Lecture notes from IPCO Summer School 2005

Fast Randomized Algorithms for Partitioning, Sparsification, and Solving Linear Systems

Daniel A. Spielman

Contents

1 Disclaimer

These are roughly written notes for my lecture. They may contain errors, misattributions, and grammar I would be ashamed to publish in a journal. I hope you find them useful anyway.

2 Overview

In this talk, I am going to describe recent results from a paper by myself and Teng [ST04] on graph partitioning, graph sparsification, and the solution of linear systems. As our main motivation was the solution of linear systems, I will describe our work on that problem first. The algorithms that I will describe solve diagonally dominant linear systems in time nearly-linear in the number of non-zero entries in their defining matrix. In terms of worst-case complexity, these are the fastest known solvers for these systems.

In the second two parts of my lecture, I will describe some of the combinatorial machinery we use in this work. In the first of these, I will discuss the problem of graph sparsification—the problem of replacing an arbitrary graph by a functionally equivalent sparse graph. Our algorithm for sparsifying graphs will use two tools: a graph partitioning algorithm, and some random sampling. The analysis of the random sampling will use an extension of a technique of Füredi and Komlós for bounding the eigenvalues of random symmetric matrices.

In the last part of my lecture, I will discuss the problem of graph partitioning. Our graph partitioning algorithm leverages a graph partitioning algorithm implicit in the work of Lovasz and Simonovits [LS93]. I will spend most of my time explaining the remarkable analysis technique introduced in their work. It is a potential function argument. But, unlike most potential functions that take Real values, the value of their potential function is a concave curve! The potential function can be said to decrease when one of these curves lies under another.

3 Solving Linear Systems

I'll begin by describing our approach to solving linear systems, both because this is probably the most alien topic to the audience, and because it is the motivation for all that follows. I'll begin by reviewing a little bit of what is known about the complexity of solving linear systems. I'll divide the algorithms into *direct methods*, which yield an algebraic procedure that gives the exact answer, and *iterative methods*, which produce answers of greater accuracy the longer they run.

3.1 Direct Methods

The algorithm with the lowest asymptotic complexity for solving arbitrary linear systems of the form $Ax = b$ is based on fast matrix inversion, related to fast matrix multiplication, and has complexity $O(n^{2.376})$ [CW82]. However, if the matrix A is sparse and positive semi-definite, then much faster algorithms exist. These are quite useful because many of the matrices encountered in many application domains are sparse and positive semi-definite. If A is an n-by-n positive semi-definite matrix with m-nonzeros, then the Conjugate Gradient algorithm, used as a direct solver, will solve for x in time $O(mn)$ (See [BT97, Theorem 38.4]). If the matrix is very sparse, by which I mean $m = O(n)$, then this algorithm takes time $O(n^2)$—just as much time as it would even take to write down the inverse! However, sometimes there are ways of implicitly representing the inverse in much less space than it would take to write it explicitly In these situations, one can often find faster algorithms.

Faster algorithms exist if the non-zero entries of the matrix have special structure. To every symmetric matrix A, we will associate the graph in which nodes u and v share an edge if $A_{u,v} \neq 0$. Many of you will have seen a proof that if this graph is a path, then one can solve $Ax = b$ in linear time. This fact extends to the case when the graph is a tree. The idea is to write A as $PLDL^TP^T$ where P is a permutation matrix, and L is a sparse lower-triangular matrix with 1's on the diagonal, and D is a diagonal matrix. The permutation just exists to provide an ordering of the vertices. In this case, we want to order the vertices so that, for every k, if one removes vertices $1, \ldots, k$ the remaining graph is still a connected tree. Given such an ordering, the process to compute L and D looks a lot like Gaussian elimination, but we perform eliminations both in rows and columns. Assuming A is in the proper order, we know that its first vertex has degree 1. Let the non-zero entries in column 1 be $a_{1,1}$ and $a_{u,1}$. We will use the diagonal entry $a_{1,1}$ to eliminate $a_{u,1}$. Note that this changes the value of $a_{u,u}$, but of no other entry. We then use the first column to eliminate entry $a_{1,u}$ from the uth column. This leaves the other entries of the uth column unchanged. If we let A' denote the resulting matrix, and set L to be the matrix with ones on the diagonal and the non-zero entry $l_{1,u} = a_{u,1}/a_{1,1}$, we have

$$A = LA'L^T.$$

One can show that if A is positive definite, then A' is as well, and so we can keep going until we have expressed $A = LL^T$, Moreover, each row of L will have only 2 non-zero entries. Once we have finished, we can solve the system in A by solving systems in P, L, and D, which take time proportional to their number of non-zeros. Let's do an example

This idea was applied to a much broader family of matrices in a paper entitled "Generalized Nested Dissection" by Lipton, Rose and Tarjan [LRT79]. One of their results is that if G is a planar graph, then one can solve $Ax = b$ in time $O(n^{1.5})$. Moreover, the number of non-zeros in L under their ordering is at most $O(n \log n)$.

3.2 Iterative Methods

If the matrix A is positive definite, one can apply the conjugate gradient as an iterative method to solve $Ax = b$. The running time of this algorithm will depend upon the condition number of A, denoted $\kappa(A)$, and defined to be the ratio of the largest eigenvalue of A to the smallest eigenvalue of A. If A has m non-zero entries, then in time $O(m\sqrt{\kappa(A)}\log(1/\epsilon))$, this algorithm will output an ϵ-accurate solution to $Ax = b$ (See [BT97, Theorem 38.5]). By ϵ-approximate solution, I mean an x such that

$$\left\| Ax - b \right\|_A \leq \epsilon \left\| b \right\|_A ,$$

where

$$\|y\|_A \overset{\text{def}}{=} \sqrt{y^T A y}.$$

For those unfamiliar with the A-norm, $\|\cdot\|_A$, let me point out that for every vector y,

$$\|y\|_A \leq \kappa(A)\,\|y\|\,, \text{ and}$$
$$\|y\| \leq \kappa(A)\,\|y\|_A\,.$$

So, if we are willing to run for $\log(\kappa(A)/\epsilon)$ iterations, we can reduce the error to ϵ in the standard Euclidean norm.

At each iteration, the conjugate gradient only needs to multiply a vector by A, and perform a constant number of vector operations. So, it takes time $O(m+n)$ per iteration, and produces the above-mentioned output after at most $O(\sqrt{\kappa}\log(1/\epsilon))$ iterations. While it might seem strange to state the error of the algorithm in terms of the A-norm, it turns out that this is often the best and most useful way to state it.

The Conjugate Gradient method can often be sped up by preconditioning. This amounts to solving a system

$$B^{-1}Ax = B^{-1}b,$$

where B is chosen so $\kappa(B^{-1}A)$ is small, and it is easy to solve systems of the form $Bz = c$. Do not worry that $B^{-1}A$ is not necessarily symmetric. If B is positive definite, then this matrix also has all real eigenvalues. The in each iteration, the Preconditioned Conjugate Gradient will multiply a vector by A, solve a system in B, and perform a few vector operations. After $\sqrt{\kappa(B^{-1}A)}\log(1/\epsilon)$ iterations, it is guaranteed to output an ϵ-approximate solution. So, all we need to do is find a matrix B such that $\kappa(B^{-1}A)$ is small, and systems in B are easy to solve. Such matrices B are called *preconditioners*.

There are many ways to constructing preconditioners, few of them analyzable. One of the most popular is the incomplete Cholesky preconditioner. To obtain this preconditioner, we proceed as if we are going to factor $A = LL^T$, but we drop most of the entries from L. One common rule is to only keep entries of L that are non-zero in A, or to only keep large entries in L. If A is the Laplacian matrix of a regular grid (to be defined in later), then one can prove this preconditioner will allow one to solve the system in time $O(n^{5/4})$.

3.3 Graphs and Diagonally Dominant Matrices

It is now time to explain what all this has to do with graphs. First, I should say that we are only going to solve a limited family of linear systems: those in which the matrix A is symmetric and diagonally dominant. Formally, A is diagonally dominant if, for all i,

$$A_{i,i} \geq \sum_{j \neq i} |A_{i,j}|\,.$$

That is, each diagonal is at least the sum of the absolute entries in its row. It is easy to show that all diagonally dominant matrices are positive semi-definite. They are one of the simplest families of matrices to study, and many approaches to solving linear systems, including Multigrid and Incomplete Cholesky preconditioners, were first studied for these matrices.

Every graph is naturally associated with a diagonally dominant matrix: its Laplacian. For a weighted graph G on n vertices, its Laplacian L is the n-by-n matrix in which $L_{i,j}$ is the negative of the weight of the edge from node i to node j, and the diagonal $L_{i,i}$ is the sum of the weights of the edges connected to vertex i. Thus, all diagonals are non-negative, and all the off-diagonals are non-positive.

