1/3}\log^{O(1)} n)$ and balance $\Omega(b)$ in time $m((\log m)/\phi)^{O(1)}$.

Using this graph partitioning algorithm, we design fast graph sparsifiers and graph ultra-sparsifiers. We say that a graph is d-sparse if it has at most dn edges. We say that a graph is k-ultra-sparse if it has at most $n-1+k$ edges, and note that a spanning tree is 0-ultra-sparse. We say that a weighted graph $\tilde{A}$ γ-approximates a weighted graph A if

$$\mathcal{L}(\tilde{A}) \preccurlyeq \mathcal{L}(A) \preccurlyeq \gamma\mathcal{L}(\tilde{A}),$$

where $\mathcal{L}(A)$ is the Laplacian of A (the diagonal matrix of the weighted degrees of A minus the adjacency matrix of A) and $X \preccurlyeq Y$ means that for all $\boldsymbol{x} \in \mathbb{R}^n$,

$$\boldsymbol{x}^T X \boldsymbol{x} \leq \boldsymbol{x}^T Y \boldsymbol{x}.$$

On input a weighted graph A, `Sparsify`(A, β) outputs a graph $\tilde{A}$ that is $O(n\log^{O(1)}(n/\beta)/\beta^2)$-sparse that $(1+\beta)$-approximates A with high probability. Similarly, `Ultra-Sparsify`(A, k) outputs a graph $\tilde{A}$ that is $kn^{o(1)}$-ultra sparse that n/k-approximates A with high probability. Both algorithms run in time $m\log^{O(1)} m$.

For convenience, if E and $\widetilde{E}$ are sets of weighted edges, we write

$$E \preccurlyeq \widetilde{E}$$

if $\mathcal{L}(A) \preccurlyeq \mathcal{L}(\tilde{A})$ where A and $\tilde{A}$ are the corresponding graphs.

Due to space limitations, we omit almost all proofs and refer the reader to the full version of the paper [24].

1.1 Solving Linear Systems

Our linear system solvers exploit the preconditioned inexact Chebyshev method [12][1]. Given symmetric positive semi-definite matrices A and B, the preconditioned Chebyshev method finds ϵ-accurate solutions to $A\boldsymbol{x} = \boldsymbol{b}$ in time

$$O(m\sqrt{\kappa_f(A,B)}S(B)\log(\kappa_f(A)/\epsilon)),$$

[1]We could use the preconditioned Conjugate Gradient for the one-shot algorithms, we cannot use it for our recursive algorithms as we can not find of a strong enough analysis of the accuracy of the solutions obtained by CG if the inner system is solved inaccurately.

where m is the number of non-zeros in A, $S(B)$ is the time it takes to solve systems in B, and

$$\kappa_f(A,B) = \left(\max_{x:Ax\neq 0} \frac{\boldsymbol{x}^T A\boldsymbol{x}}{\boldsymbol{x}^T B\boldsymbol{x}}\right)\left(\max_{x:Ax\neq 0} \frac{\boldsymbol{x}^T B\boldsymbol{x}}{\boldsymbol{x}^T A\boldsymbol{x}}\right),$$

for symmetric A and B with $\mathbf{Span}(A) = \mathbf{Span}(B)$.

Vaidya [25] had the remarkable idea of using a subgraph of A as a preconditioner. In particular, Vaidya proved that a maximum spanning tree of A nm-approximates A and that by adding t^2 edges to such a tree, one can obtain a t^2-ultra-sparse graph that $O(nm/t^2)$-approximates A. In the case of planar graphs, Vaidya only needed to add $O(t)$ edges. Vaidya thereby obtained algorithms for solving SDD linear systems with non-positive off-diagonals of degree d in time $O((dn)^{1.75}\log(\kappa_f(A)/\epsilon))$, and for solving planar systems in time $O((dn)^{1.2}\log(\kappa_f(A)/\epsilon))$. Before Vaidya's contribution, the only worst-case bounds for solving such systems required time $O(nm)$ and $O(n^{1.5})$ respectively [19]. While Vaidya's work was unpublished, proofs of his results as well as extensions may be found in [16, 13, 14, 4, 5, 6]. Two ways of extending Vaidya's construction to systems with both positive and negative off-diagonals were found: a direct method [5] and a transformation from such a system to one with non-positive off-diagonals [13]. There is also a transformation from the problem of preconditioning a SDD system to that of preconditioning a system in which the diagonals are precisely the sums of the absolute values of the off-diagonals in their columns. Thus, it generally suffices to consider preconditioning Laplacian matrices.

By recursively applying Vaidya's preconditioners, Joshi [16] showed how to solve a system where A is a regular grid in time $O(n\log(n/\epsilon))$. Reif [22] recursively applied Vaidya's preconditioners to improve the running time for constant-degree planar linear systems to $O(n^{1+\beta}\log^{O(1)}(\kappa_f(A)/\epsilon))$, for every $\beta > 0$. Boman and Hendrickson [6, 7] applied the trees of [2] to construct 0-ultra-sparse $m^{1+o(1)}$-approximations of A, and showed that these could be used to solve arbitrary SDD systems in time $m^{1.5+o(1)}\log(\kappa_f(A)/\epsilon)$. Exploiting techniques that add vertices and edges to the graph, Maggs, *et. al.* [21] improved this time to $O(mn^{1/2}\log^2(n\kappa_f(A)/\epsilon))$, after some preprocessing. Spielman and Teng [23] augmented Boman and Hendrickson's construction to obtain $O(t^2\log n)$-ultra-sparse graphs that $(m^{1+o(1)}/t)$-approximate the original, resulting in a $m^{1.31+o(1)}\log(\kappa_f(A)/\epsilon)$ time algorithm.

In this work, we augment the low-stretch spanning trees of Alon, Karp, Peleg and West [2] to obtain $tn^{o(1)}$-ultra-sparse graphs that $\left((n/t)\log^{O(1)} n\right)$-approximate A for all $t \geq 1$. Our linear system solver is obtained immediately by plugging this ultra-sparsifier construction into the recursive algorithm of [23].

1.2 Sparsifiers

While the analysis in this paper may be long, the idea behind the construction of our sparsifiers is quite simple: we show that if a graph A has no sparse cuts, then a natural random rounding of A will be a good approximation of A. Thus, to approximate a general graph A, we would like to remove a small fraction of the edges of A so that each remaining component has no sparse cuts. We then sparsify each of these components via a random rounding, and then apply the algorithm recursively to the edges we removed. Thus, to make the algorithm efficient, we need merely find a fast algorithm for removing those edges. This turns out to be tricky. The other part—proving that the random rounding of a graph with no sparse cuts is a good approximation of the original—is cleanly accomplished in Section 5 by adapting techniques of Füredi and Komlós [11].

The graph sparsifiers most closely related to ours are those developed by Benczur and Karger [3]. They develop an $O(n\log^3 n)$ time algorithm that on input a weighted graph G with Laplacian L and a parameter ϵ outputs a weighted graph $\tilde{G}$ with Laplacian $\tilde{L}$ such that $\tilde{G}$ has $O(n\log n/\epsilon)$ edges and such that for all $\boldsymbol{x} \in \{0,1\}^n$

$$\boldsymbol{x}^T\tilde{L}\boldsymbol{x} \leq \boldsymbol{x}^T L\boldsymbol{x} \leq (1+\epsilon)\boldsymbol{x}^T\tilde{L}\boldsymbol{x}. \quad (1)$$

The difference between their sparsifiers and ours is that ours apply for all $\boldsymbol{x} \in \mathbb{R}^n$. To see the difference between these two types of sparsifiers, consider the graph on vertex set $\{0,\ldots,n-1\}$ containing edges between each pair of vertices i and j such that $|(i-j)| \mod n \leq k$, and one additional edge, e, from vertex 0 to vertex $n/2$. If $\tilde{G}$ is the same graph without edge e, then (1) is satisfied with $\epsilon = 1/k$ for all $\boldsymbol{x} \in \{0,1\}^n$. However, for the vector $\boldsymbol{x} = (0,1,2,\ldots n/2-1, n/2, n/2-1,\ldots,1,0)$, (1) is not satisfied for any $\epsilon < n/4k$. Moreover, the algorithm of Benczur and Karger does not in general keep the edge e in its sparsifier. That said, some of the inspiration for our algorithm comes from the observation that we must treat sparse cuts as they treat minimum cuts.

