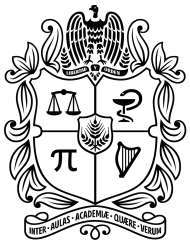

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Continuous methods for combinatorial optimization in graphs

Santiago Cano Vásquez

2025

Contents

1 Introduction

In this monograph, we present an introduction to the use of linear and convex optimization methods to combinatorial problems in graphs, which can serve as a starting point for understanding some of the recent advancements in the area of graph theory, we begin with some background in linear algebra and spectral graph theory, then we introduce laplacian systems, that is, systems of the form $Lx = b$ with L the laplacian matrix of a graph, we will develop fast methods for solving them using conjugate gradient with an efficient preconditioner, achieving an almost linear running time, in particular we focus on low stretch spanning trees and approximate cholesky factorization including some of the recent research by Gao, Kyng and Spielman [GKS23], we then show how having a fast laplacian solver can be used to make fast algorithms for the maximum flow problem, by using it as a subroutine in the multiplicative weights algorithm, finally we introduce the interior point method applied to the minimum cost flow problem, and show how we can make the algorithm faster by calculating each step by solving a laplacian system as shown by Daitch and Spielman [DS08], thus achieving an $\tilde{O}(m^{3/2})$ time.

2 Some linear algebra

For a complex number $x = a + ib$, its conjugate is $\overline{x} = a - ib$, then $x = \overline{x}$ if and only if $x \in \mathbb{R}$. Let $M \in \mathbb{C}^{m \times n}$ be a matrix, and let its conjugate transpose, denoted M^*, be the matrix such that $M(i,j) = \overline{M(j,i)}$, we say that M is hermitian if $M = M^*$, note that real symmetric matrices are hermitian. Define the inner product between two vectors x and y in $\mathbb{C}^n$ as

$$\langle x, y \rangle = x^* y$$

so $\langle x, x \rangle = ||x||^2$, and for any matrix M

$$\langle Mx, y \rangle = \langle x, M^* y \rangle$$

for a matrix M, a vector v and a scalar λ, if

$$Mv = \lambda v \tag{1}$$

we say that v is an eigenvalue corresponding to the eigenvalue λ of M, (1) implies $(M - I\lambda)v = 0$, so we can find λ by solving $\det(M - I\lambda) = 0$, note that $\det(M - I\lambda)$ is a degree n polynomial, we will refer to this polynomial as the *characteristic polynomial* of M, by the fundamental theorem of algebra it has n solutions, we will call the number of times λ appears as a solution to the characteristic polynomial the algebraic multiplicity of λ. Let λ be an eigenvalue of M, with associated eigenvectors $v_1, v_2, ..., v_k$, the subspace $span\{v_1, v_2, ..., v_n\}$ is the *eigen-space* of λ, and the dimension of this subspace is called the geometric multiplicity of λ.

Since we will be working mainly with matrices that are real and symmetric, we prove a few useful properties.

Lemma 1. *If M is a real symmetric matrix, then all of its eigenvalues are real*

Proof. Let v be an eigenvalue of M, then

$$\overline{\lambda}\langle v, v\rangle = \langle Av, v\rangle = \langle v, A^*v\rangle = \langle v, Av\rangle = \lambda\langle v, v\rangle$$

then $\lambda = \overline{\lambda}$ is a real number. □

Lemma 2. *If u and v are eigenvectors of a real symmetric matrix M corresponding to different eigenvalues λ_1 and λ_2, then u is orthogonal to v.*

Proof.

$$\lambda_1\langle u, v\rangle = \langle Au, v\rangle = \langle u, Av\rangle = \lambda_2\langle u, v\rangle$$

since $\lambda_1 \neq \lambda_2$, then $\langle u, v\rangle = 0$ □

Note that this lemma does not say anything about eigenvectors with the same eigenvalue, if for every eigenvalue the algebraic multiplicity equals the geometric multiplicity, then the subspace spanned by the eigenvectors of M will make a basis for R^n and we could use Gram-Schmidt on every eigen-space to obtain an orthonormal basis of eigenvectors, we will see that real symmetric matrices do indeed have an orthonormal basis of eigenvectors, this result is known as the *spectral theorem*, before we give a proof we first need one more result.

Definition 1. *A matrix M is diagonalizable if there exists an invertible matrix P and a diagonal matrix D, such that*

$$M = PDP^{-1}$$

This definition leads to the next theorem about the existence of these matrices D and P.

Theorem 1. *[Diagonalization theorem] Let M be a matrix, $v_1, ..., v_n$ be its set of eigenvectors and $\lambda_1, ..., \lambda_n$ its associated set of eigenvalues, then $M = PDP^{-1}$ if and only if M's eigenvectors make a basis, $P = [v_1, ..., v_n]$ and $D(i, i) = \lambda_i$.*

Proof. Suppose M is diagonalizable, then there exists an invertible matrix P and a diagonal matrix D such that $M = PDP^{-1}$, this implies $MP = PD$, since D is diagonal, then $PD = [D(1, 1)v_1, ..., D(n, n)v_n]$, this means that the columns of P are eigenvectors of M and the diagonal elements of D are its eigenvalues. Since P is invertible, then its columns are linearly independent, therefore M's eigenvalues make a basis.

For the other direction suppose we have an invertible P and a diagonal matrix D as defined in the theorem, then by definition of eigenvector and eigenvalue $MP = PD$, and since P is invertible, then $M = PDP^{-1}$, so M is diagonalizable. □

This theorem implies that, we would prove the spectral theorem if we prove that every real symmetric matrix is diagonalizable, by lemma 2, we could use Gram-Schmidt and obtain an orthonormal basis of eigenvectors, instead, we give a more direct proof, we will show that any real symmetric matrix is orthogonally diagonalizable, implying an orthonormal eigen-basis.

Theorem 2 (the spectral theorem for real symmetric matrices)**.** *For any real symmetric matrix $M \in \mathbb{R}^{n\times n}$, there exists a matrix P whose columns make an orthonormal basis of eigenvalues for M, and a matrix D with the elements in the diagonal the eigenvalues of M, such that $M = PDP^{-1}$.*

Proof. We use induction on n, the base case $n = 1$ is trivial. Suppose it is true for $n-1$, consider a symmetric matrix M and choose one eigenvalue λ_1 and associated eigenvector v_1, we can normalize it so that $||v_1|| = 1$, by Gram-Schmidt we can find an orthonormal basis $v_1, ..., v_n$, we define $P = [v_1, ..., v_n]$ so that P is orthonormal, then we have $P^{-1} = P^T$, let $A = P^{-1}MP$, notice that

$$A^T = (P^TMP)^T = P^T(P^TM)^T = P^TMP = A$$

so A is symmetric, also

$$P^TMPe_1 = P^TMv_1 = \lambda_1 P^Tv_1 = \lambda_1 e_1$$

so A is of the form

$$\begin{vmatrix} \lambda_1 & 0 \\ 0 & S \end{vmatrix}$$

with S a symmetric matrix, then by induction hypothesis there exists orthogonal Q and diagonal C such that $C = Q^TSQ$, let $R = P\begin{vmatrix} 1 & 0 \\ 0 & Q \end{vmatrix}$, note that

$$R^{-1} = \begin{vmatrix} 1 & 0 \\ 0 & Q \end{vmatrix}^{-1} P^{-1} = \begin{vmatrix} 1 & 0 \\ 0 & Q^{-1} \end{vmatrix} P^{-1} = \begin{vmatrix} 1 & 0 \\ 0 & Q^T \end{vmatrix} P^T = R^T$$

so R is orthogonal, finally

$$R^TMR = \begin{vmatrix} 1 & 0 \\ 0 & Q^T \end{vmatrix} P^TMP \begin{vmatrix} 1 & 0 \\ 0 & Q \end{vmatrix} = \begin{vmatrix} 1 & 0 \\ 0 & Q^T \end{vmatrix} \begin{vmatrix} \lambda_1 & 0 \\ 0 & S \end{vmatrix} \begin{vmatrix} 1 & 0 \\ 0 & Q \end{vmatrix} = \begin{vmatrix} \lambda_1 & 0 \\ 0 & C \end{vmatrix} = D$$

so we have found orthogonal R and diagonal D such that $R^TMR = D$

□

Suppose $M = PDP^{-1}$ as given, then we can write M as

$$M = \sum_i \lambda_i v_i v_i^T$$

we call this representation of M, the *spectral decomposition* of M.

Definition 2. *The trace of a matrix $M \in \mathbb{R}^{n\times n}$ denoted by $Tr(M)$ is defined as the sum of its diagonal elements, that is*

$$Tr(M) = \sum_{i=1}^{n} M(i,i)$$

Let $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times n}$ be two matrices, note that AB and BA is well defined, then $AB(i,j) = \sum_{i=1}^{m} A(i,j)B(j,i)$, this means

$$\begin{aligned} Tr(AB) =& \sum_{i=1}^{n}\sum_{i=1}^{m} A(i,j)B(j,i) \\ =& \sum_{i=1}^{m}\sum_{i=1}^{n} B(j,i)A(i,j) \\ =& Tr(BA) \end{aligned}$$

this implies the cyclic property of the trace, this is, for matrices A,B and C

$$Tr(ABC) = Tr(BCA) = Tr(CAB) \tag{2}$$

One alternative definition of the trace of a matrix A, is that of the sum of the eigenvalues of A, even though the two definitions are equal for any matrix, we will give a nice proof for diagonalizable matrices, by using the properties we just derived

Lemma 3. *For a diagonalizable matrix M*

$$Tr(M) = \sum_{i=1}^{n} \lambda_i$$

Proof. Using the fact that M is diagonalizable and (2)

$$Tr(M) = Tr(PDP^{-1}) = Tr(DPP^{-1}) = Tr(D) = \sum_{i=1}^{n} \lambda_i$$

□

2.1 Variational characterization of eigenvalues

We now present a different view of eigenvalues, we know that eigenvalues are solutions to a polynomial $\det(M-\lambda I)$, we now show how eigenvalues of a matrix can be viewed as solutions to optimization problems. We define the *Rayleigh quotient* of a vector x with respect to a matrix M as

$$\frac{x^T M x}{x^T x}$$

if v is an eigenvector associated with eigenvalue λ then

$$\frac{v^T M v}{v^T v} = \frac{\lambda v^T v}{v^T v} = \lambda$$

Lemma 4. *Let $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ be the eigenvalues of a real symmetric matrix M then*

$$\lambda_1 = \max_x \frac{x^T M x}{x^T x}$$

similarly

$$\lambda_n = \min \frac{x^T M x}{x^T x}$$

Proof. We only give a proof for the first claim, let $v_1, v_2, ..., v_n$ be the associated eigenvectors, since they make an orthonormal basis, we can write x as $x = \sum_i c_i v_i$ then

$$x^T A x = \sum_{i=1}^{n} c_i v_i^T M \sum_{j=1}^{n} c_i v_i = \sum_{i=1}^{n} c_i^2 \lambda_i$$

$$x^T x = \sum_{i=1}^{n} c_i v_i^T \sum_{j=1}^{n} c_i v_i = \sum_{i=1}^{n} c_i^2$$

this means

$$\frac{x^T M x}{x^T x} = \frac{\sum_{i=1}^{n} c_i^2 \lambda_i}{\sum_{i=1}^{n} c_i^2} \leq \frac{\lambda_1 \sum_{i=1}^{n} c_i^2}{\sum_{i=1}^{n} c_i^2} = \lambda_1$$

□

For intuition as to why this is true, we can use the geometric definition of the dot product to get

$$\lambda_1 = \max_x \frac{||x|| \cdot ||Mx|| \cos\theta}{||x||^2}$$

by making $x = v_1$, that is, x is the eigenvector associated to the λ_1 eigenvalue, we have

$$\lambda_1 = \frac{||Mv_1||}{||v_1||} \qquad \text{or} \qquad \lambda_1 = -\frac{||Mv_1||}{||v_1||}$$

this is exactly the definition of eigenvalue, how much is an eigenvector rescaled by the linear transformation of M, this implies that maximizing the Rayleigh quotient is equal to finding the vector with the biggest rescalling under the transformation of M.