Another way to define the Laplacian is as a sum of elementary matrices. Let $(1, 2)$ denote the graph on two vertices with one edge. We define

$$L_{(1,2)} \overset{\text{def}}{=} \begin{bmatrix} 1 & -1 \\ -1 & 1. \end{bmatrix}$$

Note that

$$x^T L_{(1,2)} x = (x_1 - x_2)^2. \tag{1}$$

In general, for the graph with n vertices and just one edge between vertices u and v, we can define the Laplacian similarly. For concreteness, I'll call the graph (u, v) and define it by

$$L_{(u,v)}(i,j) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } i = j \text{ and } i \in u, v \\ -1 & \text{if } i = u \text{ and } j = v, \text{ or } \textit{vice versa}, \\ 0 & \text{otherwise.} \end{cases}$$

For a graph G with edge set E, we now define

$$L(G) \overset{\text{def}}{=} \sum_{(u,v) \in E} L_{(u,v)}.$$

Many elementary properties of the Laplacian follow from this definition. In particular, we see that $L_{G_{1,2}}$ has eigenvalues 0 and 2, and so is positive semidefinite, where we recall that a symmetric matrix M is positive semidefinite if all of its eigenvalues are non-negative. Also recall that this is equivalent to

$$x^T M x \geq 0, \text{ for all } x \in R^n.$$

It follows immediately that the Laplacian of every graph is positive semidefinite. One way to see this is to sum equation (1) to get

$$x^T L_G x = \sum_{(u,v) \in E} (x_u - x_v)^2.$$

It is known [Gre96] and [BGH$^+$, Lemma 2.5] that if one wants to precondition any diagonally dominant matrix, it suffices to find a preconditioner for a related Laplacian. So, for the rest of this section, we will just concern ourselves with Laplacian matrices. Do not worry that the system $Lx = b$ may be degenerate; we can solve it as long as b is orthogonal to $\mathbf{1}$.

Diagonally dominant matrices arise in many applications, including the analysis of resistor networks, heat flow, and random walks on graphs. Recently Boman, Hendrickson and Vavasis [EBV] have proved that the problem of preconditioning the matrices that arise when applying the finite element method solving elliptic partial differential equations can be reduce to the problem of preconditioning diagonally dominant matrices.

One final note before we proceed: if A is a Laplacian matrix, then we cannot measure a condition number by computing A^{-1}, as this matrix does not exist. We can get around this technicality in one of two ways. If A and B have the same span, then we can just view them as operators on their span, and measure their condition number there. The resulting condition number is called the finite condition number.

4 Combinatorial Preconditioners

I will now give you another definition of the finite condition number, and show how it enables us to place combinatorial bounds on the condition number. We will begin by borrowing some terminology from the optimization community. For a symmetric matrix A, we will write

$$A \succcurlyeq 0$$

if A is positive semi-definite. Similarly, we will write

$$A \succcurlyeq B$$

if $A - B$ is positive semi-definite. If G and H are graphs, and L_G and L_H are their Laplacian matrices, then I will write

$$G \succcurlyeq H$$

if

$$L_G \succcurlyeq L_H.$$

If c is a constant, then by $c \cdot G$, I mean the graph in which the weight of every edge has been multiplied by c.

Since the set of positive semi-definite matrices forms a convex cone, it follows that

- for all $c > 0$ and $A \succcurlyeq B$, $cA \succcurlyeq cB$, and
- if $A_1 \succcurlyeq B_1$ and $A_2 \succcurlyeq B_2$, then $A_1 + A_2 \succcurlyeq B_1 + B_2$.

The connection to the condition number is given by the following lemma. [BH]

Lemma 4.1. *Let G and H be connected graphs. Let σ_1 be the least number such that*

$$G \preccurlyeq \sigma_1 H$$

and let σ_2 be the greatest number such that

$$\sigma_2 H \preccurlyeq G.$$

Then,

$$\kappa_f(G,H) = \sigma_1/\sigma_2.$$

Corollary 4.2. *If*

$$c_2 H \preccurlyeq G \preccurlyeq c_1 H,$$

then

$$\kappa_f(G,H) \leq c_1/c_2.$$

So, to bound the condition number, it suffices to prove that some constant times one graph is less than another. I refer to such an inequality as a *Graphic Inequality.* Our constructions of preconditioners will essentially rely on three graphic inequalities. The first two are straightforward, and will be given in the next section. The third is an inequality between a graph G and a graph H obtained by randomly sampling G, and is the subject of Section 6.

4.1 Graphic Inequalities

Here's the simplest graphic inequality. If G is an unweighted graph, and H is a subgraph, then

$$H \preccurlyeq G.$$

A simple generalization says that if G is weighted and H contains the same edges as G, but with lower weights, then $H \preccurlyeq G$. To prove it, let w_e denote the weight of edge e in G, and let $w'_e \leq w_e$. Then, we have

$$\sum_e w'_e L_e \preccurlyeq \sum_e w_e L_e.$$

We will now introduce a more interesting inequality. It's simplest version is

$$L_{(1,k)} \preccurlyeq (k-1) \cdot \left(L_{(1,2)} + L_{(2,3)} + \cdots + L_{k-1,k} \right).$$

This is roughly equivalent to the statement that if one puts k unit resistors in serial, the chain has resistance k. To prove this inequality, we will prove the more general weighted version. In this version, the weights should be interpreted as the reciprocal of the resistance.

Lemma 4.3. *Let $c_1, \ldots, c_{n-1} > 0$. Then,*

$$\left(\frac{1}{1/c_1 + 1/c_2 + \cdots + 1/c_{k-1}} \right) L_{(1,k)} \preccurlyeq \left(\sum_{i=1}^{k-1} c_i L_{(i,i+1)} \right).$$

Proof. We first note that if we can prove this lemma for $k = 3$, then we can prove it for all k. To see this, observe that we could then prove

$$\begin{aligned} \sum_{i=1}^{k-1} c_i L_{(i,i+1)} &\succcurlyeq \left(\frac{1}{\sum_{i=1}^{k-2} (1/c_i)} \right) \cdot L_{(1,k-1)} + c_{k-1} \cdot L_{(k-1,k)} \quad \text{(assuming proved for } k-1\text{)} \\ &\succcurlyeq \left(\frac{1}{\sum_{i=1}^{k-1} (1/c_i)} \right) \cdot L_{(1,k)}, \end{aligned}$$

where in this last inequality, we use the case for $k = 3$.

In the case $k = 3$, it suffices to prove the inequality in the case $1/c_1 + 1/c_2 = 1$. If we now let

$$\begin{aligned} a &= (x_1 - x_2), \text{ and} \\ b &= (x_2 - x_3), \end{aligned}$$

then the inequality reduces to

$$c_1 a^2 + c_2 b^2 \geq (a+b)^2,$$

which is Cauchy's inequality. □

4.2 Graphic Inequalities with Trees

If H is a tree[1] with the same set of vertices as G, then there is a very natural way to apply Lemma 4.3 to prove an inequality of the form $G \preccurlyeq c \cdot H$. For every edge $(u,v) \in G$, there is unique path in H between vertices u and v. Let $P_{(u,v)}$ denote this path, and let $l_{u,v}$ denote its length. For each edge $(x,y) \in H$, let $s_{(x,y)}$ be the sum of the lengths of the paths $P_{u,v}$ that use it,

$$\sum_{(u,v) \in G : (x,y) \in P_{u,v}} l_{u,v},$$

and let s be the sum of the lengths of all paths,

$$s = \sum_{(u,v) \in G} l_{u,v},$$

Note that $s_{(x,y)} \leq s$. The quantity s is called the *total stretch* of the spanning tree H.

For simplicity, let's assume that G and H are unweighted. By inequality 4.3, we have

$$(u,v) \preccurlyeq l_{u,v} P_{(u,v)},$$

and so

$$G = \sum_{(u,v) \in G} (u,v) \leq \sum_{(u,v) \in G} l_{u,v} P_{(u,v)} = \sum_{(x,y) \in H} s_{(x,y)} L_{(x,y)} \leq \sum_{(x,y) \in H} s L_{(x,y)} = s \cdot H.$$

That is, $G \preccurlyeq s \cdot H$ where s is the sum over all edges of G of the lengths of their paths in H. This inequality was proved in [BH]..., A year later, Boman and Hendrickson [BH01] observed that Alon, Karp, Peleg and West [AKPW95] found an algorithm for constructing trees H that were subgraphs of G and that have total stretch $n 2^{O(\sqrt{\log n \log \log n})}$. Thus, these subtrees are preconditioners with condition number at most $n 2^{O(\sqrt{\log n \log \log n})}$. These can immediately be used to find ϵ-accurate solutions to diagonally-dominant linear systems in time $mn^{1/2+o(1)}$. We note that Elkin, Emek, Spielman and Teng [EEST05] have recently found fast algorithms for constructing subtrees of average stretch $O(n \log^2 n \log \log n)$.