Other matrix sparsifiers that randomly sample entries have been devised by Achlioptas and McSherry [1] and Frieze, Kannan and Vempala [10]. The algorithm of Achlioptas and McSherry takes as input a matrix A and outputs a sparse matrix $\hat{A}$ that satisfies inequalities analogous to (1) for all $\boldsymbol{x}$ in the range of the dominant eigenvectors of A. Similarly, if one applies the algorithm of Frieze, Kannan and Vempala to the directed edge-vertex adjacency matrix of a graph G, then one obtains a graph $\tilde{G}$ satisfying (1) for all $\boldsymbol{x}$ in the span of the few singular vectors of largest singular value. In contrast, our sparsifiers must satisfy this equation on the whole space. Again, one can observe that neither of these algorithms is likely to keep the edge e in the example above. That said, we do prove that a rounding similar to that used by Achlioptas and McSherry works for our purposes if the graph A has reasonably large isoperimetric number.

1.3 Partitioning

In Section 3, we present an algorithm that quickly finds crude cuts in graphs of approximately optimal balance. Given a graph G containing a set of vertices S such that $\Phi(S) < \phi$ and $\mathrm{Vol}(S) \leq \mathrm{Vol}(V)/2$, our algorithm `Partition` finds a set of vertices T such that $\mathrm{Vol}(T) \geq \mathrm{Vol}(S)/2$ and $\Phi(T) \leq O(\phi^{1/3}\log^{O(1)} n)$ in time $O(m((\log n)/\phi)^{O(1)})$. For our purposes, we may apply this algorithm with $\phi = 1/\log^{O(1)} n$. This algorithm works by approximating the distributions of many random walks on the graph, and its analysis is based on techniques used by Lovasz and Simonovits [20] to analyze their volume estimation algorithm.

We are aware of three theoretically analyzable general-purpose algorithms for graph partitioning: the spectral method [8, 9], the linear-programming relaxation of Leighton and Rao [18], and the random-walk algorithm implicit in the work of Lovasz and Simonovits [20]. Of these, the linear-programming based algorithm provides the the best approximation of the sparsest cut, but is by far the slowest. The spectral method partitions by computing an eigenvector of

the Laplacian matrix of a graph, and produces a quadratic approximation of the sparsest cut. This algorithm can be sped up by applying the Lanczos algorithm to compute an approximate eigenvector. Given a graph with a cut of sparsity less than ϕ, this sped-up algorithm can compute a cut of sparsity at most $\sqrt{\phi}$ in time $O(n\sqrt{1/\phi})$. However, there seems to be no way to control the balance of the cut it outputs. Finally, Lovasz and Simonovits essentially show that by examining random walks in a graph, one can obtain an algorithm that produces similar cuts in time $O(n/\phi)$. To quickly obtain cuts of reasonable balance, our graph partitioning algorithm exploits truncated random walks, and our analysis builds upon the techniques of [20].

We remark that the most successful graph partitioning algorithms in practice are the multi-level methods incorporated into Metis [17] and Chaco [15]. However, there are still no theoretical analyses of the qualities of the cuts produced by these algorithms on general graphs.

The most natural way to partition a graph into pieces such that each has large isoperimetric number would be to apply `Partition`, and then apply it again to each component, etc. However, we have been unable to prove that this algorithm will terminate after a bounded number of iterations. Instead, we analyze an algorithm `MultiwayPartition` that performs these partitions a bounded number of times. Instead of proving that each resulting component has large isoperimetric number, we prove that each remaining component lies within a subgraph of large isoperimetric number. The relation between these components and subgraphs is somewhat technical, and appears in Theorem 4.1. The key to the analysis of this algorithm is the introduction of a variant of the isoperimetric number, which we denote Φ.

2. NOTATION

We recall that a matrix is diagonally dominant if $A_{i,i} \geq \sum_{j=1}^{n} |A_{i,j}|$ for all i. As explained in [23], the reductions introduced in [13, 4] allow us to solve SDD systems by merely preconditioning Laplacian systems. We recall that a symmetric matrix is a Laplacian if all its off-diagonals are non-positive and the sum of the entries in each row is 0. For a non-negative matrix A, we let $\mathcal{L}(A)$ denote the corresponding Laplacian.

There are three natural ways to formulate the problem of finding an approximate solution to a system $A\boldsymbol{x} = \boldsymbol{b}$. A vector $\tilde{\boldsymbol{x}}$ has *relative residual error* ϵ if $\|A\tilde{\boldsymbol{x}} - \boldsymbol{b}\| \leq \epsilon \|\boldsymbol{b}\|$. We say that a solution $\tilde{\boldsymbol{x}}$ is an ϵ-approximate solution if it is at relative distance at most ϵ from the actual solution—that is, if $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\| \leq \epsilon \|\boldsymbol{x}\|$. Finally, one sometimes requires $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_A < \epsilon$, where $\|\boldsymbol{y}\|_A \stackrel{\text{def}}{=} \boldsymbol{y}^T A \boldsymbol{y}$. One can relate these three notions of approximation by observing that each of these measures of error are within a factor of $\kappa_f(A)$ of each other.

The l_2 norm of a matrix, $\|A\|$, is the maximum of $\|Ax\| / \|x\|$, and equals the largest eigenvalue of A if A is symmetric. For non-symmetric matrices, $\lambda_{max}(A)$ and $\|A\|$ are typically different. We let $|A|$ denote the number of non-zero entries in A.

For Laplacian matrices L and $\tilde{L}$ such that the nullspace of $\tilde{L}$ is contained in the nullspace of L, we recall the definition of the *support* of $\tilde{L}$ in L:

$$\sigma_f(L, \tilde{L}) = \max_{\boldsymbol{x}: \tilde{L}\boldsymbol{x} \neq 0} \frac{\boldsymbol{x}^T L \boldsymbol{x}}{\boldsymbol{x}^T \tilde{L} \boldsymbol{x}},$$

and note that for matrices L and $\tilde{L}$ with the same nullspace, we may express

$$\kappa_f(L, \tilde{L}) = \sigma_f(L, \tilde{L}) \sigma_f(\tilde{L}, L),$$

We also note that

$$\sigma_f(L, \tilde{L}) \leq \lambda \text{ if and only if } \lambda L \succcurlyeq \tilde{L},$$

and that there exists a scaling factor μ such that $\mu\tilde{L}$ is an $\kappa_f(L, \tilde{L})$-approximation of L. For more information on these quantities, we refer the reader to [6].

Let $G = (V, E)$ be an undirected unweighted graph with n vertices and m edges. For each $S \subseteq V$, we let $G(S)$ be the induced graph on the vertices in S. We also define $\mathrm{Vol}_V(S) = \sum_{v \in S} d(v)$ where $d(v)$ is the degree of vertex v in G. We note that $\mathrm{Vol}_V(V) = 2m$.

Each subset $S \subseteq V$ defines a *cut* and hence a *partition* $(S, \bar{S})$ of G, where $\bar{S} = V - S$. Let $\partial_V(S) = E(S, \bar{S})$ be the set of edges with exactly one endpoint in S and one endpoint in $\bar{S}$. The *sparsity* of the set is defined to be

$$\Phi_V(S) \stackrel{\text{def}}{=} \frac{|\partial_V(S)|}{\min(\mathrm{Vol}_V(S), \mathrm{Vol}_V(\bar{S}))},$$

and the *isoperimetric number* of the graph is

$$\Phi_V = \min_{S \subset V} \Phi_V(S).$$

The *balance* of a cut S or a partition $(S, \bar{S})$ where $\mathrm{Vol}_V(S) \leq \mathrm{Vol}_V(\bar{S})$ is

$$\mathbf{bal}(S) = \frac{\mathrm{Vol}_V(S)}{\mathrm{Vol}_V(V)}.$$

We also define these terms in the subgraph of G induced by a subset of the vertices $W \subseteq V$: For $S \subseteq W$,

$$\mathrm{Vol}_W(S) \stackrel{\text{def}}{=} \sum_{v \in S} |w \in W : (v, w) \in E|,$$

$$\partial_W(S) \stackrel{\text{def}}{=} \sum_{v \in S} |w \in W - S : (v, w) \in E|,$$

$$\Phi_W(S) \stackrel{\text{def}}{=} \frac{|\partial_W(S)|}{\min(\mathrm{Vol}_W(S), \mathrm{Vol}_W(\bar{S}))}.$$

3. PARTITIONING

The algorithm `Nibble` works by approximately computing the distribution of a few steps of the random walk starting at the seed vertex v. It is implicit in the analysis of the volume estimation algorithm of Lovasz and Simonovits [20] that one can find a small cut from the distributions of the steps of the random walk starting at any vertex from which the walk does not mix rapidly. We first note that a random vertex in S is probably such a vertex. We then extend the analysis of Lovasz and Simonovits to show one can find a small cut from approximations of these distributions, and that these approximations can be computed quickly. In particular, we will truncate all small probabilities that appear in the distributions to 0. In this way, we minimize the work required to compute our approximations.