Note that this results gives a characterization for the smallest and biggest eigenvalues only, the following lemma shows how we can find any eigenvalue.

Lemma 5. *Given a real symmetric matrix A with eigenvalues $\lambda_1, ..., \lambda_n$ and associated eigenvectors $v_1, ..., v_n$, let T_k be the subspace orthogonal to $v_1, ..., v_k - 1$ then*

$$\lambda_k = \max_{x \in T_k} \frac{x^T M x}{x^T x}$$

Proof. We begin by noticing that $\langle x, v_i \rangle = c_i$, then since $x \in T_k$ we have $c_1 = c_2 = ... = c_{k-1} = 0$, we obtain

$$\frac{x^T M x}{x^T x} = \frac{\sum_{i=k}^{n} c_i^2 \lambda_i}{\sum_{i=k}^{n} c_i^2} \leq \frac{\lambda_k \sum_{i=k}^{n} c_i^2}{\sum_{i=k}^{n} c_i^2} = \lambda_k$$

Since $v_k \in T_k$ and $\frac{v_k^T M v_k}{v_k^T v_k} = \lambda_k$ the max is reached, which proves the lemma.

□

Although the previous lemma gives a way to find any eigenvalue, we need all $\lambda_1, ..., \lambda_{k-1}$ to find λ_k, we now show how we can find λ_k without the need to know any other eigenvalue.

Theorem 3. *(Courant-Fischer Theorem) For a real symmetric matrix M with eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$*

$$\lambda_k = \max_{\substack{S \subseteq \mathbb{R}^n \\ dim(S)=k}} \min_{\substack{x \in S \\ x \neq 0}} \frac{x^T M x}{x^T x}$$

and

$$\lambda_k = \min_{\substack{T \subseteq \mathbb{R}^n \\ dim(T)=n-k+1}} \max_{\substack{x \in T \\ x \neq 0}} \frac{x^T M x}{x^T x}$$

Proof. Let $S_k = span\{v_1, ..., v_k\}$ for $x \in S_k$

$$\frac{x^T M x}{x^T x} = \frac{\sum_{i=1}^{n} c_i^2 \lambda_i}{\sum_{i=1}^{n} c_i^2} \geq \lambda_k$$

this means

$$\max_{\substack{S \subseteq \mathbb{R}^n \\ \dim(S)=k}} \min_{\substack{x \in S \\ x \neq 0}} \frac{x^T M x}{x^T x} \geq \min_{\substack{x \in S_k \\ x \neq 0}} \frac{x^T M x}{x^T x} \geq \lambda_k$$

Let $T = span\{v_k, ..., v_n\}$, for any $x \in T_k$

$$\frac{x^T M x}{x^T x} = \frac{\sum_{i=1}^{n} c_i^2 \lambda_i}{\sum_{i=1}^{n} c_i^2} \leq \lambda_k$$

as it has dimension $n - k + 1$ it intersects any k dimensional space, then

$$\max_{\substack{S \subseteq \mathbb{R}^n \\ \dim(S)=k}} \min_{\substack{x \in S \\ x \neq 0}} \frac{x^T M x}{x^T x} \leq \max_{\substack{S \subseteq \mathbb{R}^n \\ \dim(S)=k}} \min_{\substack{x \in S \cap T_k \\ x \neq 0}} \frac{x^T M x}{x^T x} \leq \lambda_k$$

□

3 Graphs

Consider a graph (V, E, w) with V the set of vertices, E the set of edges and a weight function $w : E \to \mathbb{R}^+$ that associates a non-negative real number to every edge, we denote an edge as a pair (i, j) of vertices, one common way to represent graphs is with an *adjacency matrix*, for a graph G its adjacency matrix is given by

$$A_G(i,j) \stackrel{\text{def}}{=} \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

its *degree matrix* is given by

$$D_G(i,j) \stackrel{\text{def}}{=} \begin{cases} \sum_k w(i,k) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

another very useful matrix associated with a graph is the *laplacian matrix*, it is defined as

$$L_G = \sum_{(i,j)\in E} w(i,j)(e_i - e_j)(e_i - e_j)^T$$

note that this can be written as

$$L_G(i,j) = \begin{cases} \sum_k w(i,k) & \text{if} i = j \\ -w(i,j) & otherwise \end{cases}$$

which is the same as

$$L_G = D_G - A_G$$

it is also useful to note that for every vector x

$$(L_G x)(i) = \sum_{(i,j)\in E} w(i,j)(x(i) - x(j))$$

from this formula we can see that $L_G\mathbf{1} = 0$, so the vector $\mathbf{1}$ is an eigenvector associated with the eigenvalue 0, this implies that the Laplacian matrix is singular. The following theorem gives a connection between the connectedness of G and the kernel of its Laplacian matrix.

Theorem 4. *Consider $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ the spectrum of L_G, then $\lambda_2(L_G) > 0$ if and only if G is connected*

Proof. Suppose G is disconnected, it can be represented as the union of two graphs G_1 and G_2, then we can write its Laplacian as

$$L = \begin{vmatrix} L(G_1) & 0 \\ 0 & L(G_2) \end{vmatrix}$$

with $L(G_i)$ the Laplacian of each subgraph, then $v_1 = \begin{vmatrix} \mathbf{1}_{|G_1|} \\ \mathbf{0}_{|G_2|} \end{vmatrix}$ and $v_2 = \begin{vmatrix} \mathbf{0}_{|G_1|} \\ \mathbf{1}_{|G_1|} \end{vmatrix}$ are two orthogonal eigenvectors associated with the eigenvalue 0, so if G is

disconnected $\lambda_2 = 0$. For the other direction suppose G is connected consider a non-zero eigenvector of eigenvalue 0, then $L_G v = 0$ implies

$$vL_G v = \sum_{(i,j)\in E} (v(i) - v(j))^2 = 0$$

so $v(i) = v(j)$ for every adjacent vertices, since G is connected, there is a path between every pair of vertices, so $v = [c, c, ..., c]$ for a real number c, then the eigenspace of eigenvalue 0 has dimension 1.

□

This same argument can be used to prove by induction that the eigenvalue 0 appears n times, if and only if G has n connected components.

4 Electrical networks

An undirected graph $G = (V, E)$ can be viewed as an electrical network if for each edge $(i, j) \in E$ there exists an associated real number $r(i, j) > 0$, we call this number its resistance, its inverse $c(i, j) = 1/r(i, j)$ is called conductance.

For a given node i, let $b(i)$ be the current entering the network at i, so $b(i) > 0$ represents incoming current and $b(i) < 0$ represents outgoing current. Consider an arbitrary orientation of the edges $\vec{E}$, let B be the *edge-vertex incidence matrix* associated with $\vec{E}$, whose rows are indexed by the edges in $\vec{E}$ and the columns by the vertices

$$B((i,j),k) = \begin{cases} 1 & \text{if } k = i \\ -1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

notice that the row associated with (i, j) is $(e_i - e_j)^T$.

The flow through the network must satisfy *Ohm's law* and *Kirchoff's current law*. *Ohm's law* states that there are associated potentials p such that

$$f(i,j) = \frac{p(i) - p(j)}{r(i,j)} = (p(i) - p(j))c(i,j)$$

Note that $f(j, i) = -f(i, j)$, so that $-f(i, j)$ represents positive current in the opposite direction. Let C be a matrix whose rows are indexed by the edges and the values on the diagonal correspond to the conductance of the edge, then we can express Ohm's law as

$$f = CBp$$

Kirchoff's current law states that the current flowing into a node i equals the current flowing out of it, that is

$$\sum_{h:(h,i)\in\vec{E}} f(h,i) - \sum_{j:(i,j)\in\vec{E}} f(i,j) + b(i) = 0$$

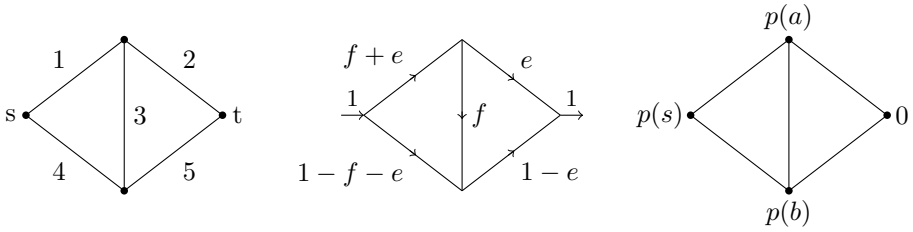

Figure 1: Resistances, flows and potentials

that is

$$\sum_{j:(i,j)\in E} f(i,j) = b(i)$$

Since the ith row of B^T has a 1 for every edge leaving i and -1 for every edge entering i, this means

$$B^T f = b$$

If we have a graph as in figure 1, with resistances as at the left of the figure, suppose we have 1 unit of current entering at node s and leaving at node t, using Kirchoff's current law we obtain the flow as in the center, and the potentials are given as the in the right of the figure, using Ohm's law we obtain the equations $p(a) = 2e$, $p(b) = 5 - 5e$, $p(s) = f + 3e$, the two remaining edges give $p(a) - p(b) = 3f$ and $p(s) - p(b) = 4 - 4f - 4e$, combining them we obtain the system

$$5f = 9 - 12e, \quad 3f = 7e - 5$$

this gives $e = 52/71$, $f = 3/71$ from which we can obtain the potentials $p(s) = 159/71$, $p(a) = 104/71$ and $p(b) = 95/71$, the solved system is shown in figure 2.

We now show a more clever way to find these potentials, notice that from Ohm's and Kirchoff's laws we obtain

$$\begin{aligned} b(i) &= \sum_{j:(i,j)\in E} f(i,j) \\ &= \sum_{j:(i,j)\in E} (p(i) - p(j))c(i,j) \\ &= L_G p(i) \end{aligned}$$

in matrix form

$$B^T C B p = L_G p = b$$

this means that the vector of potentials p will be given by

$$p = L_G^{-1} b$$

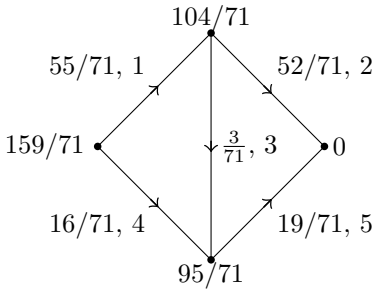

Figure 2: Electrical flow from the graph in figure 1, vertices have their potentials, edges have their current followed by their resistance.

however we know that the weighted Laplacian L_G is non invertible, furthermore the system $L_G p = b$ only has a solution if $b \in \text{im}(L_G)$, suppose the graph G is connected, then its image will be $\text{im}(L_G) = \{x | x^T \mathbf{1} = 0\}$, that is, the subspace orthogonal to the vector $\mathbf{1}$, thus we will only consider systems in which $b^T \mathbf{1} = 0$, intuitively this means that the amount of current entering the systems is equal to the amount of current leaving it.

Definition 3. *For a given symmetric matrix L, we define its Moore-Penrose pseudo-inverse L^+ as the matrix that has the same image as L and satisfies*

$$LL^+ = \Pi$$

with Π the symmetric projection onto the image of L, that is, Π is symmetric and $\Pi^2 = \Pi$.