[1]A tree is a connected graph on n vertices with $n-1$ edges. Alternatively, it can be defined as a connected graph with no cycles.

4.3 Augmented Tree Preconditioners

Vaidya [Vai90] used maximum spanning trees to precondition graphs. However, these did not immediately produce faster algorithms, and the best bound one can prove on their relative condition numbers is $O(mn)$. This led Vaidya to consider adding edges to the spanning trees to improve the condition number. By adding just a few edges, he was able to greatly reduce the condition number. On the other hand, adding a few edges to a tree does not greatly increase the time it takes to solve the linear system.

Spielman and Teng [ST03] devised a way to add edges to a low-stretch spanning tree to reduce $\max_{u,v} s_{u,v}$. I will now explain a simplification of their technique that will appear in the final version of [ST04]. We begin by observing that our derivation before was very weak. We could have obtained the inequality [ST03, Theorem 2.1]

$$G \preccurlyeq \left(\max_{(x,y)\in H} s_{(x,y)} \right) \cdot H.$$

The constant in this inequality bounds the maximum over edges in H of the sum of the lengths of the paths that go over that edge. By adding some edges to the tree, we create new paths that can be used to route edges of G, and thereby reduce this sum.

Our rule for adding edges to the tree will be quite simple. First, we will partition the nodes of the tree into sets $S_1, \dots, S_t$ so that the induced graph on each set S_i is a tree. Then, for each pair of sets S_i, S_j such that G contains an edge between a node in S_i and a node in S_j, we will add one edge to H. Now, I need to tell you how we choose the sets and how we add the edge.

For each vertex v, set

$$\operatorname{cost}(v) \stackrel{\text{def}}{=} \sum_{(u,v)\in G} l_{u,v}.$$

Note that $\sum_v \operatorname{cost}(v) \leq 2s$. For a set S of vertices, define $\operatorname{cost}(S) = \sum_{v\in S} \operatorname{cost}(v)$. Using a greedy, DFS-based, algorithm, it is possible to find at most t sets so that for all i, either

a. $\operatorname{cost}(S_i) \leq 4s/t$, or

b. $|S_i| = 1$.

If you want to see the details, look at [ST03, Lemma 3.6].

Warning 4.4. *I have lied a little bit here. To prove a bound this good, you have to allow the trees to overlap slightly. But, you'll get the correct intuition from this argument.*

For each pair of sets (S_i, S_j) that are connected by an edge in G, we will select the edge between these sets of minimal stretch, and add that edge to H. (in the weighted case, we minimize the stretch divided by weight). Let u_i and u_j be the endpoints of this edge. Now, for every other edge (v_i, v_j)in G between these two sets, we will imagine routing this edge locally: from v_i to u_i in S_i, over the *bridge* edge (u_i, u_j), and from u_j to v_j in S_j. Formally, this becomes a graphic inequality proving that the edge from v_i to v_j is less than some constant times a path that lies entirely in S_i,

S_j, and (u_i, u_j). In this way, we can make sure that no edge of H is used too many times in the inequalities we prove.

To get a feel for how good the overall inequality is, note that the length of the path from v_i to u_i plus the length of the path from u_j to v_j is at most

$$l_{v_i,v_j} + l_{u_i,u_j} \leq 2l_{v_i,v_j}.$$

So, the length of the path in H from v_i to v_j is at most $2l_{v_i,v_j} + 1$. If we now fix i and sum over j of all such inequalities between edges between S_i and S_j, we will find that the sum of all such path lengths is at most 3 times $\text{cost}(S_i)$. As these are the only inequalities involving edges of S_i, we find that each edge in S_i will have a coefficient of at most $3\text{cost}(S_i) \leq 12s/t$. So, we have

$$G \preccurlyeq (12s/t) \cdot H.$$

This is a very strong result: it says that by dividing H up into t sets, we can reduce the relative condition number by a factor of t. Now, it remains to consider how many edges we have added to the tree.

If H were planar, then we would have added only $O(t)$ edges. This would be ideal.

On the other hand, it is possible that might have to add edges between all pairs of subsets, for a total of $\binom{t}{2}$. In this case, the running time of our linear system solver will be no better than $m^{1.31+o(1)}$, as it was in [ST03]. However, it is never necessary to really add this many edges–there is always a set of $t \log^{O(1)} t$ edges that suffice, and we can find them in time $t \log^{O(1)} t + O(m)$. We do this by *sparsifying* the graph induced by G on the clusters $S_1, \ldots, S_t$. To form this graph, we treat the clusters $S_1, \ldots, S_t$ as vertices, and put an edge between each pair that has an edge in G. A *sparsifier* for this graph is a sparse graph that is functionally equivalent. Most of the work of our paper [ST04] is devoted to an algorithm for sparsifying graphs.

4.4 Sparsifying

We say that a weighted graph H κ-approximates a weighted graph G if

$$H \leq G \leq \kappa H.$$

Our main sparsification theorem says that for every weighted graph G, there exists a weighted graph H that 2-approximates G and has at most $n \log^{O(1)} n$ edges. This notion of sparsification is sufficient to reduce the number of intercluster edges in the construction of the previous section. We then obtain a graph H with $n + t \log^{O(1)} t$ edges such that (n/t)-approximates G.

The notion of sparsification was introduced in the papers .... The sparsifier that inspired ours was a construction of Benczur and Karger [BK96]. They gave a nearly-linear time algorithm that produced for every graph G a graph H with $n \log^2 n$ edges (check!) such that for every set of vertices S, the sum of the weights of edges leaving S in G is approximately equal to the sum in H. In matrix notation, this says

$$\forall x \in \{0,1\}^n : x^T L_H x \leq x^T L_G x \leq (1+\epsilon) x^T L_H x.$$

This is quite similar to the statement that we need to prove, except that we require these inequalities for all $x \in \mathbb{R}^n$. While we cannot use sparsifiers constructed by Benczur and Karger, we can use some ideas from their work. In particular, we note that for some graphs we can obtain a sparsifier by random sampling.

Before explaining how we construct sparsifiers, let me begin with three examples.

- **The complete graph:** Let G be the complete graph on n vertices. One can show that L_G only has two eigenvalues: 0 with multiplicity 1 and n with multiplicity $n-1$. Let H be a Ramanujan expander graph [Mar88, LPS88]. So, H will have degree d, and all non-zero eigenvalues of L_H will lie between $d - 2\sqrt{d-1}$ and $d + 2\sqrt{d-1}$. The graph $(n/d) \cdot H$ is a good approximation of G. We have

$$G \preccurlyeq \frac{d}{d - 2\sqrt{d-1}}(n/d) \cdot H,$$

  and

$$\frac{d}{d + 2\sqrt{d-1}}(n/d) \cdot H \preccurlyeq G.$$

- **The dumbbell:** Let G be the disjoint union of two cliques, each on $n/2$ vertices, with one edge added between them. Note that if $H \subset G$ and H does not contain the edge between the two cliques, then H will be disconnected. In that case, there would be no true inequality of the form $G \preccurlyeq c \cdot H$. So, however we sparsify G, we need to be sure to include the edge between the cliques.

- **Grid plus edge:** Let G be the graph on a m-by-n grid, with vertices indexed by (i,j) for $1 \leq i \leq m$ and $1 \leq j \leq n$, plus one edge from node $(1,1)$ to node $(m,1)$. If $m = n^2$, then any subgraph H that does not contain the edge $(m,1)$ will have $\kappa_f(G,H) \geq n$. To see why, let H be the graph consisting of all edges of G besides $(m,1)$, and consider the vector $x_{(i,j)} = i$. To evaluate $x^T L_H x$, note that each edge from a node (i,j) to a node $(i+1,j)$ will contribute 1 to the sum, and that there are $n(m-1)$ such edges, so

$$x^T L_H x = n(m-1).$$

  On the other hand, the edge from node $(1,1)$ to $(m,1)$ contributes $(m-1)^2$ to this sum in G. So,

$$x^T L_G x = (n+m-1)(m-1),$$

  and so

$$\kappa_f(L_G, L_H) \geq \frac{m+n-1}{n} \geq \frac{m}{n} = n.$$

  This example also shows how our notion of sparsification differs from that studied by Benczur and Karger, as

$$\forall x \in \{0,1\}^n : x^T L_H x \leq x^T L_G x \leq (3/2) x^T L_H x.$$

4.5 Proving sparsifiers exist

To begin, I'll convince you that it is possible to sparisfy any graph. I'll then explain how we do it quickly. The key to it all is a quantity called *conductance*, which I will formally define in a few moments. Roughly, the conductance of a graph is the minimum over cuts of the number of edges cut divided by the smaller side. Conductance is my favorite graph parameter, because every value it has is useful. If a graph has high conductance, that means it is an expander—one of the most useful types of graphs around. If it has low conductance, then it can painlessly be cut into two pieces.