We will use the definitions of the following two vectors:

$$\chi_S(x) = \begin{cases} 1 & \text{for } x \in S, \\ 0 & \text{otherwise,} \end{cases}$$

$$\psi_S(x) = \begin{cases} d(x)/\mathrm{Vol}_V(S) & \text{for } x \in S, \\ 0 & \text{otherwise.} \end{cases}$$

We note that ψ_V is the steady-state distribution of the random walk, and that ψ_S is the restriction of that walk to the set S.

Given an unweighted graph A, we will consider the walk that at each time step stays put with probability 1/2, and otherwise moves to a random neighbor of the current vertex. The matrix realizing this walk can be expressed $P = (AD^{-1} + I)/2$, where $d(i)$ is the degree of node i, and D is the diagonal matrix with $(d(1), \ldots, d(n))$ on the diagonal. We will let p_t^v denote the distribution obtained after t steps of the random walk starting at vertex v. In this notation, we have $p_t^v = P^t \chi_v$. We will omit v when it is understood. For convenience, we introduce the notation

$$\rho_t^v(x) = p_t^v(x)/d(x).$$

To describe the rounded random walks, we introduce the truncation operation

$$[p]_\epsilon(v) = \begin{cases} p(v) & \text{if } p(v) \geq 2\epsilon d(i), \\ 0 & \text{otherwise.} \end{cases}$$

We then have the truncated probability vectors

$$\begin{aligned} \tilde{p}_0 &= p_0 \\ \tilde{p}_t &= [P\tilde{p}_{t-1}]_\epsilon. \end{aligned} \tag{2}$$

That is, at each time step, we will evolve the random walk one step from the current density, and then round every $p_t(i)$ that is less than $2d(i)\epsilon$ to 0. We remark that this will result in an odd situation in which the sum of the probabilities that we are carrying around will be less than 1.

$C = \texttt{Nibble}(G, v, \theta_0, b)$
G a graph, v a vertex, $\theta_0 \in (0,1)$, b a positive integer.

(1) Set $\tilde{p}_0(x) = \chi_v$.

(2) Set $t_0 = 49\ln(me^4)/\theta_0^2$, $\gamma = \frac{5\theta_0}{7 \cdot 7 \cdot 8\ln(me^4)}$, and $\epsilon_b = \frac{\theta_0}{7 \cdot 8\ln(me^4) t_0 2^b}$.

(3) For $t = 1$ to t_0

(a) Set $\tilde{p}_t = [P\tilde{p}_{t-1}]_{\epsilon_b}$.

(b) Compute a permutation $\tilde{\pi}_t$ such that $\tilde{\rho}_t(\tilde{\pi}_t(i)) \geq \tilde{\rho}_t(\tilde{\pi}_t(i+1))$ for all i.

(c) If there exists a $\tilde{j}$ such that

i $\Phi(\tilde{\pi}_t(\{1, \ldots, \tilde{j}\})) \leq \theta_0$,

ii $\tilde{\rho}_t(\tilde{\pi}_t(\tilde{j})) \geq \gamma/\mathrm{Vol}_V(\tilde{\pi}_t(\{1, \ldots, \tilde{j}\}))$, and

iii $5\mathrm{Vol}_V(V)/6 \geq \mathrm{Vol}(\tilde{\pi}_t(\{1, \ldots, \tilde{j}\})) \geq (5/7)2^{b-1}$.

then output $C = \tilde{\pi}_t(\{1, \ldots, \tilde{j}\})$ and quit.

(4) Return $\emptyset$.

We will use the following notation.

$$\theta_+ \stackrel{\text{def}}{=} \theta_0^3/14^4 \ln^2(me^4). \tag{3}$$

LEMMA 3.1 (NIBBLE). *Nibble can be implemented so that it runs in time $O\left(2^b \ln^4(m)/\theta_0^5\right)$. If the set C output by Nibble is non-empty, it satisfies*

i. $\Phi_V(C) \leq \theta_0$,

ii. $\mathrm{Vol}_V(C) \leq (5/6)\mathrm{Vol}_V(V)$.

Moreover, for each $\theta_0 \leq 1/2$ and for each set S satisfying

$$\mathrm{Vol}_V(S) \leq (2/3)\mathrm{Vol}_V(V) \quad \text{and} \quad \Phi_V(S) \leq 2\theta_+,$$

there is a subset $S^g \subseteq S$ such that $\mathrm{Vol}_V(S^g) \geq \mathrm{Vol}_V(S)/2$ that can be decomposed into sets S_b^g for $b = 1, \ldots, \lg m$ such that if Nibble is started from a vertex $v \in S_b^g$ and run with parameters θ_0 and b, then it will output a set of vertices C such that

iii. $(4/7)2^{b-1} \leq \mathrm{Vol}_V(C \cap S)$.

The following are some definitions and lemmas used in the analysis of Nibble.

DEFINITION 3.2 (S^g). *For each set $S \subseteq V$, we define S^g to be the set of nodes x in S such that*

$$\left\langle \chi_{\bar{S}} | P^{t_0} \chi_x \right\rangle \leq 2\left\langle \chi_{\bar{S}} | P^{t_0} \psi_S \right\rangle.$$

PROPOSITION 3.3 (MASS OF S^g).

$$\mathrm{Vol}_V(S^g) \geq \mathrm{Vol}_V(S)/2.$$

LEMMA 3.4 (CUT FROM RANDOM WALK). *For any $\phi > 0$, let $t_0 = \ln(me^4)/\phi^2$, $\alpha = 1/4t_0$. Then, for every set S such that $\mathrm{Vol}(S) \leq \mathrm{Vol}(V)/2$ and*

$$\Phi(S)t_0 < 1/32,$$

for all $v \in S^g$, if we start the random walk at χ_v, then there exists a $t < t_0$ and a $j \leq m$ such that

(a) $\Phi(\pi_t(\{1, \ldots, j\})) \leq \phi$, *and*

(b) for the j_0 and j_1 defined by $k_j^t - 2\phi\bar{k}_j^t \in (k_{j_0-1}^t, k_{j_0}^t]$ and $k_j^t + 2\phi\bar{k}_j^t \in (k_{j_1-1}^t, \leq k_{j_1}^t]$,

$$\rho_t(\pi_t(j_0)) - \rho_t(\pi_t(j_1)) > \frac{\phi}{4\ln(me^4)\mathrm{Vol}(\pi_t(\{1, \ldots, j\}))}, \tag{4}$$

where for all integers $j \in [0, n]$, we define $k_j^t = \sum_{i=1}^j d(\pi_t(i))$, and we define $\bar{k}_j^t = \min(k_j^t, 2m - k_j^t)$.

PROPOSITION 3.5 (MONOTONICITY OF MULT BY P). *For all non-negative vectors p,*

$$\left\|D^{-1}(Pp)\right\|_\infty \leq \left\|D^{-1}p\right\|_\infty.$$

PROPOSITION 3.6 (ESCAPING MASS).

$$\left\langle \chi_S | P^{t_0} \psi_S \right\rangle \geq 1 - t_0\Phi_V(S).$$

LEMMA 3.7 (OVERLAP WITH S). *For a set S for which*

$$\Phi(S) \leq \frac{\theta_0^3}{7^4 \cdot 8\ln^2(me^4)}, \tag{5}$$

if the truncated random walk (2) *is started from any vertex* $v \in S^g$*, then for every* $t < t_0 = 7^2 \ln(me^4)/\theta_0^2$ *and* $\tilde{j}$ *satisfying*

$$\tilde{\rho}_t(\tilde{j}) \geq \frac{5\theta_0}{7^2 \cdot 8 \ln(me^4) \mathrm{Vol}\left(\tilde{\pi}_t(\{1,\ldots,\tilde{j}\})\right)},$$

we have

$$\mathrm{Vol}\left(\tilde{\pi}_t(\{1,\ldots,\tilde{j}\}) \cap S\right) \geq (4/5)\mathrm{Vol}\left(\tilde{\pi}_t(\{1,\ldots,\tilde{j}\})\right).$$

LEMMA 3.8 (LOW-IMPACT TRUNCATION). *Let the values* $\rho_t(v)$ *be derived from the ordinary random walk and the values* $\tilde{\rho}_t(v)$ *be derived from the truncated random walk with truncation factor* ϵ_b*. Then, for all* t *and* v*,*

$$\rho_t(v) \geq \tilde{\rho}_t(v) \geq \rho_t(v) - 2t\epsilon_b.$$

DEFINITION 3.9 (S_b^g). *For every set* $S \subseteq V$*, we define* S_b^g *to be the set of vertices in* S^g *such that when the random walk is started at that vertex, the first* t *for which there is a* j *satisfying conditions* (*a*) *and* (*b*) *of Lemma 3.4 has the property that for the least such* j

$$2^{b-1} \leq \mathrm{Vol}\left(\pi_t(\{1,\ldots,j\})\right) < 2^b.$$

LEMMA 3.10 (ANALYSIS OF TRUNCATED WALK). *For each* $\theta_0 \leq 1$*, if* S *is a set satisfying* $\mathrm{Vol}(S) \leq (2/3)\mathrm{Vol}(V)$ *and*

$$\Phi(S) \leq \frac{\theta_0^3}{7^4 \cdot 8 \ln^2(me^4)}$$

and `Nibble` *is started at a vertex* $v \in S_b^g$ *with parameter* b*, then there exists a* $t < t_0$ *and a* $\tilde{j}$ *such that conditions i, ii and iii of line* (3.*c*) *of* `Nibble` *are satisfied.*

3.1 Random Nibble and Partition

To define `Partition`, we first define an intermediate algorithm `Random Nibbles` which calls `Nibble` on carefully chosen random inputs.