This gives the following corollary

Corollary 1. *The eigenvalues of Π are 0 or 1.*

Proof. Suppose v is an eigenvalue of Π, then

$$\lambda v = \Pi v = \Pi^2 v = \lambda^2 v \Rightarrow \lambda = \lambda^2$$

this means $\lambda \in \{0, 1\}$ □

Suppose we have a vector x in the image of L, then $\Pi x = x$, this is, L^+ acts as an inverse in the image of L. Suppose that $0 = \lambda_1 < ... \leq \lambda_n$ and $v_1, ..., v_n$ are the eigenvectors and eigenvectors, respectively, of L then

$$L^+ = \sum_{i>1} (1/\lambda_i) v_i v_i^T$$

Proof. Since each $v_i v_i^T$ is symmetric, then L^+ is symmetric, by spectral decomposition $L = \sum_i \lambda_i v_i v_i^T$, then

$$LL^+ = \sum_{i>1} v_i v_i^T$$

and

$$(LL^+)^2 = \sum_{i>1} v_i v_i^T \sum_{j>1} v_i v_i^T = \sum_{i>1} v_i v_i^T = LL^+$$

finally, notice that $\mathrm{im}(L^+)$ is the $n-1$ dimensional space orthogonal to $\mathbf{1}$, so $im(L^+) = \mathrm{im}(L)$, we conclude that LL^+ is the symmetric projection matrix to the image of L. □

Its easy to verify that the pseudo-inverse satisfies

$$L^+ L = \Pi$$

$$LL^+ L = L$$

$$L^+ LL^+ = L^+$$

When b belongs in the image of L_G, we can find p with $p = L_G^+ b$ if we consider a flow of 1 unit of current entering at s and leaving at t then

$$p = L_G^+ b = L_G^+ (e_s - e_t)$$

the potential difference between two vertices i and j is given by

$$p(i) - p(j) = (e_i - e_j)p = (e_i - e_j)L_G^+ (e_s - e_t)$$

by the symmetry of L_G^+, this equals

$$(e_s - e_t)L_G^+ (e_i - e_j)$$

this means that the potential difference between i and j when we flow one unit of current from s to t is the same as the potential difference between s and t when we flow one unit of current from i to j. The *effective resistance* between two nodes i and j is the difference in potentials between i and j if we view the entire network as a resistor, this is

$$R_{\text{eff}} \stackrel{\text{def}}{=} (e_i - e_j)L_G^+ (e_i - e_j)$$

The energy dissipated by a resistor of resistance r is given by $f^2 r$, we then define the energy of the network as

$$\mathcal{E}(f) \stackrel{\text{def}}{=} \sum_{(i,j)\in E} f^2(i,j) r(i,j)$$

since $f(i,j) = c(i,j)(p(i) - p(j))$, we can write the energy in terms of potentials as

$$\begin{aligned}
\mathcal{E}(f) &= \sum_{(i,j)\in E} c(i,j)(p(i) - p(j))^2 \\
&= \sum_{(i,j)\in E} c(i,j) p^T (e_i - e_j)(e_i - e_j)^T p \\
&= p^T \Big(\sum_{(i,j)\in E} c(i,j)(e_i - e_j)(e_i - e_j)^T \Big) p \\
&= p^T L_G p
\end{aligned}$$

then if p and f are from an $s-t$ electrical flow we have

$$\mathcal{E}(f) = p^T L_G p = p^T L_G L_G^+ L_G p = b^T L_G^+ b = R_{\text{eff}}(s,t) \tag{3}$$

so the energy dissipated by the network in a flow from s to t equals the potential drop between s and t, we now prove one useful property of electrical flows,

Lemma 6. *Consider any flow g of value 1 from s to t that satisfies $Ug = b$ with $b^T\mathbf{1} = 0$, then*

$$\mathcal{E}(g) \geq \mathcal{E}(f)$$

this is, the electrical flow f minimizes the energy among all flows satisfying flow conservation.

Proof. consider a flow $h = f - g$, then for any node i

$$\sum_{j:(i,j)\in E} h(i,j) = \sum_{j:(i,j)\in E} f(i,j) - \sum_{j:(i,j)\in E} g(i,j) = b(i) - b(i) = 0$$

the energy of g is given by

$$\begin{aligned}
\mathcal{E}(g) &= \sum_{(i,j)\in E} g^2(i,j) r(i,j) \\
&= \sum_{(i,j)\in E} (f(i,j) + h(i,j))^2 r(i,j) \\
&= \sum_{(i,j)\in E} f^2(i,j) r(i,j) + \sum_{(i,j)\in E} h^2(i,j) r(i,j) + 2 \sum_{(i,j)\in E} f(i,j) h(i,j) r(i,j)
\end{aligned}$$

the last term is

$$\begin{aligned}
\sum_{(i,j)\in E} f(i,j) h(i,j) r(i,j) &= \sum_{(i,j)\in E} (p(i) - p(j)) h(i,j) \\
&= \sum_{(i,j)\in E} p(i) h(i,j) + p(j) h(j,i) \\
&= \sum_{i\in V} p(i) \sum_{j:(i,j)\in E} h(i,j) \\
&= 0
\end{aligned}$$

then

$$\mathcal{E}(g) = \mathcal{E}(f) + \mathcal{E}(h) \geq \mathcal{E}(f)$$

□

5 Graph sparsification

Given a graph $G = (V, E)$ define its *edge capacity* as $u : E \to \mathbb{R}$, such that for an edge (i, j) any flow $f(i, j)$ over that edge satisfies

$$f(i,j) \leq u(i,j)$$

Consider a partition $(S, \bar{S})$ of V, we call S a cut on G, we define $\delta(S) = \{(i,j)|i \in S, j \notin S\}$ as the subset of edges in the cut, the capacity of the cut is given by

$$u(\delta(S)) = \sum_{(i,j)\in\delta(S)} u(i,j)$$

Given $\epsilon > 0$, we say that a graph $H = (V, E')$ with capacities $u'(i,j)$ is a cut sparsifier of G if $|u'(\delta(S)) - u(\delta(S))| \leq \epsilon u(\delta(S))$ and $|E'| = O(n \log n/\epsilon^2)$ for any cut S.

Definition 4. *A matrix M is positive definite, denoted as $M \succ 0$, if for all $x \in \mathbb{R}^n$ $x^T Mx > 0$, similarly, a matrix M is positive semidefinite, denoted $M \succeq 0$, if for all $x \in \mathbb{R}^n$ $x^T Mx \geq 0$*

Similarly for two matrices A and B, $A \succeq B$ if $x^T Ax \geq x^T Bx$ for all $x \in \mathbb{R}^n$, we now define the *ϵ-approximation* G' of G, as a graph such that

$$(1-\epsilon)L_G \preceq L_{G'} \preceq (1+\epsilon)L_G \tag{4}$$

and $|E'| = O(n \log n/\epsilon^2)$.

We proceed to show that an ϵ-approximation is a cut sparsifier, consider the vector x_S such that $x_S(i) = 1$ if $i \in S$ and 0 otherwise, then

$$x_S^T L_{G'} x_S = \sum_{(i,j)\in E} u'(i,j)(x_S(i) - x_S(j))^2 = u'(\delta(s))$$

this is because $(x_S(i) - x_S(j))$ is 1 only if $(i,j) \in \delta(S)$, then by (4) G' is a cut sparsifier.

We will construct this sparsifier by assigning a probability $p(i,j)$ to each edge and draw a random sample with replacement, the algorithm is given by

SPARSIFY(G)
1 $u' = 0$
2 $k = 8n\ln(n)/\epsilon^2$
3 Calculate $R_{\text{eff}}(i,j)$ for every (i,j)
4 **for** $i = 1$ **to** k
5 Sample edge with (i,j) probability $u(i,j)R_{\text{eff}}(i,j)/(n-1)$
6 $u'(i,j) = u'(i,j) + (n-1)/kR_{\text{eff}}(i,j)$
7 $E' = E' \cup (i,j)$
8 **return** (V, E')

To show that the algorithm works we begin by showing that probabilities given by $p_{i,j} = u(i,j)R_{\text{eff}}(i,j)/(n-1)$ make a probability distribution, since $R_{\text{eff}}(i,j)/(n-$

1) equals $\mathcal{E}(i,j)$ it is positive, now

$$\begin{aligned}\frac{1}{n-1}\sum_{(i,j)\in E} u(i,j)R_{\text{eff}}(i,j) =& \frac{1}{n-1}\sum_{(i,j)\in E} u(i,j)(e_i-e_j)^T L_G^+(e_i-e_j)\\ =& \frac{1}{n-1}\sum_{(i,j)\in E} Tr(u(i,j)L_G^+(e_i-e_j)(e_i-e_j)^T)\\ =& \frac{1}{n-1}Tr(L_G^+\sum_{(i,j)\in E} u(i,j)(e_i-e_j)(e_i-e_j)^T)\\ =& \frac{1}{n-1}Tr(L_G^+L_G)\\ =& \frac{1}{n-1}Tr(\Pi)\\ =& 1\end{aligned}$$

The last equality comes from (3) given that Π has eigenvalue 0 with multiplicity 1, and 1 with multiplicity $n-1$. To prove that the resulting graph is indeed an $\epsilon-$*approximation*, let Z_l be a random matrix associated with the change in each iteration of the algorithm, so that $Z_l = \frac{n-1}{kR_{\text{eff}}(i,j)}(e_i-e_j)(e_i-e_j)^T$ for a selected edge (i,j) with probability $u(i,j)R_{\text{eff}}(i,j)/(n-1)$, then

$$\begin{aligned}\mathbb{E}[Z_l] =& \sum_{(i,j)\in E}\frac{u(i,j)R_{\text{eff}}(i,j)}{n-1}\frac{n-1}{kR_{\text{eff}}(i,j)}(e_i-e_j)(e_i-e_j)^T\\ =& \frac{1}{k}\sum_{(i,j)\in E} u(i,j)(e_i-e_j)(e_i-e_j)^T\\ =& \frac{L_G}{k}\end{aligned}$$

as $L_{G'} = \sum_{i=1}^k Z_i$ we have

$$\mathbb{E}[L_{G'}] = \sum_{i=1}^k \frac{L_G}{k} = L_G$$

So that in expectation $L_{G'}$ equals L_G, to prove that the resulting graph is close to its expectation we need the following concentration inequality, we state it without a proof.

Theorem 5. *Let Z be a random, symmetric, positive semi-definite $n\times n$ matrix. Let $X = \mathbb{E}[Z]$, suppose $Z \preceq RX$ for some scalar $R \geq 1$. Let $Z_1, ..., Z_k$ be independent copies of Z. For any $\epsilon \in (0,1)$, we have*

$$Pr\left[(1-\epsilon)X \preceq \frac{1}{k}\sum_{i=1}^k Z_i \succeq (1+\epsilon)X\right] \geq 1 - 2n\exp\left(-\epsilon^2 k/4R\right)$$

We also need

Lemma 7.

$$Z \preceq (n-1)\mathbb{E}[Z]$$

Proof. We want to prove that

$$\frac{n-1}{k} L_G - Z \succeq 0$$

as matrix Z is given by $Z = \frac{n-1}{kR_{\text{eff}}(i,j)}(e_i - e_j)(e_i - e_j)^T$, this is equal to

$$L_G - \frac{1}{R_{\text{eff}}(i,j)}(e_i - e_j)(e_i - e_j)^T \succeq 0$$

which by definition of positive semidefinite equals

$$x^T L_G x - x^T \Big(\frac{1}{R_{\text{eff}}(i,j)}(e_i - e_j)(e_i - e_j)^T \Big) x \geq 0$$

is the same as

$$(x(i) - x(j))^2 \leq R_{\text{eff}}(i,j) x^T L_G x$$

if $x(i) = x(j)$ the inequality holds, otherwise, notice that a scaling of x maintains the inequality, then we will prove for $x \in \mathbb{R}^n$ such that $x(i) - x(j) = R_{\text{eff}}(i,j)$, we get

$$R_{\text{eff}}(i,j) \leq x^T L_G x$$

to show that this is true, consider the function $y = b^T x - \frac{1}{2} x^T L_G x$, notice that the maximum must satisfy $\frac{\partial y}{\partial x(i)} = 0$, that is

$$b(i) = \sum_{(i,j) \in E} c(i,j)(x(i) - x(j))$$

so that the vector of potentials p maximizes y, furthermore for $b^T \mathbf{1} = 0$ we have

$$2b^T p - p^T L_G p = 2p^T L_G p - p^T L_G p = p^T L_G p = R_{\text{eff}}(i,j)$$

since $b^T x = (x(i) - x(j)) = R_{\text{eff}}(i,j)$ we obtain

$$2R_{\text{eff}}(i,j) - p^T L_G p \leq R_{\text{eff}}(i,j)$$

this means

$$R_{\text{eff}}(i,j) \leq x^T L_G x$$

which concludes the proof.