In our case, we will prove that every graph of high conductance can be well approximated by a sparse random subgraph. On the other hand, we will prove that the vertices of every graph can be partitioned into sets $V_1, \ldots, V_k$ so that at most half of the edges cross the partition, and the induced graph on each vertex set V_i has high conductance. These two observations suffice to prove the existence of sparsifiers: we sparsify the induced graphs by random sampling, and then recursively sparsify the graph consisting of the edges crossing the partition. As at most half of the edges cross the partition, the depth of the recursion is at most $\log_2 m$.

Now, let me define conductance formally. I will define it in terms of the adjacency matrix of a graph, A, and I will let $a_{u,v}$ denote the weight of the edge from node u to node v. For each vertex u, I define its degree, d_u, by

$$d_u \stackrel{\text{def}}{=} \sum_v a_{u,v}.$$

For a set of vertices, S, I define its volume by

$$\text{vol}(S) \stackrel{\text{def}}{=} \sum_{v \in S} d_v,$$

and I define the volume of a set of edges F by

$$\text{vol}(F) \stackrel{\text{def}}{=} \sum_{(u,v) \in F} a_{u,v}.$$

Finally, I define the boundary of a set of vertices S by

$$\partial(S) \stackrel{\text{def}}{=} \{(u,v) \in E : u \in S, v \notin S\}.$$

We now define the *conductance* of a cut $S \subset V$ by

$$\Phi(S) \stackrel{\text{def}}{=} \frac{\text{vol}(\partial(S))}{\min(\text{vol}(S), \text{vol}(\bar{S}))}.$$

And, we define the *conductance* of a graph by

$$\Phi_G \stackrel{\text{def}}{=} \min_{S \subset V} \Phi(S).$$

I will call a partition $V_1, \ldots, V_k$ of the vertices of a graph a ϕ-partition if the induced subgraph on each V_i has conductance at least ϕ. The following lemma tells us that, assuming one could find the largest set of conductance less than ϕ, it is possible to find a ϕ-partition:

Lemma 4.5. *For any $\phi \leq \Phi(G)$, let S be the set of vertices satisfying $\Phi(S) \leq \phi$ maximizing* $\mathrm{vol}(S)$, *subject to* $\mathrm{vol}(S) \leq \mathrm{vol}(V)/2$. *If* $\mathrm{vol}(S) \leq \mathrm{vol}(V)/4$, *then*

$$\Phi_{G(\bar{S})} \geq \phi/3,$$

where $G(\bar{S})$ denotes the graph induced on the vertices not in S.

That is, if $\mathrm{vol}(S) \leq \mathrm{vol}(V)/4$, then we can make all the vertices not in S a set in a our partition.

Using this lemma, it is easy to prove that, for every $\phi \geq 1/m$, a ϕ-partition exists in which the sum of the weights of the edges crossing the partition is at most $O(\phi \mathrm{vol}(V) \log n)$: let S be the largest set of vertices with at most half the volume and conductance less than 3ϕ. Partition the graph into $(S, \bar{S})$. Now, recursively apply the same procedure on each part of the partition. In the unweighted case, it is easy to see that this procedure will have recursion depth at most $\log_{4/3} m$. In the weighted case, we observe that every set that occurs after $2 \log_{4/3} m$ steps down in the recursion can have at most a $1/m^2$ fraction of the volume. So, whatever edges are cut by dividing these sets up can contribute at most a n/m^2 fraction to the total.

Of course, finding the cut required by Lemma 4.5 is a hard problem. So, we will apply an algorithm that satisfies a weaker guarantee. In the next section, I will explain how we can find maximal cuts of approximately minimal conductance in nearly-linear time, for the case $\phi = 1/\log^{O(1)}$, which is the case we care about here. Kannan, Vempala and Vetta [KVV04] have analyzed how a partitioning algorithm like the one above behaves if one uses any algorithm that can find cuts of approximately optimal conductance. Their analysis is more complicated as the recursion depth is no longer bounded, as Lemma 4.5 does not hold for approximately maximal cuts or cuts of approximately minimal conductance. This also poses problems for us, as a recursion depth greater than $\log^{O(1)} n$ would make our algorithm take too long. However, in the paper we are able to show that, for the purpose of sparsification, it suffices to cut the recursion at depth $\log^{O(1)} n$, regardless of whether or not the partition at that depth is a ϕ-partition ([ST04, Theorem 4.1] and [ST04, Lemma 6.1]).

4.6 History of Combinatorial Preconditioning

Vaidya [Vai90] was the first to suggest preconditioning Laplacians of graphs by Laplacians of their subgraphs. He proved that by augmenting spanning trees with a few edges, one could find ϵ-approximate solutions to SDD linear systems of maximum valence d in time $O((dn)^{1.75} \log(\kappa_f(A)/\epsilon))$, and of planar linear systems in time $O((dn)^{1.2} \log(\kappa_f(A)/\epsilon))$. While Vaidya's work was unpublished, proofs of his results as well as extensions may be found in [Jos97, Gre96, GMZ95, BGH$^+$, BCHT, BH]. By recursively applying Vaidya's preconditioners, Reif [Rei98] improved the running time for constant-valence planar linear systems to $O(n^{1+\beta} \log^{O(1)}(\kappa_f(A)/\epsilon))$, for every $\beta > 0$. Boman and Hendrickson [BH01] used low-stretch spanning trees to improve the running time for general linear systems to $m^{1.5+o(1)} \log(\kappa_f(A)/\epsilon)$. Maggs, *et. al.* [MMP$^+$05] to $O(m/\sqrt{\Phi_G} \log^4(n\kappa_f(A)/\epsilon))$, after some preprocessing. Their algorithm follows an approach introduced by Gremban, Miller and Zagha in which vertices are *added* to the graph.

By augmenting low-stretch spanning trees, by Spielman and Teng [ST03] obtained a running time of $m^{1.31+o(1)} \log(1/\epsilon) \log^{O(1)}(n/\kappa_f(A))$.

5 Cheeger's Inequality

Before I explain how to analyze the quality of random subsamples of graphs as preconditioners, I should explain why conductance should be related to preconditioning at all. The explanation comes from one of the most important theorem in Spectral Graph Theory: Cheeger's Inequality [Che70]. The inequality that Cheeger proved concerned manifolds. The version of Cheeger's inequality for graphs that we use here was proved by Jerrum and Sinclair [SJ89].

Cheeger's inequality establishes a relationship between the conductance of a graph and the smallest non-zero eigenvalue of its normalized Laplacian. To obtain the normalized Laplacian, let let L denote the Laplacian of a graph, and let D be the diagonal matrix with diagonal entries $(d_1, \ldots, d_n)$, where d_i is the degree of node i. Then the normalized Laplacian (See [Chu97]), often written $\mathcal{L}$ is $D^{-1/2}LD^{-1/2}$. All the diagonal entries of this matrix are 1, it's smallest eigenvalue is zero, and we will write λ_2 for its second-smallest eigenvalue. It's largest eigenvalue is at most 2.

Cheeger's inequality says:

$$\frac{1}{2}\Phi_G^2 \leq \lambda_2 \leq \Phi_G. \tag{2}$$

The upper bound on λ_2 is easy: one proves it by constructing a test vector from a set of minimum conductance. The other direction is non-trivial. It would be very interesting if someone could come up with a novel proof of the other direction. I put quite a bit of effort into this, but only found a somewhat different proof, which is in the lecture notes from my course Applied Extremal Combinatorics (see `http://www-math.mit.edu/ spielman/AEC/lect5.ps`).

Cheeger's inequality tells us that if Φ_G is big, then all of the eigenvalues of the normalized Laplacian lie in a small range.

Finally, let me note that Cheeger's inequality is sometimes stated in different forms. Sometimes, the matrix $D^{-1}L$ appears instead of $D^{-1/2}LD^{-1/2}$; but, these have the same eigenvalues.

Sometimes, Cheeger's inequality is stated in terms of the normalized adjacency matrix, $D^{-1/2}(A + D)D^{-1/2}/2$, where the adjacency matrix is the matrix with zero diagonals and entry $A_{i,j}$ equal to the weight of the edge from i to j. This is the matrix that arises when studying random walks on the graph. In this case, the theorem concerns the second-largest eigenvalue. All these formulations are equivalent.

However, there are times when it is legitimately more convenient to work with one of these matrices as opposed to another.

6 Randomly Sampling Graphs

Let A be the adjacency matrix of a (possibly weighted) graph G. In a few moments, we will give a procedure for randomly sampling G to obtain a graph with adjacency matrix $\tilde{A}$ such that all the eigenvalues of $D^{-1}(A - \tilde{A})$ are small, where D is the diagonal matrix of the degrees of A. If we let L and $\tilde{L}$ be the Laplacian matrices of these graphs, it is pretty easy to also show that all the eigenvalues of $D^{-1}(L - \tilde{L})$ are small.