$C = \texttt{RandomNibble}(G, \theta_0)$

(1) Choose a vertex v according to ψ_V.

(2) Choose a b in $1, \ldots, \lceil \log m \rceil$ according to

$$\Pr[b = i] = 2^{-i}/(1 - 2^{-\lceil \log m \rceil}).$$

(3) $C = \texttt{Nibble}(G, v, \theta_0, b)$.

LEMMA 3.11 (RANDOM NIBBLE). *The expected running time of* `Random Nibble` *is* $O\left(\ln^4(m)/\theta_0^5\right)$*. If the set* C *output by* `Random Nibble` *is non-empty, it satisfies*

i. $\Phi_V(C) \leq \theta_0$,

ii. $\mathrm{Vol}_V(C) \leq (5/6)\mathrm{Vol}_V(V)$.

Moreover, for each $\theta_0 \leq 1/2$ *and for each set* S *satisfying*

$$\mathrm{Vol}_V(S) \leq (2/3)\mathrm{Vol}_V(V) \quad \textit{and} \quad \Phi_V(S) \leq 2\theta_+,$$

iii. $\mathbf{E}[\mathrm{Vol}(C \cap S)] \geq \mathrm{Vol}(S)/14m$,

where θ_+ *is as defined in* (3).

$D = \texttt{Partition}(G, \theta_0, p)$
where G is a graph, $\theta_0, p \in (0,1)$.

(0) Set $W_1 = V$.

(1) For $j = 1$ to $56m\lceil \lg(1/p) \rceil$.

(a) Set $D_j = \texttt{RandomNibble}(G(W_j), \theta_0)$

(b) Set $W_{j+1} = W_j - D_j$.

(c) If $\mathrm{Vol}_{W_{j+1}}(W_{j+1}) \leq (5/6)\mathrm{Vol}_V(V)$, then go to step (2).

(2) Set $D = V - W_{j+1}$.

$$\theta_0 \stackrel{\text{def}}{=} (5/36)\theta. \tag{6}$$

THEOREM 3.12 (PARTITION). *The expected running time of* `Partition` *is at most* $O\left(m \lg(1/p) \ln^4(m)/\theta_0^5\right)$*. Let* D *be the output of* $\texttt{Partition}(G, \theta_0, p)$*, where* G *is a graph and* $\theta_0, p \in (0,1)$*. Then*

i. $\mathrm{Vol}_D(D) \leq (31/36)\mathrm{Vol}_V(V)$,

ii. $\Phi_V(D) \leq \theta$*, as defined in* (6).

Moreover, for each set S *satisfying*

$$\mathrm{Vol}_V(S) \leq (2/3)\mathrm{Vol}_V(V) \quad \textit{and} \quad \Phi_V(S) \leq 2\theta_+,$$

with probability at least $1 - p$*, either*

iii.a. $\mathrm{Vol}_{V-D}(V - D) \leq (5/6)\mathrm{Vol}_V(V)$*, or*

iii.b. $\mathrm{Vol}_{V-D}(S \cap (V - D)) \leq (1/2)\mathrm{Vol}_V(S)$,

where θ_+ *is as defined in* (3).

4. MULTIWAY PARTITION

Our multiway partitioning algorithm is:

$\mathcal{C} = \texttt{MultiwayPartition}(G, \theta, p)$

(0) Set $\mathcal{C}_1 = V$ and $S = \emptyset$.

(1) For $t = 1$ to $\lceil \log_{17/16} m \rceil \cdot \lceil \lg m \rceil \cdot \lceil \lg(2/\epsilon) \rceil$

(a) For each component $C \in \mathcal{C}_t$,
$D = \texttt{Partition}(G(C), \theta_0, p/m)$.
Add D and $C - D$ to $\mathcal{C}_{t+1}$.

(2) Return $\mathcal{C} = C_{t+1}$.

Let m be an upper bound on the number of edges of the input graph. We let

$$\epsilon \stackrel{\text{def}}{=} \min\left(\frac{1}{16}, \frac{1}{4\lceil \lg m \rceil}\right), \text{ and} \tag{7}$$

$$\theta_* \stackrel{\text{def}}{=} \epsilon\theta_+/32, \tag{8}$$

where θ_+ is defined as in Equation (3).

THEOREM 4.1 (MULTIWAYPARTITION). *Let* $G = (V, E)$ *be an undirected graph of* n *vertices and at most* m *edges. For any* $0 < \theta < 1$*, let* $\mathcal{C}$ *be the set of components returned by* `MultiwayPartition`*. Then, with probability at least* $1 - p$*,*

1 $\boldsymbol{cut\text{-}size}(\mathcal{C}) \leq \left(\theta \log_{17/16} m \cdot \lg m \cdot \lg(2/\epsilon)\right)(m/2)$*, and*

2. *there exists a set* $\mathcal{W}$ *of subsets of* V*, an assignment* $\textbf{\textit{level}} : \mathcal{W} \to \left\{1, \ldots, \lceil \log_{17/16} m \rceil\right\}$*, and a mapping* $\pi : \mathcal{C} \to \mathcal{W}$ *such that*
   a. *For all* $W \in \mathcal{W}$, $\Phi_W \geq \theta_*$,
   b. *For all* $C \in \mathcal{C}$, $C \subseteq \pi(C)$
   c. *For all* $l \in \left\{1, \ldots, \lceil \log_{17/16} m \rceil\right\}$, $\{W \in \mathcal{W} : \textbf{\textit{level}}(W) = l\}$ *are pair-wise disjoint.*
   d. *For each pair* $W_i \in \mathcal{W}$ *and* $W_j \in \mathcal{W}$ *such that* $\textbf{\textit{level}}(W_i) > \textbf{\textit{level}}(W_j)$,

$$W_i \cap \left(\cup_{C \in \mathcal{C}:\pi(C)=W_j} C\right) = \emptyset.$$

In addition, the expected running time of `MultiwayPartition` *is* $m\left(\lg(1/p)\lg^{O(1)}(m)\right)/\theta^5$.

Our analysis of `MultiwayPartition` employs the following variant of the isoperimetric number: for a graph $G = (V, E)$ and a subset S of V, we define

$$\Phi_V(S) = \frac{\partial_V(S)}{\min\left(\mathrm{Vol}(S), \mathrm{Vol}(V - S)\right)^{1+4\epsilon}}.$$

We also define Φ of a subset S by

$$\Phi_S = \min_{T \subseteq S} \Phi_S(T) = \min_{T \subseteq S} \frac{\partial_S(T)}{\min\left(\mathrm{Vol}(T), \mathrm{Vol}(S - T)\right)^{1+4\epsilon}}.$$

Note that the induced graph of S is connected if and only if $\Phi_S > 0$.

The purpose of this definition of Φ is to satisfy the following lemma.

LEMMA 4.2 (UNION OF SETS WITH SMALL INTERSECTION). *Let* $0 < \epsilon < 1/4$. *Let* S *and* T *be sets of vertices such that* $\mathrm{Vol}(S \cap T) \leq \epsilon \min(\mathrm{Vol}(S), \mathrm{Vol}(T))$. *Then*

$$\mathrm{Vol}(S \cup T)^{1+4\epsilon} > \mathrm{Vol}(S)^{1+4\epsilon} + \mathrm{Vol}(T)^{1+4\epsilon}.$$

If in addition $\mathrm{Vol}(S \cup T) \leq (1/2)\mathrm{Vol}(V)$, *then*

$$\Phi_V(S \cup T) \leq \max\left(\Phi_V(S), \Phi_V(T)\right)$$

PROPOSITION 4.3 (Φ AND Φ). *For every set* S, $\Phi_V(S)/2 \leq \Phi_V(S) \leq \Phi_V(S)$.