□

We can now apply theorem 5 and lemma 7 to obtain

$$\begin{aligned}
Pr\left[(1-\epsilon)\frac{L_G}{k} \preceq \frac{1}{k}L_{G'} \succeq (1+\epsilon)\frac{L_G}{k}\right] \geq & 1 - 2n\exp\left(-\epsilon^2 8n\ln(n)/4\epsilon^2(n-1)\right) \\
\geq & 1 - 2n\exp\left(-2\ln n\right) \\
= & 1 - 2n/n^2 \\
= & 1 - 2/n
\end{aligned}$$

so that graph G' returned by SPARSIFY(G) is a spectral sparsifier with probability $1 - 2/n$.

6 Conjugate gradient

We know develop an iterative algorithm for solving systems of the form $Ax = b$, consider the function $f(x) = \frac{1}{2}x^T Ax - b^T x$, we can write it as

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a(i,j)x(i)x(j) - \sum_{i=1}^{n} b(i)x(i)$$

then its derivative with respect to x is

$$\frac{\partial f(x)}{\partial x(i)} = \frac{1}{2}\sum_{i=1}^{n} a(i,j)x(j) + \frac{1}{2}\sum_{i=1}^{n} a(j,i)x(j) - b(i)$$

this is $f'(x) = \frac{1}{2}A^T x + \frac{1}{2}x^T A - b$, if A is symmetric then

$$f'(x) = Ax - b \tag{5}$$

so f's critical points satisfy $Ax = b$, additionally if A is positive semidefinite then this critical point is the global minimum, to see this consider $x \in \mathbb{R}^n$ such that $Ax = b$, and $e \in \mathbb{R}^n$ an error term, then

$$\begin{aligned}
f(x+e) = & \frac{1}{2}(x+e)^T A(x+e) - b^T(x+e) \\
= & \frac{1}{2}x^T Ax - b^T x + e^T b + \frac{1}{2}e^T Ae - b^T e \\
= & f(x) + \frac{1}{2}e^T Ae
\end{aligned} \tag{6}$$

if A is positive semidefinite, f reaches its minimum at x with $Ax = b$.

6.1 Conjugate directions

Suppose we want to reach the minimum by iteratively choosing a direction d_i and a step length α_i so that our solution on step $i+1$ is

$$x_{i+1} = x_i + \alpha_i d_i \tag{7}$$

we define the error term in step i as $e_i = x_i - x^*$, with x^* the minimum of f, and the residual $r_i = b - Ax_i$, if we want to move in each direction d_i only once we need for e_{i+1} to be orthogonal to d_i, thus

$$\begin{aligned} d_i^T e_{i+1} &= 0 \\ d_i^T (x_i - x^* + \alpha_i d_i) &= 0 \\ \alpha_i &= -\frac{d_i^T e_i}{d_i^T d_i} \end{aligned}$$

but our knowledge of e_i depends on our knowledge of x^*, which is exactly what we are trying to find, to avoid this problem we will make the direction d_i *A-orthogonal* to the error term e_{i+1}, for two vectors x_i and x_j we say they are A-orthogonal if $x_i A x_j = 0$, so we want $d_i A e_{i+1} = 0$, then α_i is

$$\alpha_i = -\frac{d_i^T A e_i}{d_i^T A d_i} = -\frac{d_i^T (Ax_i - b)}{d_i^T A d_i} = \frac{d_i^T r_i}{d_i^T A d_i} \tag{8}$$

which we can find with the information we have, to prove that we only need to move in each direction d_i once, given a set of A-orthogonal directions $d_0, d_1, ..., d_{n-1}$, we write the initial error e_0 as

$$e_0 = \sum_{i=0}^{n-1} \eta_i d_i$$

using the fact that the search directions are A-orthogonal we can find the values of η_i by

$$d_i^T A e_0 = \sum_{j=0}^{n-1} \eta_j d_i^T A d_j = \eta_i d_i^T A d_i$$

thus

$$\eta_i = \frac{d_i^T A e_0}{d_i^T A d_i} = \frac{d_i^T A (e_0 + \sum_{j=1}^{i-1} \alpha_i d_i)}{d_i^T A d_i} = \frac{d_i^T A e_i}{d_i^T A d_i} = -\alpha_i$$

The error in step i is

$$\begin{aligned} e_i &= e_0 + \sum_{j=0}^{i-1} \alpha_j d_j \\ &= \sum_{j=0}^{n-1} \eta_j d_j - \sum_{j=0}^{i-1} \eta_j d_j \\ &= \sum_{j=i}^{n-1} \eta_j d_j \end{aligned} \tag{9}$$

So on each step we are canceling one component of the error, therefore $e_n = 0$, this is, we find the solution in n steps.

We now need a way to find these A-orthogonal directions, for this we use a *conjugate Gram-Schmidt* process, suppose we have n linearly independent vectors $v_0, v_1, ..., v_{n-1}$, we can construct d_i by taking $d_0 = v_0$ and for $i > 0$:

$$d_i = v_i + \sum_{k=1}^{i} \beta_{ik} d_k \qquad i > k \tag{10}$$

To find the values of B_{ik} we use the fact that we want the vectors d_i to be A-orthogonal to each other, this gives

$$\begin{aligned} d_i^T A d_j &= v_i^T A d_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T A d_j \\ 0 &= v_i^T A d_j + \beta_{ik} d_j^T A d_j \qquad i > j \\ \beta_{i,j} &= -\frac{v_i^T A d_j}{d_j A d_j} \end{aligned}$$

note that we keep keep all previous search directions and since matrix vector multiplication takes $O(n^2)$ time, using $e_0, e_1, ..., e_{n-1}$ as a choice for $v_0, v_1, ..., v_{n-1}$ means we need $O(n^3)$ operations and $O(n^2)$ memory to compute the directions $d_0, d_1, ..., d_{n-1}$ needed to find the solution to the system $Ax = b$, this isn't any better than gaussian elimination, the algorithm known as conjugate gradient shows that by being clever on the choice of this gram-schmidt basis we can reduce the running time of the algorithm.

Let $D_i = span\{d_0, d_1, ..., d_{n-1}\}$, we define the A-norm of vector v as

$$||v||_A = \sqrt{v^T A v}$$

Lemma 8. *The method of conjugate directions minimizes $||e_i||_A$ in the set $e_0 + D_i$*

Proof. We begin by proving that conjugate gradient minimizes $f(x_i)$ over the set $span\{d_0\}$, to show this note that by (5)

$$\begin{aligned} \frac{\partial f(x_{i+1})}{\partial \alpha_i} &= 0 \\ f'(x_{i+1})^T \frac{d}{d\alpha} x_{i+1} &= 0 \\ (Ax_{i+1} - b)^T d_i &= 0 \\ d_i^T A e_{i+1} &= 0 \end{aligned}$$

this means that on each step, the method of conjugate directions is minimizing

$f(\hat{x})$ for $\hat{x}$ in the set, $x_i + span\{d_i\}$, then we have

$$\begin{aligned} f(x_i) - f(x^*) &= \frac{1}{2}x_i^T A x_i - b^T x_i + \frac{1}{2}x^* A x^* \\ &= \frac{1}{2}x_i^T A x_i - \frac{1}{2}x^{*T} A x_i - \frac{1}{2}x_i^T A x^* + \frac{1}{2}x^* A x^* \\ &= \frac{1}{2}(x_i - x^*)^T A (x_i - x^*) \\ &= \frac{1}{2}||e_i||_A^2 \end{aligned}$$

this implies that by minimizing $f(e_{i+1})$ we are also minimizing $||e_{i+1}||_A$ over the set $e_i + span\{d_i\}$, thus inductively, we are minimizing it over $e_0 + D_i$. A simpler argument is this, write $e_i = \sum_{j=i}^{n-1} \gamma_j d_j$ for scalars γ_j, then

$$\begin{aligned} ||e_i||_A &= (\sum_{j=i}^{n-1} \gamma_j d_j)^T A (\sum_{j=i}^{n-1} \gamma_j d_j) \\ &= \sum_{j=i}^{n-1} \sum_{k=i}^{n-1} \gamma_j \gamma_k d_j A d_k \\ &= \sum_{j=i}^{n-1} \gamma_j^2 d_j^T A d_j \end{aligned}$$

since it only depends on future directions and they appear on any $e \in e_0 + D_i$ we conclude that e_i minimizes $||e||_A$ over $e_0 + D_i$.

□

One important property for the development of conjugate gradient is that the residual r_i is orthogonal to D_i, seeing that for $i < j$

$$d_i^T r_j = -d_i^T A e_j = -d_i^T A (\sum_{k=j}^{n-1} \gamma_k d_k) = -\sum_{k=j}^{n-1} \gamma_k d_i^T A d_k = 0$$

the fact that the search directions constructed from the vectors $v_0, v_1, ..., v_{n-1}$ imply that r_i is orthogonal to $span\{v_0, v_1, ..., v_{n-1}\}$ is orthogonal, recall from (10) that $d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k$, then

$$\begin{aligned} d_i^T r_j &= v_i^T r_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T r_j \\ 0 &= v_i^T r_j \qquad i < j \end{aligned}$$

note that this also implies

$$d_i^T r_i = v_i^T r_i \tag{11}$$

one final useful property of conjugate directions is that we can find the residuals by the recurrence

$$r_{i+1} = -A e_{i+1} = r_i - \alpha_i A d_i \tag{12}$$

6.2 The conjugate gradient method

The main idea of conjugate gradient is setting $u_i = r_i$, since the search directions d_i are obtained from r_i, it implies $span\{r_0, r_1, ..., r_{n-1}\} = D_i$. as each residual is orthogonal to D_i this means

$$r_i^T r_j = 0 \quad i \neq j$$

from (12) we have that $D_{i+1} = D_i \cup AD_i$, thus $D_i = span\{d_0, Ad_0, ..., A^{i-1}d_0\}$, by setting $d_0 = r_0 = b$

$$D_i = span\{b, Ab, ..., A^{i-1}b\}$$

this is called a krylov subspace, note that since $AD_i \subseteq D_{i+1}$, as the residual r_{i+1} is orthogonal to D_{i+1} it implies that r_{i+1} is A-orthogonal to D_i, so gram-schmidt is simplified because r_{i+1} is A-orthogonal to all previous directions except d_i, by using (12)

$$\begin{aligned} r_i^T r_{j+1} &= r_i^T r_j - \alpha_j r_i^T A d_j \\ \alpha_j r_i^T A d_j &= r_i^T r_j - r_i^T r_{j+1} \\ r^T A d_j &= \begin{cases} \frac{r_i^T r_i}{\alpha_i} & \text{if} i = j \\ -\frac{r_i^T r_i}{\alpha} & \text{if} i = j+1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

recall that gram-schmidt coefficients $\beta_{i,j} = -\frac{u_i^T A d_j}{d_j A d_j}$ with $i > j$, this means

$$\beta_{i,j} = \begin{cases} \frac{r_i^T r_i}{\alpha_{i-1} d_{i-1} A d_{i-1}} & \text{if} i = j+1 \\ 0 & \text{otherwise} \end{cases}$$

we now set $\beta_i = \beta_{i,i-1}$, then using (8) and (11) we get

$$\begin{aligned} \beta_i &= \frac{d_{i-1}^T A d_{i-1}}{d_{i-1}^T r_{i-1}} \frac{r_i^T r_i}{d_{i-1} A d_{i-1}} \\ &= \frac{r_i^T r_i^T}{d_{i-1}^T r_{i-1}} \\ &= \frac{r_i^T r_i^T}{r_{i-1}^T r_{i-1}} \end{aligned}$$

the conjugate gradient algorithm is given by

CGSolve(A)