To make this statement more useful, we observe that the matrices

$$D^{-1}(L - \tilde{L}) \quad \text{and} \quad D^{-1/2}(L - \tilde{L})D^{-1/2}$$

have the same eigenvalues, and so all the eigenvalues of the later matrix are small. However, only in special circumstances does this imply that $D^{-1/2}\tilde{L}D^{-1/2}$ is a good preconditioner for $D^{-1/2}LD^{-1/2}$. One such circumstance is when the largest eigenvalue of $D^{-1/2}(L - \tilde{L})D^{-1/2}$ is much smaller than the smallest non-zero eigenvalue of $D^{-1/2}LD^{-1/2}$. For example, assume

$$\begin{aligned}\lambda_{min}\left(D^{-1/2}LD^{-1/2}\right) &\geq \alpha, \text{ and} \\ \lambda_{max}\left(D^{-1/2}(L - \tilde{L})D^{-1/2}\right) &\leq \epsilon.\end{aligned}$$

We would then have for all x orthogonal to the nullspace of $D^{-1/2}LD^{-1/2}$,

$$\begin{aligned}\frac{x^T D^{-1/2}\tilde{L}D^{-1/2}x}{x^T D^{-1/2}LD^{-1/2}x} &= \frac{x^T D^{-1/2}LD^{-1/2}x + x^T D^{-1/2}(\tilde{L} - L)D^{-1/2}x}{x^T D^{-1/2}LD^{-1/2}x} \\ &= 1 + \frac{x^T D^{-1/2}(\tilde{L} - L)D^{-1/2}x}{x^T D^{-1/2}LD^{-1/2}x} \\ &\leq 1 + \frac{\epsilon}{\alpha}.\end{aligned}$$

So, if $\alpha > 2\epsilon$, $D^{-1/2}\tilde{L}D^{-1/2}$ is a good preconditioner for $D^{-1/2}LD^{-1/2}$. By Cheeger's inequality, we know that the case when α is not too small is exactly when G has high conductance.

But, you might be wondering how this helps us. After all, we want to precondition L by $\tilde{L}$, without the D^{-1} interfering. Let me quickly point out that the D^{-1} term does not interfere. That is,

$$D^{-1/2}LD^{-1/2} \preccurlyeq \kappa \cdot D^{-1/2}\tilde{L}D^{-1/2} \quad \iff \quad L \preccurlyeq \kappa \cdot \tilde{L}.$$

I recommend proving this as an exercise.

6.1 The Sampling Procedure

To build $\tilde{A}$, we will independently flip a coin for each edge of A and, based on the outcome, decide whether or not to keep that edge. The coins will be biased, and if we decide to keep an edge, we will assign it a weight inversely proportional to our probability of keeping it. We apply these weights so that the expectation of $\tilde{A}$ is A. In particular, this means that the expected weighted degree of each node remains approximately the same. We need to make sure that the expected number of neighbors of each node is large enough that with high probability no node becomes isolated. On the other hand, to reduce the total number of edges we must make sure to get rid of most edges from high degree nodes. To meet both of these objectives, we choose a parameter δ that will govern the number of edges in $\tilde{A}$, and decide to keep an edge between nodes i and j with probability $p_{i,j}$, where

$$p_{i,j} = \begin{cases} \frac{\delta}{\min(d_i, d_j)} & \text{if } \delta < \min(d_i, d_j), \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

We then set

$$\tilde{A}_{i,j} = \begin{cases} A_{i,j}/p_{i,j} & \text{with probability } p_{i,j} \text{ and} \\ 0 & \text{with probability } 1 - p_{i,j}. \end{cases}$$

In [ST04] we prove

Theorem 6.1 (Random Subgraph). *For all $\alpha \geq 1$, and even integers k,*

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

This theorem is particularly helpful when we choose k to be approximately $\log_2 n$, in which case $n^{1/k}$ becomes 2. If δ is then larger than $\log^2 n$, the bound becomes meaningful.

Our proof of this theorem applies a modification of techniques introduced by Füredi and Komlós [FK81] (See also the paper by Vu [Vu05] that corrects some bugs in their work). However, they consider the eigenvalues of random graphs in which every edge can appear. Some interesting modifications are required to make an argument such as ours work when downsampling a graph that may already be sparse. In this lecture, we will just consider unweighted graphs, as the proof for this case already contain all the interesting details. We will allow the graphs to be irregular.

Remark 6.2. *A good problem would be to find a sampling scheme under which one can improve upon this bound. Perhaps one can do it by forcing each node to have at least some minimal number of neighbors.*

6.2 Analysis

Let's first observe that $\tilde{A}$ is probably sparse.

Lemma 6.3. *The expected number of edges in $\tilde{A}$ is at most δn.*

Proof. Let $X_{i,j}$ be a random variable that is 1 if we keep the edge between nodes i and j, and 0 otherwise. Then, the expected number of edges is

$$\mathbf{E}\left[\sum_{(i,j)\in E} X_{i,j}\right] = \sum_{(i,j)\in E} \frac{\delta}{\min(d_i, d_j)} \leq \sum_{(i,j)\in E} \left(\frac{\delta}{d_i} + \frac{\delta}{d_j}\right) = \delta n.$$

□

One can also use a modified Hoeffding inequality to prove that the number of edges is highly concentrated around its expectation.

To simplify our notation, let's define

$$\Delta = D^{-1}(\tilde{A} - A),$$

so

$$\Delta_{i,j} = \begin{cases} \frac{1}{d_i}(\frac{1}{p_{i,j}} - 1) & \text{with probability } p_{i,j}\text{, and} \\ -\frac{1}{d_i} & \text{with probability } 1 - p_{i,j}. \end{cases}$$

Rather than trying to upper bound the largest eigenvalue of Δ directly, we will upper bound a power of it's trace. Recall that the trace of a matrix is the sum of its eigenvalues and also the sum of its diagonal entries. So, for every even power k $\mathrm{Tr}\left(\Delta^k\right)$ is an upper bound on λ_{max}^k, and as k grows the relation between the two becomes tighter.

The technical part of our argument will involve proving the following bound on the trace.

Lemma 6.4. *For even k,*

$$\mathbf{E}\left[\mathrm{Tr}\left(\Delta^k\right)\right] \leq \frac{n(2k)^k}{\delta^{k/2}}.$$

From this lemma, we may immediately prove the main theorem.

Proof of Theorem 6.1. Lemma 6.4 implies that, for even k,

$$\mathbf{E}\left[\lambda_{max}\left(\Delta^k\right)\right] \leq \frac{n(2k)^k}{\delta^{k/2}}.$$

Applying Markov's inequality, we obtain

$$\mathrm{P}\left[\lambda_{max}\left(\Delta^k\right) > \alpha^k \frac{n(2k)^k}{\delta^{k/2}}\right] \leq 1/\alpha^k.$$

Recalling that $\lambda_{max}(\Delta^k) = \lambda_{max}^k(\Delta)$, and taking k-th roots, we conclude

$$\mathrm{P}\left[\lambda_{max}\left(\Delta\right) > \alpha \frac{n^{1/k}(2k)}{\delta^{1/2}}\right] \leq 1/\alpha^k.$$

□

6.3 The Combinatorial Bounds on the Trace

To bound the trace, we will use the following expression for entries of Δ^k.

$$\left(\Delta^k\right)_{v_0,v_k} = \sum_{v_1,\ldots,v_{k-1}} \prod_{i=1}^{k} \Delta_{v_{i-1},v_i}.$$

We obtain an analogous expression for the expectation.

$$\mathbf{E}\left[\left(\Delta^k\right)_{v_0,v_k}\right] = \sum_{v_1,\ldots,v_{k-1}} \mathbf{E}\left[\prod_{i=1}^{k} \Delta_{v_{i-1},v_i}\right]. \tag{3}$$

The first idea in the analysis is to observe that most of the terms in this latter sum are zero. The reason is that, for all $v_i \neq v_j$

$$\mathbf{E}\left[\Delta_{v_i,v_j}\right] = 0.$$

As Δ_{v_i,v_j} is independent of every term in Δ other than Δ_{v_j,v_i}, we see that the term corresponding to $v_1, \dots, v_{k-1}$ will be zero unless each edge (v_{i-1}, v_i) appears at least twice. Of course, any term involving an edge (v_{i-1}, v_i) for which Δ_{v_{i-1},v_i} is always zero will also contribute zero to the sum.

We will now devise a way of describing every sequence $v_1, \dots, v_{k-1}$ that could possibly contribute to the sum (we will refer to such a sequence as a walk). We set S to be the set of time steps i such that the edge between v_{i-1} and v_i does not appear earlier in the walk. We then let τ denote the map from $[k] - S \to S$, indicating for each time step not in S the time step in which the edge (v_{i-1}, v_i) first appeared (regardless of in which direction it is traversed). We let $p = |S|$, and note that we need only consider the cases in which $p \leq k/2$, as otherwise some edge appears only once in the walk. To finish our description of a walk, we need a map

$$\sigma : S \to \{1, \dots, n\},$$

indicating the vertex encountered at time i.