LEMMA 4.4 (DIVIDED OR COVERED: EACH EPOCH). *For each* $t \geq 1$ *and* $C \in \mathcal{C}_t$, *let* $t' = t + \lceil \lg m \rceil \cdot \lceil \lg(2/\epsilon) \rceil$. *If every call made by* `MultiwayPartition` *to* `Partition` *succeeds, then either*

- *for all components* $C' \in \mathcal{C}_{t'}(C)$, $\mathrm{Vol}_{C'}(C') \leq (16/17)\mathrm{Vol}_C(C)$, *or*
- *Let* $C_{t'}$ *be the unique component in* $\mathcal{C}_{t'}(C)$ *such that* $\mathrm{Vol}_{C_{t'}}(C_{t'}) > (16/17)\mathrm{Vol}_C(C)$. *There exists a set* $W \in C_t$ *with* $\Phi_W \geq \theta_*$ *and* $C_{t'} \subseteq W$.

PROOF SKETCH. Assume

$$\mathrm{Vol}_{C_{t'}}(C_{t'}) > (16/17)\mathrm{Vol}_C(C). \tag{9}$$

Let $C_t = C$ for notational simplicity. For $t \leq j \leq t'$, let C_j be the unique component in $\mathcal{C}_j(C)$ such that $\mathrm{Vol}_{C_j}(C_j) > (16/17)\mathrm{Vol}_C(C)$. Then $C_{t'} \subseteq C_{t'-1} \subseteq \cdots \subseteq C_{t+1} \subseteq C_t$.

Let $V_0 = C_t$. For $i \in [0 : \lceil \lg m \rceil]$ we iteratively define U_{i+1}, V_{i+1}, and W_{i+1} and S_{i+1} by:

- $S_i \subset V_i$ is the largest subset such that $\mathrm{Vol}_{V_i}(S_i) \leq \mathrm{Vol}_{V_i}(V_i)/2$ and $\Phi_{V_i}(S_i) \leq 2\theta_*$,
- $W_{i+1} = C_{t+i\lg(2/\epsilon)}$ and $U_{i+1} = V_i - W_{i+1}$, and
- $V_{i+1} = V_i - (S_i \cap U_{i+1})$.

As $W_1 \subseteq V_0$, inductively it follows from $V_{i+1} = V_i - (S_i \cap U_{i+1}) = V_i - (S_i \cap (V_i - W_{i+1}))$ and $W_{i+1} \subseteq W_i \subseteq V_i$, that $W_{i+1} \subseteq V_{i+1}$. In particular, $C_{t'} = W_{\lceil \lg m \rceil} \subseteq V_{\lceil \lg m \rceil}$. Using Lemma 4.5, we show that $S_{\lceil \lg m \rceil} = \emptyset$. So, $\Phi_{V_{\lceil \lg m \rceil}} \geq \theta_*$, and $W = V_{\lceil \lg m \rceil}$ is the set claimed to exist by the lemma. □

LEMMA 4.5 (REDUCTION). *For* V_i, U_i, W_i *and* S_i *as defined in the proof of Lemma 4.4, if*

a. $\mathrm{Vol}_{V_i}(S_i \cap W_{i+1}) < (\epsilon/2)\mathrm{Vol}_{V_i}(S_i)$,
b. $\mathrm{Vol}_{V_i}(S_i \cup S_{i+1}) < \mathrm{Vol}_{V_i}(V_i)/2$, *and*
c. $\mathrm{Vol}_{V_i}(S_i) \leq \mathrm{Vol}_{V_i}(V_i)/16$,

then,

i. $\mathrm{Vol}_{V_{i+1}}(S_{i+1}) \leq \mathrm{Vol}_{V_{i+1}}(V_{i+1})/16$, *and*
ii. $\mathrm{Vol}_{V_{i+1}}(S_{i+1}) \leq \mathrm{Vol}_{V_i}(S_i)/2$.

5. RANDOM SAMPLING

If we let $\tilde{L}$ be the result of randomly sampling the edges of L, we can not in general assume that $\kappa_f(L, \tilde{L})$ will be bounded. However, we can bound $\kappa_f(L, \tilde{L})$ if the smallest eigenvalue of $D^{-1}L$ is bounded from below.

LEMMA 5.1 (SMALL NORM GOOD PRECONDITIONER). *Let* L *and* $\tilde{L}$ *be Laplacian matrices and let* D *be a diagonal matrix with positive diagonals. If* L *has co-rank 1 and* $\lambda_{max}(D^{-1}(L - \tilde{L})) < (1/2)\lambda_{min}(D^{-1}L)$, *then*

$$\begin{aligned} \sigma_f(L, \tilde{L}) &\leq 1 + 2\frac{\lambda_{max}(D^{-1}(L - \tilde{L}))}{\lambda_{min}(D^{-1}L)}, \quad \textit{and} \\ \sigma_f(\tilde{L}, L) &\leq 1 + \frac{\lambda_{max}(D^{-1}(L - \tilde{L}))}{\lambda_{min}(D^{-1}L)}. \end{aligned}$$

Our graph sampler is much like that used in [3] and [1].

$\tilde{A} =$ `Sample`(A, c)
A is an unweighted adjacency matrix, $c \geq 1$.

(1) Set $d(i) = \sum_j a_{i,j}$.

(2) For all i, j for which $a_{i,j} \neq 0$, set
$$p_{i,j} = \begin{cases} \frac{c a_{i,j}}{\min(d(i), d(j))} & \text{if } c < \min(d(i), d(j))/a_{i,j}, \\ 1 & \text{otherwise.} \end{cases}$$

(3) For all i, j for which $a_{i,j} \neq 0$, set $\tilde{a}_{i,j} = \tilde{a}_{j,i} =$
$$\begin{cases} \frac{a_{i,j}}{p_{i,j}} & \text{with probability } p_{i,j}, \\ 0 & \text{with probability } 1 - p_{i,j}. \end{cases}$$

(4) Return the matrix $\tilde{A}$ of the $\tilde{a}_{i,j}$s.

By adapting techniques used by Füredi and Komlós [11] to study random matrices, we prove:

THEOREM 5.2 (SAMPLING). *Let A be a non-negative symmetric matrix and let $c \geq 1$. Let $d(i) = \sum_j a_{i,j}$, and let $D = \mathrm{diag}(d(1), \ldots, d(n))$. Let $\tilde{A}$ be the output of* `Sample` *(A,c). Then, for all $\alpha \geq 1$, and even integers k,*

$$\Pr\left[\lambda_{max}\left(D^{-1}(\tilde{A}-A)\right) \geq \frac{2\alpha k n^{1/k}}{\sqrt{c}}\right] < \alpha^{-k}.$$

LEMMA 5.3 (CLOSE WEIGHTED DEGREES). *Let A be the adjacency matrix of an unweighted graph, and let $\tilde{A}$ be the output of* `Sample`*(A,c). Let $d(1), \ldots, d(n)$ be the degrees of the vertices of A and let $\tilde{d}(1), \ldots, \tilde{d}(n)$ be the corresponding terms for $\tilde{A}$. Then, for $\delta < 1$,*

(a) for all i, $\Pr\left[\left|1 - d(i)^{-1}\tilde{d}(i)\right| > \delta\right] < 2e^{-c\delta^2/3}$, and

(b) the probability that $\tilde{A}$ has more than $2nc$ edges is at most $(4/e)^{-cn/2}$.

THEOREM 5.4 (PRECONDITIONING BY SAMPLING). *Let A be the adjacency matrix of an unweighted graph, L be its Laplacian, D the diagonal matrix of its degrees, and let $\lambda_{min}(D^{-1}A) \geq \lambda$. Let B be the adjacency matrix of a subgraph of A. Let $0 < p < 1$ and*

$$k \stackrel{\mathrm{def}}{=} \max\left(\lceil \lg(2/p) \rceil, \lceil \lg n \rceil\right).$$

For any $\beta < 1$, let $\widetilde{B} =$ `Sample`*(B,c), where*

$$c = (30k/\beta\lambda)^2.$$

If we then let $\tilde{A} = \widetilde{B} + (A - B)$, and let $\tilde{L}$ be its Laplacian, then

$$\Pr\left[\sigma_f(L,\tilde{L}) > 1 + \beta/3 \quad \textit{and} \quad \sigma_f(\tilde{L},L) > 1 + 2\beta/3\right] < p.$$

6. UNWEIGHTED SPARSIFIERS

Our construction of sparsifiers depends upon sparsifiers for unweighted graphs. For a multiway partition $\mathcal{C}$, and a set of edges F, we let $\mathrm{bridge}(\mathcal{C}, F)$ denote the set of edges of F going between components of the partition.