1 $x_0 = 0$
2 $r_0 = b$
3 $d_0 = r_0$
4 **for** $(i = 0)$ **to** $n - 1$
5 $\quad \alpha_i = \frac{r_i^T r_i}{d_i A d_i}$
6 $\quad x_{i+1} = x_i + \alpha_i d_i$
7 $\quad r_{i+1} = r_i - \alpha_i A d_i$
8 $\quad \beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$
9 $\quad d_{i+1} = r_{i+1} + \beta_{i+1} d_i$

We have seen $D_i = span\{r_0, Ar_1, ..., A^{n-1}r_{i-1}\}$, as $x_i \in D_i$ it can be expressed as $x_i = x_0 + \sum_{j=0}^{i-1} \gamma_j A^j r_0$, consider the $i-1$ degree polynomial $q(x) = \sum_{j=0}^{i-1} \gamma_j A^j$, then

$$x_i = x_0 + q(A)r_0 = x_0 + q(A)A(x - x_0)$$

the error term can be expressed as

$$\begin{aligned} e_i &= x_i - x^* \\ &= x_0 + q(A)A(x - x_0) - x^* \\ &= (x_0 - x^*)(I - q(A)A) \\ &= p(A)e_0 \end{aligned}$$

with $p(A)$ the i degree polynomial $(I - q(A)A)$, we notice that this implies a 1 to 1 relationship between points in $x_0 + D_i$ and degree i polynomials p_i with $p_i(0) = 1$, so that for every $x_t \in x_0 + D_i$ there is an associated polynomial p_t with $p_t(0) = 1$, similarly given a polynomial p_t we can construct a solution x_t. Consider a orthonormal eigenbasis $v_0, v_1, ..., v_{n-1}$ with associated eigenvalues $\lambda_0, \lambda_1, ..., \lambda_{n-1}$ of A, then it is clear that $P(A)v_i = P(\lambda_i)v_i$, we write

$$e_0 = \sum_{j=0}^{n-1} \phi_j v_j$$

which means for e_i

$$e_i = \sum_{j=0}^{n-1} \phi_j p(\lambda_j) v_j$$

$$||e_i||_A = \sum_{j=0}^{n-1} \phi_j^2 p(\lambda_j)^2 \lambda_j$$

recall that conjugate gradient minimizes $||e_i||_A$, then we have

$$\begin{aligned} ||e_i||_A^2 &\leq \min_{p_i} \max_{\lambda_j} p(\lambda_j)^2 \sum_{j=0}^{n-1} \phi_j^2 \lambda_j \\ &= \min_{p_i} \max_{\lambda_j} p(\lambda_j)^2 ||e_0||_A \end{aligned} \tag{13}$$

we know develop a special type of polynomial that will be useful for the analysis of the algorithm.

6.3 Chebyshev polynomials

For an integer define the chebyshev polynomial of degree n by

$$T_n = \cos(n \cos^{-1} x) \quad x \in [-1, 1]$$

to see that it is a polynomial let $\theta = \cos^{-1} x$ for $\theta \in [0, \pi]$, then $T_n(x) = \cos n\theta$, thus

$$T_1(x) = x$$

$$T_2(x) = \cos(2\theta) = \cos^2(\theta) - \sin^2\theta = 2\cos^2\theta - 1 = 2x^2 - 1$$

in general for T_n we have

$$T_{n-1}(x) = \cos((n-1)\theta) = \cos n\theta \cos\theta + \sin n\theta \sin\theta$$

$$T_{n+1}(x) = \cos((n+1)\theta) = \cos n\theta \cos\theta - \sin n\theta \sin\theta$$

this means

$$T_{n+1} + T_{n-1} = 2\cos n\theta \cos\theta = 2xT_n(x)$$

$$T_{n+1} = 2xT_n(x) - T_{n-1}$$

inductively T_{n+1} is a polynomial, from the definition it can be easily seen that $|T_n(x)| \leq 1$, and T_n has n roots in the range $[-1, 1]$, particullary in $\frac{(2i-1)\pi}{2}$ for $i = 1, 2, ..., n$, finally we use the fact that

$$\cos\theta = \frac{1}{2}(e^{i\theta} + e^{-i\theta})$$

$$\sin(\cos^{-1} x) = \sqrt{1 - x^2}$$

to write

$$\begin{aligned} T_n &= \frac{1}{2}(e^{in\theta} + e^{-in\theta}) \\ &= \frac{1}{2}((\cos\theta + i\sin\theta)^n + (\cos\theta - i\sin\theta)^n) \\ &= \frac{1}{2}((x + i\sqrt{1-x^2})^n + (x - i\sqrt{1-x^2})^n) \\ &= \frac{1}{2}((x + \sqrt{x^2-1})^n + (x - \sqrt{x^2-1})^n) \end{aligned}$$

We now consider the function

$$P_t(x) = \frac{T_t(\frac{\lambda_{max}+\lambda_{min}-2x}{\lambda_{max}-\lambda_{min})})}{T_t(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}})}$$

note that $P_t(0) = 1$ and for $\lambda_{min} \leq x \leq \lambda_{max}$ the argument in the numerator is in $[-1, 1]$, this implies that

$$|P_t(x)| \leq T_t\Big(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}}\Big)^{-1}$$

let $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$, we will call this κ the *condition number* of matrix A, then

$$\begin{aligned}|P_t(x)| &\leq T_t(\frac{\kappa+1}{\kappa-1})^{-1} \\ &= \left(\left(\frac{\kappa+1}{\kappa-1} + \sqrt{\Big(\frac{\kappa+1}{\kappa-1}\Big)^2 - 1}\right)^t + \left(\frac{\kappa+1}{\kappa-1}\right) - \sqrt{\Big(\frac{\kappa+1}{\kappa-1}\Big)^2 - 1}\right)^t\Bigg)^{-1}\end{aligned}$$

by using $\sqrt{\big(\frac{\kappa+1}{\kappa-1}\big)^2 - 1} = \frac{2\sqrt{\kappa}}{\kappa-1}$ we get

$$|P_t(x)| \leq 2\Big(\Big(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\Big)^t + \Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^t\Big)^{-1} \leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^t$$

hence by (13)

$$||e_i||_A^2 = \min_{p_i} \max_{\lambda_j} p(\lambda_j)^2 ||e_0||_A \leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^t ||e_0||_A$$

we have

$$2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^t = \Big(1 - \frac{2}{\sqrt{\kappa}}\Big)^t \leq 2e^{-2t/(\sqrt{\kappa}+1)}$$

we can now state formally the running time of the conjugate gradient algorithm

Theorem 6. *Using the conjugate gradient algorithm, we can reduce the error by an ϵ term, this is $||e_i||_A \leq \epsilon||e_0||_A$, in $O(\sqrt{\kappa}\ln(\frac{1}{\epsilon}))$ iterations, for a time complexity of $O(\sqrt{\kappa}m\ln(\frac{1}{\epsilon}))$, with m the number of nonzero entries of the matrix A.*

7 Preconditioning conjugate gradient

We saw that the time complexity of the conjugate gradient algorithm depends on 2 values, m which is the number of nonzero entries of matrix A, and the condition number κ, we can reduce the values of m by using a sparsifier, we now show how we can reduce κ. We call a symmetric positive definite matrix

M a *preconditioner* of A if M approximates A, $\kappa(MA) \ll \kappa(A)$, and we can solve $Mx = b$ efficiently for any b, then solving

$$MAx = Mb$$

will be faster than solving $Ax = b$, however, MA isn't necessarily symmetric, notice that since M is symmetric there exist orthogonal P and diagonal D such that $PDP^T = M$, and as M is positive definite then $D^{1/2}$ with $D^{1/2}(i,i) = \sqrt{D(i,i)}$ exists, if we set $E = PD^{1/2}$ it is easy to see that

$$M = EE^T$$

We remark that the matrix E isn't unique, another way to find E could be, for example, to use Cholesky factorization. We can notice that if v is an eigenvector of MA with associated eigenvalue λ, then

$$(EAE^T)(E^{-T}v) = (E^{-T}E^T)(EAv) = E^{-T}M^TAv = \lambda E^{-T}v$$

this means that $\kappa(MA) = \kappa(EAE^T)$, EAE^T is also symmetric and positive semidefinite, it is symmetric because as A is symmetric, then

$$(EAE^T)^T = EA^TE^T = EAE^T$$

and it is positive semidefinite as A is PSD

$$x^TEAE^Tx = y^TAy \geq 0$$

with $y = E^Tx$, we can then solve

$$EAE^T\hat{x} = Eb \qquad \hat{x} = E^{-T}x$$

so we modify the conjugate gradient method to

$$\hat{d_0} = \hat{r}_0 = Eb - EAE^T\hat{x}_0$$

$$\alpha_i = \frac{\hat{r}_i^T\hat{r}_i}{\hat{d}_iEAE^T\hat{d}_i}$$

$$\hat{x}_{i+1} = \hat{x}_i + \alpha_i\hat{d}_i$$

$$\hat{r}_{i+1} = \hat{r}_i - \alpha_iEAE^T\hat{d}_i$$

$$\beta_{i+1} = \frac{\hat{r}_{i+1}^T\hat{r}_{i+1}}{\hat{r}_i^T\hat{r}_i}$$

$$\hat{d}_{i+1} = M\hat{r}_{i+1} + \beta_{i+1}\hat{d}_i$$

however we would need to calculate E^{-T}, fortunately we can use a substitution of the variables $\bar{r_i} = Er_i$, $\bar{d_i} = E^Td_i$ $\bar{x_i} = E^Tx_i$ to obtain

$$\bar{r}_0 = b - Ax_0$$

$$\bar{d_0} = Mr_0$$
$$\alpha_i = \frac{r_i^T M r_i}{d_i A d_i}$$
$$\bar{x}_{i+1} = x_i + \alpha_i d_i$$
$$\bar{r}_{i+1} = r_i - \alpha_i A d_i$$
$$\beta_{i+1} = \frac{r_{i+1}^T M r_{i+1}}{r_i^T M r_i}$$
$$\bar{d}_{i+1} = M\bar{r}_{i+1} + \beta_{i+1}\bar{d}_i$$

8 Low-stretch spanning trees

Consider a graph $G = (V, E)$ and a weight/length function w on its edges, we say that the *distance* $d_G(u, v)$ between two vertices $u, v \in V$ is the length of the shortest path between u and v in G, it is easy to see that $d_G(u, u) = 0$, d_G is symmetric, this is, $d_G(u, v) = d_G(v, u)$, and also satisfies the triangular inequality $d_G(u, v) \leq d_G(u, w) + d_G(w, v)$, then (V, d_G) forms a *metric space*.