Let's show that S, τ and σ are enough to reconstruct $v_1, \dots, v_{k-1}$. We know that the walk starts at node v_0. We will now show that if we know v_{i-1}, then we can figure out v_i. If $i \in S$, then v_i is just $\sigma(i)$. If $i \notin S$, then we use τ to determine which edge we should traverse. One of its endpoints will be v_{i-1}, and the other will be v_i.

The tuple (S, τ, σ) can encode many walks that are not realizable. We will call σ a valid assignment for S and τ if if it corresponds to a walk on the non-zero variables of Δ. Formally, the assignment is *valid* if

- For each $i \in S$, $\sigma(i)$ is a neighbor of v_{i-1} in A, and $p_{v_{i-1},\sigma(i)} \neq 0$. That is, the corresponding entry in Δ has some chance of being non-zero.
- For each $i \notin S$, $v_{i-1} \in \left\{v_{\tau(i)-1}, v_{\tau(i)}\right\}$.

We have

$$\mathbf{E}\left[\left(\Delta^k\right)_{v_0,v_k}\right] = \sum_{S,\tau} \sum_{\text{valid } \sigma} \mathbf{E}\left[\prod_{i=0}^{k-1} \Delta_{v_i,v_{i+1}}\right], \text{where } (v_1, \dots, v_{k-1}) \text{ is the sequence encoded}$$

$$= \sum_{S,\tau} \sum_{\text{valid } \sigma} \prod_{s \in S} \mathbf{E}\left[\Delta_{v_{s-1},v_s} \prod_{i:\tau(i)=s} \Delta_{v_{i-1},v_i}\right],$$

where $v_1, \dots, v_{k-1}$ is the sequence encoded by (S, τ, σ). Each of the terms

$$\mathbf{E}\left[\Delta_{v_{s-1},v_s} \prod_{i:\tau(i)=s} \Delta_{v_{i-1},v_i}\right]$$

is independent of the others, and involves a product of the terms Δ_{v_{s-1},v_s} and $\Delta_{v_s,v_{s-1}}$. Below, we will prove that

$$\mathbf{E}\left[\Delta_{v_{s-1},v_s}\prod_{i:\tau(i)=s}\Delta_{v_{i-1},v_i}\right]\leq\frac{1}{\delta^{|\{i:\tau(i)=s\}|}}\frac{1}{d(v_{s-1})}. \tag{4}$$

This is very useful, because there are exactly $d(v_{s_j-1})$ ways that we could choose $\sigma(j)$ given that the choices for $\sigma(1),\ldots,\sigma(j-1)$ lead us to reach vertex v_{s_j-1} at step s_j-1. Thus, the presence of $d(v_{s_j-1})$ in the denominator allows us to cancel out the sum over the choices for σ. This is the part of the argument that is special for sparse graphs A. We thereby obtain

$$\sum_{\text{valid }\sigma}\prod_{s\in S}\mathbf{E}\left[\Delta_{v_{s-1},v_s}\prod_{i:\tau(i)=s}\Delta_{v_{i-1},v_i}\right]\leq\frac{1}{\delta^{k-p}}.$$

As there are n choices for v_0, (we only care about the terms where $v_k=v_0$) at most 2^k choices for S, and at most k^k choices for τ, we have

$$\mathbf{E}\left[\operatorname{Tr}\left(\Delta^k\right)\right]\leq\frac{n(2k)^k}{c^{k/2}}.$$

Let's also observe that $\Delta_{i,j}$ is never too big:

Claim 6.5.

$$|\Delta_{i,j}|\leq 1/\delta.$$

Proof. In order for $\Delta_{i,j}$ to not be fixed to zero, it must be the case that $\delta<\min(d_i,d_j)$. So, $1/d_i<1/\delta$, which takes care of one case. On the other hand,

$$\frac{1}{d_i}\frac{1}{p_{i,j}}=\frac{\min(d_i,d_j)}{d_i\delta}\leq\frac{1}{\delta}.$$

Now, subtracting $1/d_i$ from this term cannot make its magnitude exceed $1/\delta$ because $1/d_i<1/\delta$. ☐

Lemma 6.6. *For all edges (t,r) and integers $k\geq 1$ and $l\geq 0$ such that $k+l\geq 2$,*

$$\mathbf{E}\left[\Delta_{r,t}^k\Delta_{t,r}^l\right]\leq\frac{1}{c^{k+l-1}}\frac{1}{d_r}.$$

Proof. To make life easier, we'll just prove this in the case $k+l=2$. The other cases are similar. We note that $\Delta_{r,t}=\tilde{A}_{r,t}/d_r$ and $\Delta_{t,r}=\tilde{A}_{r,t}/d_t$, so we can compute

$$\begin{aligned}\mathbf{E}\left[\Delta_{r,t}^k\Delta_{t,r}^l\right]&=\frac{1}{d_r^kd_t^l}\left(p_{r,t}\left(\frac{1-p_{r,t}}{p_{r,t}}\right)^2+(1-p_{r,t})\right)\\&=\frac{1}{d_r^kd_t^l}\left(\frac{1-p_{r,t}}{p_{r,t}}\right)\\&\leq\frac{1}{d_r^kd_t^l}\left(\frac{1}{p_{r,t}}\right)\\&=\frac{1}{d_r^kd_t^l}\left(\frac{\min(d_r,d_t)}{\delta}\right).\end{aligned}$$

To finish the proof, we observe

$$\frac{\min(d_r, d_t)}{d_r d_t} = \frac{1}{\max(d_r, d_t)} \leq \frac{1}{d_r},$$

and

$$\frac{\min(d_r, d_t)}{d_r^2} \leq \frac{1}{d_r}.$$

□

7 Graph Partitioning

In this section, I will explain how our nearly-linear time graph partitioning algorithm works. It will leverage a little-known algorithm implicit in the work of Lovasz and Simonovits. First, let me review the better-known algorithms and their drawbacks.

- Algorithms based on linear and semi-definite programming [LR99, ARV04, AHK04]. These algorithms provide very good approximations to the cut of minimal conductance. However, their running times are at least $\Omega(n^2)$.
- Spectral algorithms. These make a cut using the second-smallest eigenvector of the Laplacian, an approach we will discuss more in the next section. These algorithms can be made fast enough if we are only concerned with the case $\phi = 1/\log^{O(1)}$. However, they cannot produce cuts of almost optimal balance. In fact, it is very difficult to control the balance of the cuts made by these algorithms, and we do not know how to use them to produce cuts of approximately optimal balance in time less than $\Omega(n^2)$.
- Multilevel methods. These are implemented in the packages Chaco [HL94] and Metis [KK98]. They work very well in practice. But, it has not yet been proved that these algorithms have good worst-case behavior.

The approach indicated by Lovasz and Simonovits is based on an examination of the distributions of random walks in the graph. While the algorithm it not necessarily fast, we will be able to modify it to obtain a $n \log^{O(1)} n$ time algorithm. To begin, let's define precisely how we get a random walk from a weighted graph. We will make our definition from the adjacency matrix of the graph, A, and we will allow A to have self-loops, which corresponding to diagonal entries in A.

When our random walk is at a vertex u, it will go to node v with probability proportional to $a_{u,v}$:

$$m_{u,v} \stackrel{\text{def}}{=} \frac{a_{u,v}}{\sum_w a_{u,w}}.$$

So that $m_{u,v}$ can be the probability of moving from v to w, I am going to have to do something I hate: multiplying by row-vectors from the left. So, if p is a probability distribution, then pM is the probability distribution obtained after one step.

In matrix notation, we can form the matrix of probabilities, M, by setting

$$\begin{aligned}
d_u &\overset{\text{def}}{=} \sum_w a_{u,w} \\
D &\overset{\text{def}}{=} \operatorname{diag}(d_1, \ldots, d_n) \\
M &\overset{\text{def}}{=} D^{-1}A.
\end{aligned}$$

I will call M the *walk matrix* of the weighted graph.

We will want to restrict our consideration to the case when A is positive semi-definite. To guarantee that we are in this situation, we will require that A be diagonally dominant. So, $m_{i,i} \geq 1/2$ for all i. Under this condition, it is known that for every initial probability distribution p, the random walk starting with probability distribution p will eventually approach the steady-state distribution,

$$q(u) \overset{\text{def}}{=} d_u / \sum_v d_v.$$

This is because q is the eigenvector of the largest eigenvalue of M, and it has eigenvalue 1.

To see why some self-loops are required if we want the walk to converge, consider the graph composed of two vertices connected by one edge. If we start at one vertex, then after the next step we will be at the other vertex with probability 1. At each step, the walk will alternate between vertices, and never converge to any one distribution. Putting the self-loops in prevents this from happening.