$\widetilde{E} =$ `UnweightedSparsifier`(E,β),
E is a set of unweighted edges and $\beta < 1$.

(0) Set $\theta = \left(\log_{17/16} m \cdot \lg m \cdot \lg(8 \lg m)\right)^{-1}$ and $\lambda = \theta_*^2/2$, where θ_* is given by (6), (3) and (8).

(1) $\mathcal{C} =$ `MultiwayPartition`$(E, \theta, 1/n^2)$

(2) For each $C \in \mathcal{C}$ set $\widetilde{C} =$ `Sample`(C,c), where $c = (30(\lg n + 2)/\beta\lambda)^2$. Set $\tilde{A} = \tilde{A} \cup \widetilde{C}$.

(3) $S = \mathrm{bridge}(\mathcal{C}, E)$

(4) $\widetilde{S} =$ `UnweightedSparsifier`(S,β). Set $\tilde{A} = \tilde{A} \cup \widetilde{S}$.

LEMMA 6.1 (UNWEIGHTED SPARSIFIER). *Let E be a set of unweighted edges E, let $\beta < 1$, and let $\widetilde{E}$ be the output of* `UnweightedSparsifier`(E,β). *Then, $|\widetilde{E}| < n \log^{O(1)} n/\beta^2$ with exponentially high probability. Moreover,*

$$\Pr\left[\begin{array}{l} E \preccurlyeq (1+\beta/3)^{O(\lg^2 m)}\, \widetilde{E} \quad \textit{and} \\ \widetilde{E} \preccurlyeq (1+2\beta/3)^{O(\lg^2 m)}\, E \end{array}\right] \geq 1 - \frac{\lg m}{m^2}.$$

The expected running time of `Unweighted Sparsifier` *is $O(m \log^{O(1)}(m/\beta))$.*

7. SPARSIFYING WEIGHTED GRAPHS

This section will require the following definitions:

For a set E of weighted edges, we let $\mathrm{edges}(E)$ denote the set of edges in E. If needed, elements of $\mathrm{edges}(E)$ are assumed to have weight 1. The *degree* of vertex i in E is given by $\deg_E(i) = |\{(i,j) \in \mathrm{edges}(E)\}|$. The *weighted degree* of vertex i in E is given by $\mathrm{wdeg}_E(i) = \sum_{(\{i,j\},w) \in E} w$.

The *capacity* of a path containing edges of weights $\omega_1, \ldots, \omega_k$ is given by

$$1/\left(1/\omega_1 + \cdots + 1/\omega_k\right).$$

If T is a weighted tree and e is an edge whose endpoints are connected by a path in T, then the weighted dilation of e in T, $\mathbf{wd}_T(e)$ is the weight of e divided by the capacity of the path.

If F is another set of weighted edges, we let $\mathrm{bridge}(E,F)$ be the set of edges in F that span connected components of the graph defined by E.

So that we can state `Rewire` in `UltraSparsify`, we need the following variation of a definition from [23]:

DEFINITION 7.1. *For a set of edges F, an F-decomposition is a pair $(\mathcal{W}, \pi)$ where $\mathcal{W}$ is a collection of sets of vertices and π is a map from F into sets or pairs of sets in $\mathcal{W}$ satisfying*

1. *$|W_i \cap W_j| \leq 1$ for all $i \neq j$, and*

2. *for each edge in $e \in F$, if $|\pi(e)| = 1$, then both endpoints of e lie in $\pi(e)$; otherwise, one endpoint of e lies in one set in $\pi(e)$, and the other endpoint lies in the other.*

The pair is an F-decomposition of E if in addition

3. *for each set $W_i \in \mathcal{W}$, the graph induced by E on W_i is connected, and*

4. *each edge of E lies in exactly one set in $\mathcal{W}$,*

For now, it is probably best to first consider the case in which $E = F$ and all the sets in $\mathcal{W}$ are disjoint, in which case π merely maps each edge to the names of subsets in which its endpoints lie. This is how the definition is used in `Sparsify`. We note that, in general, this definition allows there to be sets $W \in \mathcal{W}$ containing just one vertex of V.

For a set of edges F and a pair $((W_1, \ldots, W_s), \pi)$, we define

$$\mathrm{metaGraph}\left(\{W_1, \ldots, W_s\}, \pi, F\right)$$

to be the weighted graph on vertices $\{1, \ldots, s\}$ with edge $\{i,j\}$ having weight

$$\left|\{(\{a,b\}, w) \in F : \pi(\{a,b\}) = \{W_i, W_j\}\}\right|.$$

Finally, we say that $\{W_1, \ldots, W_s\}$ has a γ-center under E if for all $i \in \{1, \ldots, s\}$, there exists $w_i \in W_i$ such that for all $u \in W_i$, there exists a path in the graph induced by E on W_i from u to w_i of capacity at least γ.

$\widetilde{F} = \texttt{Rewire}(F, (\{W_1, \ldots, W_l\}, \pi), \widetilde{H})$, F is set of unit-weight edges, $(\{W_1, \ldots, W_l\}, \pi)$ is an F-decomposition, and $\widetilde{H}$ is a weighted graph on vertex set $\{1, \ldots, l\}$

(1) Construct a map τ from $\widetilde{H}$ to F as follows:

  (a) For each $(i,j) \in \widetilde{H}$, choose an arbitrary edge $(u,v) \in F$ with $u \in W_i$, $v \in W_j$ and $\pi(u,v) = \{W_i, W_j\}$. Set $\tau(i,j) = (u,v)$.

(2) For each edge (u,v) in the range of τ, set $\tilde{f}_{u,v} = \sum_{(\{i,j\},v):\tau(i,j)=(u,v)} v$.

(3) Let $\widetilde{F}$ be the set of all the weighted edges $\tilde{f}_{u,v}$.

We will make use of the following inequality, which may be derived from the Rank-One Support Lemma of [6]

LEMMA 7.2. *Let $u_0, u_1, \ldots, u_l$ be a path in a graph in which the edge from u_i to u_{i+1} has weight ω_i. Let ω be the capacity of the path. Then, for all $\boldsymbol{x} \in \mathbb{R}^n$,*

$$\omega(x_{u_0} - x_{u_l})^2 \leq \sum_{i=0}^{l-1} \omega_i (x_{u_i} - x_{u_{i+1}})^2.$$

LEMMA 7.3 (REWIRE). *Let E be a set of weighted edges and let F be a set of weight-1 edges on the same vertex set. Let $(\{W_1, \ldots, W_s\}, \pi)$ be an F-decomposition of E such that for each $f \in F$, $|\pi(f)| = 2$. Let $\widetilde{H}$ be a weighted graph on $\{1, \ldots, s\}$. Let $\widetilde{F}$ be the output of `Rewire` on these inputs. Let $H = \texttt{metaGraph}((W_1, \ldots, W_s), \pi, F)$. If d is at least the maximum weighted degree of H and $(W_1, \ldots, W_s)$ has a γ center under E, then*

$$F \preccurlyeq E \cdot d\left(1 + \sigma_f(H, \widetilde{H})^2 (1 + 2/(\gamma - 2))\right) + \widetilde{F}\left(\sigma_f(H, \widetilde{H})(1 + 2/(\gamma - 2))^2\right), \quad (10)$$

and

$$\widetilde{F} \preccurlyeq E \cdot d\left(1 + \sigma_f(H, \widetilde{H})^2 (1 + 2/(\gamma - 2))\right) + F\left(\sigma_f(H, \widetilde{H})(1 + 2/(\gamma - 2))^2\right). \quad (11)$$

$\widetilde{F} = \texttt{metaSparsify}((W_1, \ldots, W_s), \pi, F, \epsilon, p)$,
F is set of unit-weight edges, $((W_1, \ldots, W_l), \pi)$ is an F-decomposition, and $\epsilon, p \in (0,1)$,

(1) $H = \texttt{metaGraph}((W_1, \ldots, W_s), \pi, F)$.

(2) For $q = 0, \ldots, \lceil \log_{1+\epsilon} m \rceil$,

  (a) $H_q = \left\{ \{i,j\} : \begin{array}{l} (\{i,j\}, v) \in H, \\ v \in \left[(1+\epsilon)^{q-1}, (1+\epsilon)^q\right) \end{array} \right\}$.