Suppose we want to build a spanning tree $T = (V, E_T)$ so that we can use T to solve problems in G, with the hope that problems will be easier to solve in T than in G, can we build a tree such that $d_T(u, v) = d_G(u, v)$?, consider a clique on n vertices with unit length, then every vertex has $n - 1$ edges connected with every other vertex in the graph, as a tree only has $n - 1$ edges, most pairs of vertices don't have edges connecting them, this is $d_T(u, v) \geq 2$ while $d_G(u, v) = 1$. Instead suppose we try to find a tree T such that for a given α and for all $u, v \in V$

$$d_G(u, v) \leq d_T(u, v) \leq \alpha d_G(u, v) \tag{14}$$

so that the distances in T approximate the distances in G to a factor of α, we are interested in making α as small as possible, then how small can we make α such that for every graph we can find a tree satisfying (14)? consider the cycle on n vertices with unit length, then consider the single edge (u, v) not in the tree, then $d_G(u, v) = 1$ but $d_T(u, v) = n - 1$, this means that making α less that $n - 1$ is impossible in general, the way we will solve this problem is to focus on making this α small on average

Definition 5. *A low stretch spanning tree T of stretch α is a probability distribution $\mathcal{D}$ of spanning trees of G such that for all $u, v \in V$*

$$d_G(u, v) \leq d_T(u, v)$$
$$\mathbb{E}_{T \sim D}[d_T(u, v)] \leq \alpha d_G(u, v)$$

with this definition we can achieve what we want?, namely, trees that approximate the all pairs shortest path, and thus have a small stretch α, consider the two counter examples we gave earlier, for the complete graph, let the spanning tree be a star graph, for any pair of vertices u, v, the distance between

them is 1 if any of them is at the center, and 2 otherwise, then the expected distance is $\frac{2}{n} \cdot 1 + \frac{n-2}{n} \cdot 2 = 2 - \frac{2}{n}$, for the cyclic graph, select one edge uniformly at random, this gives each edge a chance of $1/n$ of being selected and thus have a stretch of $n-1$, the expected distance is $\frac{1}{n} \cdot (n-1) + \frac{n-1}{n} \cdot 1 = 2 - \frac{2}{n}$.

This new definition shows that it doesn't have the same problems we had earlier, we will now show that we can make $\alpha \approx O(\log n)$

Theorem 7. *For a graph G, we can find a spanning tree T such that $\alpha = str_T(G) = O(m \log n \log\log(n))$, in $O(m \log n \log\log(n))$ time, with $m = |E|$.*

Even though the proof for this theorem is beyond the scope of this monograph, we will prove a similar result for finite metric spaces

Definition 6. *A low stretch tree with stretch α for a metric space $M = (V, d)$ is a probability distribution $\mathcal{D}$ over the vertex set V such that for all $u, v \in V$*

$$d_G(u, v) \leq d_T(u, v)$$

$$\mathbb{E}_{T \sim D}[d_T(u, v)] \leq \alpha d_G(u, v)$$

The difference with definition 5 is that we have a metric instead of a graph, and we are not restricted to spanning trees, we also define for a metric space $M = (V, d)$ its *aspect ratio* Δ as

$$\Delta_M = \left(\frac{\max_{u \neq v \in V} d(u, v)}{\min_{u \neq v \in V} d(u, v)}\right)$$

Theorem 8. *For any metric space $M = (V, d)$ there is a low stretch tree T with stretch $O(\log n \log \Delta_M)$*

to prove this result we begin by defining the *diameter* of a set S as the maximum distance between two points in the set, that is $\max_{u,v \in S} d(u, v)$.

Definition 7. *A low diameter decomposition scheme or LDD scheme with parameter β for a metric space $M = (V, d)$ is a randomized algorithm that given $D > 0$ partitions the set V into clusters $C_1, C_2, ..., C_t$ such that for all $i \in \{1, ..., t\}$, the diameter of C_i is at most D, and for all $x, y \in V$ with $x \neq y$, let C_x be the cluster containing x and C_y be the cluster containing y, we have $Pr[C_x \neq C_y] \leq \beta \frac{d(x,y)}{D}$*

as the aspect ratio is invariant to rescaling, we let the smallest distance be 1, so the largest is Δ, a basic algorithm for this LDD scheme is

LDD($M = (V, d), D$)
1 $p = min(1, \frac{4 \log n}{D})$
2 **while** there exists an unmarked point
3 $v =$ any unmarked point
4 sample $R_v \sim Geometric(p)$
5 $C_v = \{\text{unmarked } u | d(v, u) < R_v\}$
6 mark points in C_v
7 **return** set of clusters

we claim that this algorithm satisfies:

1. the diameter of every cluster if D with probability at least $1 - 1/n$
2. any pair $x, y \in V$ is separated with probability no greater than $2pd(x, y)$

Proof. 1. We begin by showing that $R_v \leq D/2$ for all clusters, because for any $x, y \in C_v$ by the triangular inequality

$$d(x, y) \leq d(x, v) + d(v, y) < D/2 + D/2 = D$$

the probability that $R_v > D/2$, given that the CDF of the geometric distribution is given by $Pr[X < x] = 1 - (1 - p)^x$, is

$$Pr[R_v > D/2] = (1 - p)^{D/2} \leq e^{-pD/2} \leq e^{-2 \log n} = \frac{1}{n^2}$$

then using Boole's inequality we have that the probability if a cluster C_i having diameter larger than D is

$$1 - Pr[\exists i \text{ s.t } (R_i > D)] \geq 1 - \frac{n}{n^2} = 1 - \frac{1}{n}$$

2. Remember that sampling from a geometric distribution can be seen as flipping a coin with bias p until we get a heads, then the algorithm can be seen as starting with a unit radius ball around v, repeatedly flip a coin with bias p, if it comes up heads we set the radius to R_v, build the cluster C_v and find a new V, if it comes up tails we increase the radius by one and flip again, this is repeated until all points belong to a cluster.

   We now use the fact that the process of sampling from the geometric distribution is memory-less, that is, even after performing k flips, the time until the first head is geometrically distributed, for x, y, consider the first time one one of this points belong to the current cluster centered at v, without loss of generality let this point be x, at this point there have been $d(v, x)$ flips, x and y will be separated if we see a heads in the next $\lceil d(v, y) - d(v, x) \rceil \leq \lceil d(x, y) \rceil$ flips, the probability of getting a head among these flips is, using Boole's inequality once again, bounded by

$$\lceil d(x, y) \rceil p \leq 2pd(x, y) \leq 8 \log n \frac{d(x, y)}{D}$$

   the first inequality comes from the fact that the minimum distance is 1, so the ceil of any distance doubles it at most.

□

To obtain the diameter bound with probability 1, repeat the algorithm until the obtained partition has clusters of diameter no more than D, the probability of any u, v being separated is at most the probability of being separated in any of the runs, this is at most $pd(u, v)$ times the expected number of runs

$$pd(u, v)(1/(1 - 1/n)) \leq 2pd(u, v) = O(\log n) \frac{d(u, v)}{D}$$

note that this implies that the LDD scheme achieves $\beta = O(\log n)$.

We now give the algorithm for finding a low stretch spanning tree, the idea is given a metric with diameter Δ, we use LDD to decompose it into clusters of diameter $D < \Delta/2$, build a tree recursively in these components and combine them. To obtain a low stretch spanning tree, we will run $\text{LST}(M, \lceil \log_2 \Delta \rceil)$

LST(metric $M = (V, d), \delta$)
1 **if** $|V| = 1$
2 **return** tree containing the single point in V
3 $C_1, C_2, ..., C_t = LDD(M, D = 2^{\delta-1})$
4 **for** $j = 1, j \leq t$
5 $M_j =$ metric M restricted to C_j
6 $T_j = \text{LST}(M_j, 2^{\delta-1})$
7 Add edges of length 2^δ from root r_1 of tree T_1 to the roots of $T_2, ..., T_t$
8 **return** tree rooted at r_1

To prove theorem 8 we will first prove the following lemma

Lemma 9. *If the random tree returned by some call* $\text{LST}(M', \delta)$ *has root* r*, then every vertex has* x *in* T *has distance* $d(x, r) \leq 2^{\delta+1}$ *and the expected distance between any* $x, y \in T$ *is* $\mathbb{E}[d_T(x, y)] \leq 8\delta\beta d(x, y)$

Proof. Using induction on δ, the base case is trivial, suppose it is true for $\delta - 1$, let $x \in C_j$, by hypothesis we have $d(x, r) \leq 2^\delta$, and the distance to the new root is 2^δ more, giving $2^\delta + 2^\delta = 2^{\delta+1}$.

For the second claim suppose it is true for $\delta - 1$, if any pair x, y is separated by the LDD with probability $\beta \frac{d(x,y)}{2^{\delta-1}}$, in that case their distance is no bounded by

$$d(x, r) + d(r, y) \leq 2^{\delta+1} + 2^{\delta+1} = 4 \cdot 2^\delta$$

otherwise if they lie in the same cluster they inductively have expected distance $8\delta\beta d(x, y)$, then

$$\begin{aligned}
\mathbb{E}[d_T(x, y)] &\leq Pr[C_y \neq C_x] 4 \cdot 2^\delta + Pr[C_y = C_x] 8\delta\beta d(x, y) \\
&\leq \beta \frac{d(x, y)}{2^{\delta-1}} 4 \cdot 2^\delta + 8\delta\beta d(x, y) \\
&= 8\delta\beta d(x, y)
\end{aligned}$$

□

By the fact that $\beta = O(\log n)$ and $\delta = O(\log \Delta)$ we have proved theorem 8.

8.1 Preconditioning with low-stretch spanning trees

One matrix that satisfies the requirements to be a preconditioner for a Laplacian system is the pseudo inverse of the Laplacian matrix of a spanning tree T, by

using Gaussian elimination from leaves to root, we can solve $L_T x = b$ in $O(n)$ time, and $L_T \preceq L_G$

$$x^T L_T x = \sum_{(i,j)\in T} c(i,j)(x(i) - x(j))^2 \leq \sum_{(i,j)\in G} c(i,j)(x(i) - x(j))^2 = x^T L_G x$$

this means that $I \preceq L_T^+ L_G$, so $\lambda_1(L_T^+ L_G) \geq 1$, we can now focus on finding a bound for the largest eigenvalue of $L_T^+ L_G$, we will bound its trace instead

$$\begin{aligned} Tr(L_T^+ L_G) &= Tr(L_T^+ \sum_{(i,j)\in E} c(i,j)(e_i - e_j)(e_i - e_j)^T) \\ &= \sum_{(i,j)\in E} c(i,j) Tr(L_T^+ (e_i - e_j)(e_i - e_j)^T) \\ &= \sum_{(i,j)\in E} c(i,j)(e_i - e_j)^T L_T^+ (e_i - e_j) \end{aligned} \tag{15}$$

we note that $(e_i - e_j)^T L_T^+ (e_i - e_j)$ is the effective resistance $R_{\text{eff}}(i,j)$, to see how we can use this to bound λ_n we begin by defining the *distance* $d(i,j)$ of an edge as the inverse of its weight, in the case of an electrical network this value corresponds to its resistance, then for an edge (i,j) in G, if its not in T, then its distance will be the sum of the distances in the $i,j-$Path $P(i,j)$ in T, consider the quotient of the distance in T over its distance in G, this is

$$\sum_{(i,j)\in E} \frac{1}{d(i,j)} \sum_{(u,v)\in P(i,j)} d(i,j) \tag{16}$$

we will call this value the *stretch* of edge (i,j), the kind of tree we are interested in is called a *low stretch spanning tree*, this is the tree that minimizes the stretch over all edges, then by (3)

$$R_{\text{eff}}(i,j) = \mathcal{E}(f) = \sum_{(u,v)\in P(i,j)} f^2(u,v) r(u,v) = \sum_{(u,v)\in P(i,j)} r(u,v)$$

this is because $f(i,j) = 1$ for any $(u,v) \in P(i,j)$, then by (15) and (16)

$$Tr(L_T^+ L_G) = \sum_{(i,j)\in E} c(i,j) \sum_{(u,v)\in P(i,j)} r(i,j)$$

this is just the stretch of spanning tree T, by theorem 7 we have that $\lambda_n = O(m \log n \log\log(n))$, we can use this to conclude that conjugate gradient needs $O(\sqrt{m} \log n)$ steps to find a solution, thus we can solve a Laplacian system in $O(m^{3/2} \log n \log 1/\epsilon)$.