Remark 7.1. *Rather than thinking of all this in terms of random walks, one can think in terms of diffusion of probability mass. We start with some initial distribution of mass. At each step, each node sends half of its mass to its neighbors, and keeps half for itself. When the walk converges, each node will have an amount of mass proportional to its degree.*

There is a strong relationship between $\Phi(G)$ and the rate at which random walks converge. The easy direction comes from letting S be a set such that $\Phi(S) = \Phi(G)$ and $\operatorname{vol}(S) \leq \operatorname{vol}(V)/2$. Then, consider the initial distribution

$$p_0(u) = \begin{cases} d_u / \sum_{w \in S} d_w & \text{if } u \in S \\ 0 & \text{otherwise.} \end{cases}$$

The probability that the walk will hit a vertex outside S in one step is

$$\sum_{u \in S, v \notin S} p_1(u) m_{u,v} = \frac{\sum_{u \in S, v \notin S} a_{u,v}}{\sum_{u \in S} d_u} = \Phi(S).$$

One can show that in each successive step, even less probability mass will escape. So, we must wait at least $1/4\Phi(S)$ steps before even a quarter of the probability mass escapes to $\bar{S}$, which should have at least half the probability mass under q.

Lovasz and Simonovits prove a partial converse to this observation. That is, if $\Phi(G)$ is big, then every random walk must converge quickly. Moreover, they show that if the random walk fails to converge quickly, then by examining the probability distributions that arise one can find a cut of small conductance. I will devote most of this hour to proving the theorem of Lovasz and Simonovits. If there is time remaining, I will indicate how we extend their analysis to obtain a nearly-linear time algorithm.

Theorem 7.2 (Lovasz-Simonovits). *Let A be a non-negative, diagonally-dominant matrix. Let M be the matrix realizing the corresponding random walk. Let p_0 be any initial probability distribution on vertices, and let*

$$p_t = p_0 M^t.$$

For each t, let π_t be a permutation such that

$$\frac{p_t(\pi_t(1))}{d_{\pi(1)}} \geq \frac{p_t(\pi_t(2))}{d_{\pi(2)}} \geq \cdots \frac{p_t(\pi_t(n))}{d_{\pi(n)}}.$$

For some $T \geq 1$, let

$$\phi \overset{\text{def}}{=} \min_{0 \leq t \leq T} \min_{1 \leq k < n} \Phi(\{\pi_t(1), \ldots, \pi_t(k)\}).$$

Then, for all $W \subseteq V$,

$$\left| \sum_{w \in W} p_T(w) - q(w) \right| \leq \left(\sqrt{x}, \sqrt{\sigma - x} \right) \left(1 - \frac{1}{8}\phi^2 \right)^t,$$

where $x = \sum_{w \in W} d_w$ and $\sigma = \sum_{v \in V} d_v$.

Before I prove this theorem, let me explain how we use it to find a cut in a graph. We will always start with a probability distribution p_0 that has all its weight concentrated on one vertex. If there is a cut S in the graph of conductance less than ϕ, and that vertex is chosen at random from S, then our preceding analysis shows that, after $1/4\phi$ steps, most of the probability mass will still be in S. So, ϕ will have to be small. As ϕ is the least conductance of the sets $\{\pi_t(1), \ldots, \pi_t(k)\}$, we can then find this set of low conductance by trying out each of these sets (in fact, this can be done in the time required to sort). As we wish to apply this algorithm in the case $\phi = 1/\log^{O(1)} n$, we can do this with some efficiency.

However, we will need more efficiency than this to make a nearly-linear time graph partitioning algorithm. In particular, we want to make sure that our algorithm does not run for much longer than the size of the set it cuts out. To do this, we study a *truncated diffusion*, in which we round all small values in p_t to zero. To choose the value of "small", we choose a size set that we would like to cut out, s, and then round every value less than $s/\log^{O(1)} n$ to zero. The much of the analysis in Section 3 of [ST04] is devoted to showing that this rounding does not significantly decrease the quality of the cut output by the algorithm. We prove [ST04, Lemma 3.1] that the algorithm outputs a set C such that $\text{vol}(C) \leq (5/6)\text{vol}(V)$, and that for each set S satisfying

$$\text{vol}(S) \leq (2/3)\text{vol}(V) \quad \text{and} \quad \Phi(S) \leq 1/\log^{O(1)} n,$$

there is a subset $S^g \subseteq S$ such that $\text{vol}(S^g) \geq \text{vol}(S)/2$ that can be decomposed into sets S^g_b for $b = 1, \ldots, \lg m$ such that if the algorithm is started from a vertex $v \in S^g_b$ and run with target size 2^b, then it will output a set of vertices C such that

$$(4/7)2^{b-1} \leq \text{vol}(C \cap S).$$

Now, to partition a graph, we run this algorithm starting from random nodes and with random target set sizes. We show that, after doing this enough, one can find a cut in a graph of approximately optimal balance, and conductance $1/\log^{O(1)} n$, provided that a cut of conductance $1/\log^{O(1)} n$ exists (with different constants in the big-O's).

7.1 The Lovasz-Simonovits Theorem

I will now give a proof of Theorem 7.2 for the simple case of regular, unweighted graphs, with the minimal necessary number of self-loops at each vertex. The proof for the more general case is almost exactly the same, but requires more care. In the case we consider, the matrix A only has entries 1 or 0 off the diagonal, and the each diagonal entry equals the sum of the off-diagonal entries in its row and column.

The genius of the analysis of Lovasz and Simonovits is that it is a potential function argument in which the value of the potential function is a concave curve. They prove that the curve from one time step lies strictly below the curve from the previous time step by a factor depending upon ϕ. The curve eventually approaches a straight line, and the closer it gets to the line the closer the walk is to having mixed.

Now, let me tell you how they construct the curve. At time t, they let π_t be a permutation such that

$$p_t(\pi_t(1)) \geq p_t(\pi_t(2)) \geq \cdots p_t(\pi_t(n)).$$

(Recall that we are only considering regular graphs, so the denominators go away) The curve at time t will be described by a function

$$I_t : [0, n] \mapsto [0, 1].$$

For all integers k between 0 and n, we set $I_t(k)$ to be the sum of the k highest probabilities,

$$I_t(k) = \sum_{i=1}^{k} p_t(\pi_t(i)),$$

and require that I_t be piece-wise linear in between. We remark

- The curve contains the points $I_t(0) = 0$ and $I_t(n) = 1$.
- As $p_t(i+1) \leq p_t(i)$ for all i, the curve I_t is concave.
- For every non-negative vector c such that for all i, $c(i) \leq 1$, we have that

$$\sum_i c(i) p_t(\pi_t(i)) \leq I_t\left(\sum_i c(i)\right). \tag{5}$$

  This follows immediately from the fact that $p_t(\pi_t(i))$ is monotonically increasing in i.

As the walk converges, the curve approaches the line from $(0, 0)$ to $(n, 1)$. We will now show that the curve for each time step of a walk lies under the curve for the previous step.

Theorem 7.3. *For every initial distribution p_0, all t, and every $k \in [0, n]$,*

$$I_t(k) \leq I_{t-1}(k).$$

Proof. We first observe that it suffices to prove the theorem for integral k. Let S be the set $\{\pi_t(1), \ldots, \pi_t(k)\}$, and let R be the set $\{\pi_{t-1}(1), \ldots, \pi_{t-1}(k)\}$. If all the edges connected to vertices in R had endpoints in S, then we would have

$$I_t(k) = \sum_{i=1}^{k} p_t(\pi_t(i)) = \sum_{i=1}^{k} p_{t-1}(\pi_{t-1}(i)) = I_{t-1}(k).$$

If this is not the case, we will see that $I_t(k) < I_t(k-1)$.

For each vertex $\pi_{t-1}(i)$, let α_i denote the fraction of edges going from this vertex to S. Note that $\sum_i \alpha_i = k$. We then have

$$I_t(k) = \sum_i \alpha_i p_{t-1}(\pi_{t-1}(i)) \leq I_{t-1}(k),$$

by (5). □

That was easy, so we will push it a little further: we will prove that the curve I_t has to lie below I_{t-1} by an amount depending on $\Phi(G)$. Our proof will make use of an isoperimetric property provided to G by the self-loops.

Lemma 7.4. *Let G be an unweighted graph with $2d$ edges at each vertex, d of which are self-loops, for some d. Let S be a set of vertices such that $|S| \leq n/2$. Then, for every set of vertices R of the same size as S, at most a $(1 - \Phi(S))$ fraction of the edges attached to S have endpoints in R.*

Proof. There are $2d\,|S|$ edges attached to S. Let S_0 be the subset of vertices in S that are not in R, and let R_0 be the subset of vertices in R that are not in S. Note that $|S_0| = |R_0|$.