  (b) $\widetilde{H}_q = \texttt{UnweightedSparsify}(H_q, \epsilon, p/\log_{1+\epsilon} m)$.

  (c) $\widetilde{F}_q = \texttt{Rewire}(F, (W_1, \ldots, W_s), \pi, \widetilde{H}_q)$.

(3) $\widetilde{F} = \sum_q (1+\epsilon)^q \widetilde{F}_q$.

LEMMA 7.4 (METASPARSIFY). *`metaSparsify` can be implemented to run in expected time $O(m \log^{O(1)} m)$. If $\epsilon < 1/2$ and $p < 1$, $(W_1, \ldots, W_s)$ has a γ-center under E and d is at least the maximum degree of $\texttt{metaGraph}((W_1, \ldots, W_s), \pi, F)$ and $\widetilde{F}$ is the output of `metaSparsify`, then with probability at least $1 - p$,*

$$|\widetilde{F}| \leq O(s \log^{O(1)}(m/\epsilon p)/\epsilon^2),$$

$$\widetilde{F} \preccurlyeq E \cdot d\left(1 + (1+\epsilon)^2(1 + 2/(\gamma - 2))\right) + F \cdot \left((1+\epsilon)(1 + 2/(\gamma - 2))^2\right), \quad (12)$$

and

$$F \preccurlyeq E \cdot d\left(1 + (1+\epsilon)^2(1 + 2/(\gamma - 2))\right) + \widetilde{F} \cdot \left((1+\epsilon)(1 + 2/(\gamma - 2))^2\right). \quad (13)$$

$\widetilde{E} = \texttt{Sparsify}(E, \epsilon)$
E a set of weighted edges with max weight 1, $\epsilon > 0$.

(0) Set $\gamma = 2 + 4/\epsilon$.

(1) Let $C^t = \left\{ (\{i,j\}, 1) : \begin{array}{l} (\{i,j\}, v) \in E \text{ and} \\ v \in ((1+\epsilon)^{-t-1}, (1+\epsilon)^{-t}] \end{array} \right\}$

(2) For $t = 0, \ldots,$

  (a) Let $\{W_1, \ldots, W_l\}$ be the partition of V obtained by contracting all edges in classes with index less that $t - \log_{1+\epsilon}(\gamma n m \lg(n)/\epsilon^3)$.

  (b) $\widetilde{C}^t = \texttt{metaSparsify}((W_1, \ldots, W_s), \pi, C^t, \epsilon, p/m)$.

  (c) Set $\widetilde{E} = \sum_t (1+\epsilon)^{-t} \widetilde{C}^t$

THEOREM 7.5 (SPARSIFY). *Let $\epsilon < 1/2$. `Sparsify` can be implemented to have expected running time $O(m \log^{O(1)} m)$. With probability at least $1 - 1/n$ the graph $\tilde{A}$ output by `Sparsify` has at most $O(n \log^{O(1)}(n/\epsilon)/\epsilon^2)$ edges and*

$$\sigma_f(A, \tilde{A}) \leq 1 + \epsilon \text{ and } \sigma_f(\tilde{A}, A) \leq 1 + \epsilon. \quad (14)$$

Our ultra-sparsifiers will build upon the low-stretch spanning trees of Alon, Karp, Peleg and West [2], which we will refer to as AKPW trees. As observed by Boman and Hendrickson [6], if one runs the AKPW algorithm with the reciprocals of the weights in the graph, then one obtains the following guarantee:

THEOREM 7.6 (AKPW). *There exists an $O(m \log m)$-time algorithm, `AKPW`, that on input a weighted connected graph G, outputs a spanning tree $T \subseteq G$ such that*

$$\sum_{e \in E} \boldsymbol{wd}_T(e) \leq m 2^{O(\sqrt{\log n \log\log n})}.$$

We will use the algorithm `decompose` from [23] to compute decompositions of these trees.

THEOREM 7.7 (DECOMPOSE, [23]). *There exists a linear-time algorithm with template,*

$$((W_1, \ldots, W_s), \pi) = \texttt{decompose}(T, E, \phi),$$

that on input a forest T and a set of unit-weight edges E outputs a T-decomposition of E such that

1. *for all W_i such that $|W_i| > 1$, $|\{e \in E : W_i \in \pi(e)\}| \leq \phi$, and*
2. *$s \leq 4|E|/\phi$.*

$(T, A) = \texttt{UltraSparsify}(E, k)$
E a set of weighted edges with max weight 1, $k \geq 1$.

(0) $\widehat{E} = \texttt{Sparsify}(E, 1/2)$, $\hat{m} = |\widehat{E}|$.

(1) $T = \texttt{AKPW}(\widehat{A})$.

(2) For every edge $e \in \widehat{E}$, compute $\mathbf{wd}_T(e)$.

(3) $\tilde{A} = \{e : \mathbf{wd}_T(e) > n\}$
$E_0 = \{e : \mathbf{wd}_T(e) < 2\}$
$E_z = \left\{e : \mathbf{wd}_T(e) \in [2^z, 2^{z+1})\right\}$

(3) For $z = 0, \ldots, \log n$, and $t = 1, 2, \ldots,$

(a) $R^t = \left\{(\{i,j\}, v) \in T : v > 2^{-t}\right\}$
$C_t = \text{bridge}\left(R^t, E_z\right)$

(b) For $q = -1 - \log_2 z, \ldots, 0, 1, \ldots, 3\log_2 n,$

i. $C_q^t = \left\{(\{i,j\}, v) \in C^t : v \in (2^{-t-q}, 2^{-t-q+1}]\right\}$

ii. $(\{W_1, \ldots, W_s\}, \pi) = \texttt{TreeDecomp}(C_q^t, R^t, 4\hat{m}/k2^z)$

iii. $\widehat{C}_q^t = \texttt{metaSparsify}((W_1, \ldots, W_s), \pi, \text{edges}\left(C_q^t\right), 1, p)$

iv. $\widetilde{C}_q^t = \left\{(\{i,j\}, v) \in C_q^t : \{i,j\} \in \text{edges}\left(\widehat{C}_q^t\right)\right\}$

v. $\tilde{A} = \tilde{A} \cup \widetilde{C}_q^t$

THEOREM 7.8 (ULTRASPARSIFY). *`UltraSparsify` can be implemented to have expected running time $O(m \log^{O(1)} m)$. With probability at least $1 - 2^{O(\sqrt{\log n \log\log n})}/n$, the graph $T \cup \tilde{A}$ output by `UltraSparsify` is $k2^{O(\sqrt{\log n \log\log n})}$-ultra-sparse and*

$$\kappa_f(A, \tilde{A}) \leq (n/k) \log^{O(1)} n. \tag{15}$$

8. SOLVING LINEAR SYSTEMS

In this section, we will show how the output of `UltraSparsify` can be used to solve linear systems in A with the preconditioned Chebyshev method and the preconditioned conjugate gradient.

We begin by recalling the basic outline of the use of sparsifiers established by Vaidya [25]. Given a matrix A and an ultra-sparsifier $B = T \cup \tilde{A}$ of A, after appropriately reordering the vertices of A and B, we can perform partial Cholesky factorization of B to obtain $B = L[I, 0; 0, A_1]L^T$. Here, L is a lower-triangular matrix with at most $O(n)$ non-zero entries and A_1 is a square matrix of size at most $4|\tilde{A}|$ with at most $10|\tilde{A}|$ non-zero entries (see [23, Proposition 1.1]). Moreover, if A is SDD then A_1 is as well.

We can then solve linear systems in B by solving a corresponding linear system in A_1 and performing $O(n)$ additional work: given b, one can solve $By = b$ by solving for s in $[I, 0; 0, A_1]s = L^{-1}b$, and then computing $y = L^{-T}s$ by back-substitution.