9 Almost linear time Laplacian solver

9.1 Cholesky decomposition

One way to solve a Laplacian system $Lx = b$ is to find a lower triangular matrix $\mathcal{L}$ such that $\mathcal{L}\mathcal{L}^T = L$, then the solution will be given by

$$x = (\mathcal{L}^T)^{-1}\mathcal{L}^{-1}b$$

this can be done in $O(n^2)$ time by using forward and backward substitution to apply $\mathcal{L}^{-1}$ and $\mathcal{L}^{-1}$, this is, use the triangular structure of the matrix to solve one row at a time from sparsest to densest, we call this decomposition $\mathcal{L}\mathcal{L}^T$ the *Cholesky decomposition* of matrix L. In general if we have a real symmetric positive semi-definite matrix M we can write it as

$$\begin{vmatrix} d & -a^T \\ -a & M_2 \end{vmatrix}$$

with $d \in \mathbb{R}$, $a \in R^{n-1}$ and $M_2 \in \mathbb{R}^{nxn}$, them we can construct the rank 1 matrix given by

$$d^{-1}\begin{vmatrix} d \\ -a \end{vmatrix}\begin{vmatrix} d \\ -a \end{vmatrix}^T = \begin{vmatrix} d & -a^T \\ a & d^{-1}aa^T \end{vmatrix}$$

we define *Schur complement of M with respect to the first coordinate* as

$$M/1 = M - d^{-1}\begin{vmatrix} d \\ -a \end{vmatrix}\begin{vmatrix} d \\ -a \end{vmatrix}^T = \begin{vmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & M_2 - d^{-1}aa^T \end{vmatrix}$$

we can note two things for $M_2 - d^{-1}aa^T$, it is the sum of two real symmetric matrices and therefore is a real symmetric matrix, and it is positive semi-definite, to see this let x be an arbitrary vector in $\mathbb{R}^n$ $\alpha = d^{-1}a^Tx$, then

$$0 \leq \begin{vmatrix} \alpha \\ x \end{vmatrix}^T M \begin{vmatrix} \alpha \\ x \end{vmatrix} = d\alpha^2 - 2\alpha a^Tx + x^TM_2x = x^T(M_2 - d^{-1}aa^T)x$$

this is, we can inductively repeat this procedure for each coordinate, reducing the size of the non-zero block in each step. This will be useful to find the Cholesky decomposition of a matrix, a fact we establish in the following theorem.

Theorem 9. *For a real symmetric matrix $M \in R^{n\times n}$ we can find a decomposition $M = \mathcal{L}\mathcal{L}^T$, where $\mathcal{L}$ is lower triangular, in $O(n^3)$ time.*

Proof. the algorithm begins by setting $S_0 = M$ and repeatedly eliminate the ith coordinate, to do this at each step we let

$$l_i = \frac{1}{\sqrt{S_i(i,i)}}S(:i)$$

if $S_i(i,i) = 0$ let $l_i = 0$, then S_i will be the Schur complement with respect to the ith coordinate, that is

$$S_i = S_{i-1}/i = S_{i-1} + l_il_i^T \tag{17}$$

after n steps we have $S_n = \mathbf{0}^{n\times n}$, let $C = \begin{vmatrix} l_1 & l_2 & ... & l_n \end{vmatrix}$, as S_{i-1} is supported in the coordinates $\{i, i+1, ..., n\}$ so is l_i, meaning that C is lower triangular, then we will have the Cholesky factorization $M = CC^T$, this is because

$$M = S_0 - S_n = \sum_{i=0}^{n-1} S_i - S_{i+1} = \sum_{i=0}^{n-1} l_i l_i^T = CC^T$$

calculating each Schur complement takes $O(n^2)$ time, and we have to find n of them, giving a time complexity of $O(n^3)$.

□

9.2 Cholesky decomposition on graphs

When applying the algorithm we just developed, each step like (17) can be seen as a combinatorial operation in the graph, given the interesting structure of Laplacians

$$L = \begin{vmatrix} d & -a^T \\ -a & L_2 \end{vmatrix}$$

with $d > 0$, $a > 0$ and $a^T\mathbf{1} = d$, then again defining $l = \frac{1}{\sqrt{d}} \begin{vmatrix} d \\ -a \end{vmatrix}$ so that $ll^T = \begin{vmatrix} d & -a^T \\ a & d^{-1}aa^T \end{vmatrix}$, we can calculate the Schur complement with respect to the first vertex by

$$L/1 = L - ll^T = \begin{vmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & L_2 - d^{-1}aa^T \end{vmatrix}$$

we will show that $L_2 - d^{-1}aa^T$ is a graph laplacian, and in the process see how calculating the Schur complement can be seen as a combinatorial operation on the graph, begin by defining STAR(1) as the Laplacian restricted to the edges adjacent to the first vertex, let diag($\mathbf{a}$) be the diagonal matrix induced by $\mathbf{a}$, then

$$\text{STAR}(1) = \begin{vmatrix} d & -\mathbf{a}^T \\ -\mathbf{a}^T & \text{diag}(\mathbf{a}) \end{vmatrix}$$

then by adding and subtracting it in the Schur complement we obtain

$$L/1 = (L - \text{STAR}(1)) + (\text{STAR}(1) - ll^T)$$

the term inside the first parenthesis is a Laplacian matrix, this is because is the sum of two Laplacian matrices, to see that the second term is also a Laplacian we use its matrix form

$$\text{STAR}(1) - ll^T = \begin{vmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \text{diag(a)} - d^{-1}aa^T \end{vmatrix} \tag{18}$$

its easy to see that its symmetric, the elements in the diagonal are non-negative and the elements off the diagonal are non-positive, finally using the fact that $a^T 1 = d$ we have

$$(\text{STAR}(1) - ll^T)\mathbf{1} = \text{diag(a)}\mathbf{1} - d^{-1}aa^T\mathbf{1} = a - a = \mathbf{0}$$

this proves the second term is a Laplacian, and as a result $L/1$ is the sum of two Laplacian matrices and therefore a Laplacian. Interestingly, as can be seen in (18) this is the matrix associated with a clique on the vertices adjacent to vertex 1.

In general if we want to eliminate vertex v from Laplacian S, then

$$\text{STAR(v,S)} = \sum_{(v,u)\in E} (e_v - e_u)(e_v - e_u)^T$$

let $l = \frac{1}{\sqrt{S(v,v)}} S(:,v)S(:v)^T$, we define CLIQUE(v,L) = (STAR(v,L) $- ll^T$), so that

$$L/v = L - \text{STAR(v,L)} + \text{CLIQUE(v,L)}$$

this means that calculating the Schur complement of a Laplacian on vertex v can be seen as eliminating the star induced by vertex v, then adding the clique on the vertices adjacent to v induced by eliminating v. The algorithm is given by

CHOLESKY(L)
1 $S_0 = L$
2 **for** $v = 0, i < n$
3 $\quad l_v = \frac{1}{\sqrt{S_{i-1}(i,i)}} S_{i-1}(:,v)S_{i-1}(:,v)$
4 $\quad S_i = S_{i-1} - \text{STAR(v,L)} + \text{CLIQUE(v,L)}$
5 $l_n = \mathbf{0}^n$
6 **return** $[l_1, ..., l_n]$

We already showed how in general calculating the Schur complements is really expensive, making the calculation of a Cholesky factorization take $O(n^3)$ time, however, for Laplacian matrices calculating Schur complements can be seen as graphs operations, it gives some insight as to why it might be so expensive, at each step we are eliminating a vertex but adding a clique to the remaining subgraph, making it denser, the main idea to solve this problem will be to instead of finding and adding the entire clique we will use a sparse approximation via an unbiased sampler.

9.3 Sparse Cholesky

Suppose we have a unbiased clique sampler routine CLIQUE-SAMPLE, then we can approximate the Cholesky factorization by

SPARSECHOLESKY(L)
1 $S_0 = L$
2 **for** $v = 0, i < n$
3 $\quad l_v = \frac{1}{\sqrt{S_{i-1}(i,i)}} S(:,v)S(:,v)$
4 $\quad S_i = S_{i-1} - \text{STAR(v,L)} + \text{CLIQUE-SAMPLE(v,S)}$
5 $l_n = \mathbf{0}^n$
6 **return** $[l_1, ..., l_n]$

we begin by showing that the solution given by the algorithm is gives a cholesky factorization on expectation.

Lemma 10. *If we have an unbiased sampler* CLIQUE-SAMPLER*, then the output of the* SPARSE-CHOLESKY *procedure,* $\mathcal{L} = \text{SPARSE-CHOLESKY}(\text{L})$ *satisfies*

$$\mathbb{E}[\mathcal{L}\mathcal{L}^T] = L$$

Proof. Let $L_0 = L$ and $S_i = S_{i-1} + \sum_{i=0}^{i-1} l_j l_j^T$, note that $L_n = \sum_{i=0}^{n-1} l_j l_j^T = \mathcal{L}\mathcal{L}^T$, then conditional on the samples before the ith elimination we have

$$\begin{aligned}
\mathbb{E}[L_i] &= \mathbb{E}\left[S_{i-1} + \sum_{i=0}^{i-1} l_j l_j^T\right] \\
&= \mathbb{E}\left[S_{i-2} - \text{STAR}(v, S_{i-2}) + \text{CLIQUE-SAMPLE}(v, S_{i-2}) + \sum_{i=0}^{i-1} l_j l_j^T\right] \\
&= S_{i-2} - \text{STAR}(v, S_{i-2}) + \mathbb{E}\big[\text{CLIQUE-SAMPLE}(v, S_{i-2})\big] + \sum_{i=0}^{i-1} l_j l_j^T \\
&= S_{i-2} - \text{STAR}(v, S_{i-2}) + \text{CLIQUE}(v, S_{i-2}) + \sum_{i=0}^{i-1} l_j l_j^T \\
&= S_{i-2} - \text{STAR}(v, S_{i-2}) + \text{STAR}(v, S_{i-2}) + l_{i-1} l_{i-1}^T + \sum_{i=0}^{i-1} l_j l_j^T \\
&= S_{i-1} + \sum_{i=0}^{i-2} l_j l_j^T \\
&= L_{i-1}
\end{aligned}$$

Thus, chaining expectations we have

$$\mathbb{E}[\mathcal{L}\mathcal{L}^T] = \mathbb{E}[L_n] = L$$

□

Now we need a clique sampler routine to use in SPARSE-CHOLESKY, we begin by showing a simple algorithm for this, we will also prove that it is indeed unbiased and give a bound on its running time and the number of non zeroes in the returned matrix, then in the next section we show some practical improvements on this routine. First we give some notation that will be useful when describing our first clique sampler routine, we will denote by $b_{i,j} \in \mathbb{R}^n$ the vector such that

$$b_{i,j}(k) = \begin{cases} 1 & \text{if} \quad k = i \\ -1 & \text{if} \quad k = j \\ 0 & \text{otherwise} \end{cases}$$

given a weight function w and a vertex v, we will denote

$$w_v = \sum_{(u,v)\in E} w(u,v)$$

that is, the sum of all edges adjacent to v.

simpleCliqueSample(v, S)
1 $Y_v = \mathbf{0}^{n\times n}$
2 **for** each edge (v, i)
3 pick a neighbor j of v with probability $\frac{w(j,v)}{w_v}$
4 **if** $i \neq j$
5 $Y_v = Y_v + \frac{w(i,v)w(j,v)}{w(i,v)+w(j,v)} b_{i,j} b_{i,j}^T$
6 **return** Y_v

Using an algorithm known as the alias method, we can sample an edge in $O(1)$ time, giving a complexity of $O(deg_s(v))$ for any vertex v.