We know that there are at least $2d\,|S|\,\Phi(S)$ edges attached to S that do not have their other endpoint in S. At most $d\,|R_0|$ of these can have their other endpoint in R_0. So, at least

$$2d\,|S|\,\Phi(S) - d\,|R_0|$$

of these edges have endpoints that are not in R_0. On the other hand, the self-loops attached to vertices in S_0 do not land in R either, providing $d\,|S_0|$ more such edges. Thus, there are at least

$$2d\,|S|\,\Phi(S) - d\,|R_0| + d\,|S_0| = 2d\,|S|\,\Phi(S)$$

edges with an endpoint in S whose other endpoint is not in R. □

Theorem 7.5. *For every initial distribution p_0, all $t \leq T$, and every $k \in [0, n/2]$, and every $t \leq T$*

$$I_t(k) \leq \frac{1}{2}\left(I_{t-1}(k - \phi k)) + I_{t-1}(k + \phi k))\right)$$

and for $k \in [n/2, n]$,

$$I_t(k) \leq \frac{1}{2}\left(I_{t-1}(k - \phi(n-k))) + I_{t-1}(k + \phi(n-k)))\right),$$

where ϕ is as defined in Theorem 7.2.

Proof of Theorem 7.5. We will only consider the case $k \in [0, n/2]$, and again observe that it suffices to prove the theorem in the case where k is an integer.

Let $S = \{\pi_t(1), \ldots, \pi_t(k)\}$, and let $R = \{\pi_{t-1}(1), \ldots, \pi_{t-1}(k)\}$. We will now view each edge in the graph as a pair of directed edges in opposite directions, except for the self-loops which we view as directed self-loops in which the direction is irrelevant. For any such directed edge, e, define $p_{t-1}(e)$ to equal $p_{t-1}(u)/2d$, where the edge e goes from u.

Let E_0 be the set of directed edges from R to S, let E_{in} be the set of directed edges into S, not from R, and let E_{out} be the set of directed edges from R that do not go to S. We have

$$p_t(S) = \sum_{e \in E_0} p_{t-1}(e) + \sum_{e \in E_{in}} p_{t-1}(e).$$

By counting, we find that $|E_{in}| = |E_{out}|$. Moreover,

$$p_{t-1}(e_{in}) \leq p_{t-1}(e_{out}),$$

for all $e_{in} \in E_{in}$ and $e_{out} \in E_{out}$. So,

$$p_{t-1}(E_{in}) \leq \frac{p_{t-1}(E_{in}) + p_{t-1}(E_{out})}{2},$$

and

$$p_t(S) \leq \frac{p_{t-1}(E_0)}{2} + \frac{p_{t-1}(E_0) + p_{t-1}(E_{in}) + p_{t-1}(E_{out})}{2}.$$

Setting $y = |E_{in}|$ and applying (5) as we did in the proof of Theorem 7.3, we may conclude

$$p_{t-1}(E_0) \leq I_{t-1}(|E_0|) = I_{t-1}(k-y), \text{ and}$$
$$p_{t-1}(E_0 \cup E_{in} \cup E_{out}) \leq I_{t-1}(|E_0 \cup E_{in} \cup E_{out}|) = I_{t-1}(k+y).$$

By Lemma 7.4, at least a ϕ fraction of the edges entering S do not come from R, so we may conclude $y \geq \phi k$. Thus, as the curve is concave, we find that

$$I_t(k) \leq \frac{1}{2}\left(I_t(k-y) + I_t(k+y)\right) \leq \frac{1}{2}\left(I_t(k-\phi k) + I_t(k+\phi k)\right).$$

□

Theorem 7.5 tells us that we can draw chords below the curve I_{t-1}, below which I_t must lie. Now, I claim that if I lies beneath J, then the curve underneath all the chords drawn on I will lie underneath the curve drawn underneath all the chords on J, and that both of these curves are concave.

Claim 7.6. *For a function f defined on $[0, n]$, let the operator* chord *map the function f to the function*

$$\text{chord}(f) : x \mapsto \begin{cases} \frac{1}{2}\left(f(x-\phi x) + f(x+\phi x)\right), & \text{if } x \leq n/2, \text{ and} \\ \frac{1}{2}\left(f(x-\phi(n-x)) + f(x+\phi(n-x))\right), & \text{if } x \geq n/2. \end{cases}$$

Then, for all concave functions I and J defined on $[0, n]$ such that I lies beneath J, chord(I) *lies beneath* chord(J)*, and both* chord(I) *are* chord(J) *concave.*

We now prove

Theorem 7.7. *For every initial probability distribution, p_0, every $k \in [0, n]$ and every time $t \leq T$,*

$$I_t(k) \leq \min\left(\sqrt{k}, \sqrt{n-k}\right)\left(1 - \frac{1}{8}\phi^2\right)^t + k/n.$$

In particular, for every set of vertices W,

$$\left|\sum_{w \in W} p_t(w) - |W|/n\right| \leq \left(\sqrt{|W|}, \sqrt{n - |W|}\right)\left(1 - \frac{1}{8}\phi^2\right)^t.$$

Proof. Consider the curve

$$R_0(k) = \min\left(\sqrt{k}, \sqrt{n-k}\right) + k/n.$$

It is easy to show that

$$I_0(k) \leq R_0(k), \text{for all } k \in [0, n].$$

While we can not necessarily reason about what happens to the curves I_t when we draw the chords indicated by Theorem 7.5, we can reason about the chords under R_0. If we set

$$R_t(k) = \frac{1}{2}\left(R_{t-1}(k - \phi k) + R_{t-1}(k + \phi k)\right),$$

for $k \in [0, n/2]$, and

$$R_t(k) = \frac{1}{2}\left(R_{t-1}(k - \phi(n-k)) + R_{t-1}(k + \phi(n-k))\right),$$

for $k \in [n/2, n]$, then an elementary calculation reveals that

$$R_t(k) \leq \min\left(\sqrt{k}, \sqrt{n-k}\right)\left(1 - \frac{1}{8}\phi^2\right)^t + k/n.$$

By Claim 7.6, we have

$$I_t(k) \leq R_t(k),$$

which proves the theorem. □

References

[AHK04] Arora, Hazan, and Kale. 0(sqrt log n) approximation to SPARSEST CUT in O (n^2) time. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, February 1995.

[ARV04] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing (STOC-04)*, pages 222–231, New York, June 13–15 2004. ACM Press.

[BCHT] Erik Boman, Doron Chen, Bruce Hendrickson, and Sivan Toledo. Maximum-weight-basis preconditioners. *to appear in Numerical Linear Algebra and Applications.*

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

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

[BH01] Erik Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $O(n^2)$ time. In *Proceedings of The Twenty-Eighth Annual ACM Symposium On The Theory Of Computing (STOC '96)*, pages 47–55, New York, USA, May 1996. ACM Press.

[BT97] Bau, D. and Trefethen, L. N. *Numerical Linear Algebra.* SIAM Publications, 1997.

[Che70] J. Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. In R. C. Gunning, editor, *Problems in Analysis*, pages 195–199. Princeton University Press, 1970.

[Chu97] F. R. K. Chung. Spectral graph theory. *Regional Conference Series in Mathematics, American Mathematical Society*, 92:1–212, 1997.

[CW82] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*, 11(3):472–492, August 1982.

[EBV] B. Hendrickson E. Boman and S. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. available at `http://arxiv.org/abs/cs.NA/0407022`.

[EEST05] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, 2005.

[FK81] Z. Füredi and J. Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1(3):233–241, 1981.

[GMZ95] Keith Gremban, Gary Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *9th IPPS*, pages 65–69, 1995.

[Gre96] Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD thesis, Carnegie Mellon University, CMU-CS-96-123, 1996.

[HL94] B. Hendrickson and R. Leland. The chaco user's guide, version 2.0. Tech. Rep. SAND94-2692, Sandia National Labs, Albuquerque, NM, October 1994.

[Jos97] Anil Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, UIUC, 1997.

[KK98] George Karypis and Vipin Kumar. *MeTis: Unstrctured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*, September 1998.

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[LR99] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.

[LRT79] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, April 1979.

[LS93] Lovasz and Simonovits. Random walks in a convex body and an improved volume algorithm. *RSA: Random Structures & Algorithms*, 4:359–412, 1993.

[Mar88] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, July 1988.

[MMP$^+$05] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures*, 2005.

[Rei98] John Reif. Efficient approximate solution of sparse linear systems. *Computers and Mathematics with Applications*, 36(9):37–58, 1998.

[SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, July 1989.

[ST03] Daniel Spielman and Shang-Hua Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $o(m^{1.31})$. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 416–427, 2003. Most recent version available at `http://arxiv.org/cs.DS/0310036`.

[ST04] Spielman and Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems [extended abstract]. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2004. full version available at `http://arxiv.org/abs/cs.DS/0310051`.

[Vai90] Pravin 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.

[Vu05] Van Vu. Spectral norm of random matrices. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, 2005.