If we use the output of `UltraSparsify` with $k = \sqrt{m}$ as a preconditioner and solve systems in A_1 using the conjugate gradient method as an exact solver, we obtain the following "one-shot" result

THEOREM 8.1 (ONE-SHOT ALGORITHM). *One can produce an approximate solution $\tilde{\boldsymbol{x}}$ to the system $A\boldsymbol{x} = \boldsymbol{b}$ with $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A \leq \epsilon$ in time $m \log^{O(1)} m + m^{3/4} n^{1/2+o(1)} \log(1/\epsilon)$, with probability $1 - o(1)$.*

PROOF. The time taken by `UltraSparsify` is $m \log^{O(1)} m$, and the time taken by the Cholesky factorization is $O(n)$. Having produced B and A_1, the algorithm solves $A\boldsymbol{x} = \boldsymbol{b}$ to accuracy ϵ in A-norm by applying at most $\sqrt{\kappa_f(A, B)} \log(1/\epsilon)$ iterations of the preconditioned conjugate gradient. Using the conjugate gradient as an exact algorithm, we can solve the system in A_1 in time $O(|A_1|^2)$. Thus, each iteration of the PCG takes time $O(m + n + |\tilde{A}|^2)$. Setting $k = \sqrt{m}$, and assuming that `UltraSparsify` succeeds, we obtain $\kappa_f(A, B) \leq n/m^{1/2}$ and $|\tilde{A}| \leq m^{1/2+o(1)}$. So, the time taken by the PCG algorithm will be $m^{3/4} n^{1/2+o(1)} \log(1/\epsilon)$. □

Alternatively, we may solve the system A_1 by a recursive application of our algorithm. In this case, we let $A_0 = \texttt{Sparsify}(A, 1/2)$, and let B_1 denote the output of `UltraSparsify` on input A_0. Generally, we will let B_{i+1} denote the output of $\texttt{UltraSparsify}(A_i, k_i)$, and let $L_i[D_i, 0; 0, A_i]L_i^T$ be the partial Cholesky factorization of B_i. We will let the recursion depth be r and will specify k_i's later. We solve systems in A_r using an exact method and solve systems in A_i using B_{i+1} preconditioner. At the top level, we will then use A_0 as a preconditioner for A.

In our recursion, all the inner applications of the preconditioned Chebyshev method will run for the same, predetermined, number of iterations. To bound the number of iterations we require, we use the following extension of Joshi ([16]: Corollary 5.5, page 73) of a theorem of Golub and Overton ([12], Theorem 2, page 579).

THEOREM 8.2 (PRECONDITIONED CHEBYSHEV METHOD). *Assume B and A are SDD matrices such that $\sigma(B, A) \geq 1$. Let $\boldsymbol{x}$ be the solution to $A\boldsymbol{x} = \boldsymbol{b}$. Let $\kappa > \kappa_f(A, B)$. Let $\delta = (\kappa - 1)/(200\kappa^2)$. If, in kth iteration of the preconditioned Chebyshev Method when the solution of $B\boldsymbol{z} = \boldsymbol{r}_k$ is needed, a vector $\boldsymbol{z}_k$ is returned satisfying*

$$\|\boldsymbol{z}_k - \boldsymbol{z}\|_B \leq \delta \|\boldsymbol{z}\|_B,$$

then $\boldsymbol{x}_k$ generated by the Preconditioned Inexact Chebyshev Method after k iterations satisfies

$$\|\boldsymbol{x}_k - \boldsymbol{x}\|_A \leq 6\sqrt{\kappa} \left(1 - \frac{1}{\sqrt{\kappa_f(A, B)}}\right)^k \|\boldsymbol{x}\|_A.$$

COROLLARY 8.3. *Under the assumptions of Theorem 8.2, for any $\kappa > \max(8, \kappa_f(A, B))$, after $5\sqrt{\kappa_f(A, B)} \ln \kappa$ iterations the Preconditioned Inexact Chebyshev Method outputs an approximate solution $\tilde{\boldsymbol{x}}$ to $A\boldsymbol{x} = \boldsymbol{b}$ with $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A \leq \delta \|\boldsymbol{x}\|_A$.*

By carefully choosing r and k_i, we obtain the following bound on the time of a recursive algorithm.

THEOREM 8.4 (RECURSIVE). *Let A be an n-by-n SDD matrix with m non-zero entries (assuming $n > 2$). Using the recursive algorithm, one can produce an approximate solution $\tilde{\boldsymbol{x}}$ to $A\boldsymbol{x} = \boldsymbol{b}$ with $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A \leq \epsilon$ in time*

$$m \log^{O(1)} m + O\left(m \log(1/\epsilon)\right) + n2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon)$$

with probability $1 - o(1)$.

PROOF SKETCH. Let c be the constant such that the $\tilde{A}$ output by `UltraSparsify` has at most $k2^{c\sqrt{\log n \log\log n}}$ edges

(as in Theorem 7.8), and let a be the constant hidden in the $O(1)$ in (15).

As $A_0 = \texttt{Sparsify}(A, 1/2)$, A_0 will have $n \log^{c_1} n$ edges for some constant c_1 with high probability.

Let $\kappa_0 = 2^{(2c)\sqrt{\log n \log\log n}} \log^{2a} n$ and $\kappa = \max(8, \kappa_0)$. We will set r so that

$$n^{1/2r} = \kappa_0^{1/4}.$$

As $c \geq 1$, we have that $r \leq \sqrt{\log n / \log\log n}$. We then let $B_{i+1} = \texttt{UltraSparsify}(A_i, k_i)$, where

$$k_i = n^{1-(i+1)/r} / 2^{c\sqrt{\log n \log\log n}}.$$

Thus, for $1 \leq i \leq r$, A_i will have at most $n^{1-i/r}$ edges. Let $n_i = n^{1-i/r}$ for $i = 1 : r$ and $n_0 = n$

When solving system $A\boldsymbol{x} = \boldsymbol{b}$ the recursive algorithm defines a sequence of systems in A_i for $0 \leq i \leq r$. As A_r has a constant size, we solve systems in A_r (and hence their corresponding systems in B_r) using a direct method. Systems in A_i (and their corresponding systems in B_i) with $i < r$ are solved approximately using B_{i+1} as the preconditioner.

By Theorem 7.8, we have

$$\kappa_f(A_i, B_{i+1}) \leq (n_i/k_i) \log^a n = n^{1/r} 2^{c\sqrt{\log n \log\log n}} \log^a n = \kappa_0.$$

By Corollary 8.3, in

$$(5 \ln \kappa)\sqrt{\kappa_f(A_i, B_{i+1})} \leq (5 \ln \kappa) n^{1/r}$$

iterations, an approximate solution with error in A_i-norm that is less than $\delta = (\kappa - 1)/(200\kappa^2)$ can be obtained for the system in A_i. From Theorem 8.2 and its Corollary 8.3, we know that this error is small enough that solutions with this error in A_i can be used to by the preconditioned inexact Chebyshev method to solve systems in A_{i-1}. □

9. REFERENCES

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *33rd ACM STOC*, pages 611–618, 2001.

[2] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, Feb. 1995.

[3] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $O(n^2)$ time. In *28th ACM STOC*, pages 47–55, 1996.

[4] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support-graph preconditioners. *submitted to SIAM J. Matrix Anal. & Appl.*

[5] E. Boman, D. Chen, B. Hendrickson, and S. Toledo. Maximum-weight-basis preconditioners. *to appear in Numerical Linear Algebra and Applications.*

[6] E. Boman and B. Hendrickson. Support theory for preconditioning. *submitted to SIAM J. Matrix Anal. & Appl (Revised 10/02).*

[7] E. Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[8] W. E. Donath and A. J. Hoffman. Algorithms for partitioning graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15(3):938–944, 1972.

[9] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, Sept. 1973.

[10] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In *39th IEEE FOCS*, pages 370–378, 1998.

[11] Z. Füredi and J. Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1(3):233–241, 1981.

[12] G. H. Golub and M. Overton. The convergence of inexact Chebychev and Richardson iterative methods for solving linear systems. *Numerische Mathematik*, 53:571–594, 1988.

[13] K. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, CMU-CS-96-123, 1996.

[14] K. Gremban, G. Miller, and M. Zagha. Performance evaluation of a new parallel preconditioner. In *9th IPPS*, pages 65–69, 1995.

[15] B. Hendrickson and R. Leland. The Chaco user's guide, version 2.0. Tech. Rep. SAND94-2692, Sandia National Labs, Albuquerque, NM, Oct. 1994.

[16] A. Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, UIUC, 1997.

[17] G. Karypis and V. Kumar. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*, Sept. 1998.

[18] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, Nov. 1999.

[19] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, Apr. 1979.

[20] Lovasz and Simonovits. Random walks in a convex body and an improved volume algorithm. *RSA: Random Structures & Algorithms*, 4:359–412, 1993.

[21] B. M. Maggs, G. L. Miller, O. Parekh, R. Ravi, and S. L. M. Woo. Solving symmetric diagonally-dominant systems by preconditioning. 2002.

[22] J. Reif. Efficient approximate solution of sparse linear systems. *Computers and Mathematics with Applications*, 36(9):37–58, 1998.

[23] D. Spielman and S.-H. Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $O(m^{1.31})$. In *44th Annual IEEE FOCS*, pages 416–427, 2003. Most recent version available at `http://arxiv.org/cs.DS/0310036`.

[24] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. available at `http://arxiv.org/abs/cs.DS/0310051`, 2003.

[25] P. M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.