Lemma 11. *Let* $Y_v =$ simpleCliqueSample(v,S)*, then*

$$\mathbb{E}[Y_v] = \mathit{CLIQUE}(v, S)$$

Proof. We begin by noting that both Y_v and CLIQUE(v, S) are laplacians, therefore, it is sufficient to show that $\mathbb{E}[Y_v](i, j) =$CLIQUE(v, S) for $i \neq j$

$$\begin{aligned}
\mathbb{E}[Y_v](i,j) &= -\frac{w(i,v)w(j,v)}{w(i,v)+w(j,v)}\left(\frac{w(i,v)}{w_v} + \frac{w(j,v)}{w_v}\right) \\
&= -\frac{w(i,v)w(j,v)}{w_v} \\
&= \text{CLIQUE}(v,S)(i,j)
\end{aligned}$$

□

Finally we state a theorem without proof, this theorem states that this sparseCholesky procedure gives an approximation to the Cholesky decomposition and gives a running time bound, the proof can be found on [Tro19].

Theorem 10. *The lower triangular matrix* $\mathcal{L}$ *produced by the* sparseCholesky(L) *algorithm satisfies*

$$0.5L \preceq \mathcal{L}\mathcal{L}^T \preceq 1.5L$$

additionally, let $n = |V|$ *and* $m = |E|$*, then* $\mathcal{L}$ *has* $O(m \log^2(n))$ *nonzero entries, and takes* $O(m \log^3(n))$ *time.*

9.4 Improving the clique sampler

In this section we will provide some ideas that are proven to work better in practice, but first we will give a condition for the algorithm to have a better bound on the number of non zeroes than theorem 10 on the number of non zeroes of the returned matrix.

Lemma 12. *Suppose* cliqueSample(v, S) *produces at most as many edge samples as the degree of* v*, let* $\bar{V}$ *be the current vertex set, if* sparseCholesky *is run with an elimination rule that satisfies when vertex* v *is eliminated*

$$deg_s(v) \leq O\left(\frac{1}{|\bar{V}|}\sum_{u\in\bar{V}} deg_s(u)\right)$$

then sparseCholesky *outputs a lower triangular matrix with* $O(|E|\log|E|)$ *non zeroes. If the bound on the degree of* v *holds in expectation over a random choice of* v*, then* sparseCholesky *outputs a lower triangular matrix with* $O(|E|\log|E|)$ *non zeroes in expectation.*

Proof. Since in each step we are sampling a number of edges smaller than the number of edges being eliminated, we see that S contains at most $|E|$ edges, the number of non zeroes in l_i is proportional to the degree of the vertex being eliminated, which is on average the average degree in S, this is at most $\frac{2|E|}{n-i}$, thus the total number of non zeroes in $\mathcal{L}$ is

$$\sum_{i=1}^{n-1}\frac{2|E|}{n-i} = O(|E|\log|E|)$$

if the degree bound holds in expectation, $O(|E|\log|E|)$ also holds in expectation. □

9.4.1 First idea

The first improved algorithm will be based on the idea of elimination stars, suppose we pick an ordering on the neighbors of the vertex being eliminated, we will eliminate the neighbors in order by increasing vertex number, consider eliminating vertex $v = 1$, we can break CLIQUE$(1, S)$ into a sum of smaller terms, each of them a Laplacian containing edges from neighbor i of 1 to neighbors $l \geq i$, as before let the first column of the Laplacian be $\begin{vmatrix} d \\ a \end{vmatrix}$, the elimination star graph will be given by

$$\text{elimStar}(1, i, S) = \sum_{j>i}\frac{a(i)a(j)}{d}(e_i - e_j)(e_i - e_j)^T \tag{19}$$

from this definition we can see that

$$\text{CLIQUE}(1, S) = \sum_{i:(1,i)\in E}\text{elimStar}(1, i, S) \tag{20}$$

our improved sampling algorithm will focus in approximating the clique by approximating each elimination star by a single random reweighted edge so that in expectation this edge laplacian gives elimStar$(1, i, S)$, chose a random index γ_i according to

$$Pr[\gamma_i = j] = p_i(j) = \begin{cases} \frac{a(j)}{\sum_{l>i} a(l)} & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}$$

then if the outcome is $\gamma_i = j$, we choose a weight of

$$\tilde{w}(i,j) = \frac{1}{p_i(j)} \frac{1}{d} a(j)a(i) \tag{21}$$

and output the Laplacian $\tilde{S}_i = \tilde{w}(i,j)(e_i - e_j)(e_i - e_j)^T$

Corollary 2. $\mathbb{E}[\tilde{S}_i] =$ elimStar$(1, i, S)$

Proof.

$$\begin{aligned} \mathbb{E}[\tilde{S}_i] &= \sum_{j>i} p_i(j)\tilde{w}(i,j)(e_i - e_j)(e_i - e_j)^T \\ &= \sum_{j>i} \frac{1}{d} a(j)a(i)(e_i - e_j)(e_i - e_j)^T \\ &= \text{elimStar}(1, i, S) \end{aligned}$$

□

The algorithm will work by combining the sample from each elimination star, giving the approximation

$$\sum_{i:(1,i)\in E} \tilde{S} \approx \text{CLIQUE}(1, S)$$

this is

cliqueTreeSample(v,S)
1 $\tilde{C} = \mathbf{0}^{n\times n}$
2 $d = S(v,v), a = S(v,:)$
3 **for** i s.t $a(i) \neq 0$
4 $\quad c = a(i), a(i) = 0$
5 $\quad$ sample index j with probability $p_i(j)$
6 $\quad \tilde{C} = \tilde{C} + \tilde{w}(i,j)(e_i - e_j)(e_i - e_j)^T$
7 **return** $\tilde{C}$

we can now show that this sampler is unbiased

Lemma 13. $\mathbb{E}[$cliqueTreeSample(v,S)$] = CLIQUE(v, S)$

Proof. Using corollary 2 and equation (20)

$$\begin{aligned}\mathbb{E}[\tilde{C}] &= \mathbb{E}\left[\sum_{i:(i,v)\in E} \tilde{S}\right] \\ &= \sum_{i:(i,v)\in E} \mathbb{E}[\tilde{S}] \\ &= \sum_{i:(i,v)\in E} \text{elimStar}(v,i,S) \\ &= \text{CLIQUE}(v,S)\end{aligned}$$

□

This means that as the output of cliqueTreeSample is unbiased, then by using it in the sparseCholesky algorithm in place of CLIQUE-SAMPLE, by lemma 10 the output matrix $\mathcal{L}$ satisfies $\mathbb{E}[\mathcal{L}\mathcal{L}^T] = L$, additionally, the algorithm has the useful property that the output is a Laplacian of a connected graph, in fact, the returned Laplacian corresponds to a tree on the neighbors of v in S.

Lemma 14. *The random matrix returned by* cliqueTreeSample(v, S) *is the Laplacian of a tree on the neighbors of* v *in* S*.*

Proof. Suppose we have an enumeration $1, ..., k$ on the neighbors of v, as in the i-th sample vertex i is connected with a vertex $j > i$, by induction from $k-1$ to 1, for the base case $k-1$ is connected to k deterministically, for the inductive step observe that the sample connects vertex i with a vertex $i+1, ..., k$, where by the induction hypothesis we already have a tree, and this new graph forms a tree as we are connecting $k-i+1$ vertices with $k-i$ edges.

□

Even though it would make sense to eliminate the vertex of smallest degree in the current graph, this would require us to use a priority queue, however, this would be too costly to use, instead we will eliminate vertices such that lemma 12 holds, by using an order in which the degree of the vertex being eliminated is bounded by $\frac{2|E|}{\bar{n}}$, with $\bar{n}$ the number of vertices on the current graph, then the resulting matrix will have $O(m \log m)$ non zeroes.

9.4.2 Improving accuracy

A second idea that has been shown to work better in practice is to split each edge into k multiedges, with weight $\frac{w(e)}{k}$ each, we can do this because the Laplacian only depends on the sum of multiedge weights between every pair of vertices, meaning that the new multigraph has the same Laplacian as the original graph.

We begin by modifying the sparseCholesky algorithm to keep and modify not only the Laplacian, but also the actual graph, this is necessary to make it work with multiedges and the cliqueTreeSampleMultiedge.

sparseCholeskyMultiedge(L)

$S_0 = L$
$G_0 =$ a multigraph with L as Laplacian
for $v = 0, i < n$
    $l_v = \frac{1}{\sqrt{S_{i-1}(i,i)}} S(:,v)S(:,v)$
    $S_i = S_{i-1} -$ STAR(v,L) + cliqueTreeSampleMultiedge(v, S, G)
$l_n = \mathbf{0}^n$
return $[l_1, ..., l_n]$

The sampling algorithm will work similar to cliqueTreeSample but splits each edge into k copies of multiedges, up to a maximum of l multiedges, this helps limiting how dense the multigraph can become.

cliqueTreeSampleMultiedge(l, v, S, G)

$\tilde{C} = \mathbf{0}^{n \times n}$

// the following values change on each iteration
Let $N(G, v)$ be the set of neighbor vertices of v
Let $E(G, v, i)$ be the set of multiedges between v and i
Let $w(G, v, i) = \sum_{e \in E(G,v,i)} w(e)$
Let $w(G, v) = \sum_{i \in N(G,v)} w(G, v, i)$

// d doesn't change on each iteration
$d = w(G, v)$ // initial weight of multiedges incident to v

for all vertices $i \in N(G, v)$
    $t = \min(|E(G, v, i)|, l)$
    $\bar{w}_{vi} = w(G, v, i)/t$
    Delete multiedges $E(G, v, i)$ from G
    $w_{new} = \bar{w}_{vi} \frac{w(G,v)}{d}$
    for $h = 1, h <= t$
        Sample index j with probability $p(j) = \frac{w(G,v,j)}{w(G,v)}$
        $\tilde{C} = \tilde{C} + w_{new}(e_i - e_j)(e_i - e_j)^T$
        Add a multiedge between i and j of weight w_{new} to G
return $\tilde{C}$

It is easy to see that this sampling algorithm is unbiased, by letting the result of the inside for be $\tilde{S}_i$, by a direct calculation we get $\mathbb{E}[\tilde{S}] =$ elimStar(v, i, S), then we can use the same argument given before, thus the proof is omitted.

Lemma 15. $\mathbb{E}[$cliqueTreeSample$(l, v, S, G)] = CLIQUE(v, S)$

Using Theorem 6, Theorem 10 and Lemma 12 we conclude that we can solve a Laplacian system using conjugate gradient in $O(m \log m \ln \frac{1}{\epsilon})$ plus $O(m \log^3 n)$ for the preconditioner, this is

Theorem 11. *We can solve a Laplacian system in $\tilde{O}(m)$ time, where $\tilde{O}(h(m)) = O(h(m)\log^k h(m))$ for some constant k.*

10 Solving the maximum flow problem

In this section we will give an application of the methods we developed over the course of this monograph, in particular, we will show how solving Laplacian systems can be used to find solutions to the maximum flow problem in undirected graphs.

10.1 The problem

Given an undirected graph $G = (V, E)$ with capacities $u : E \to \mathbb{R}$, suppose we are given a source vertex s and a sink vertex t, we define a *flow* f from s to t as an assignment of values $f(i, j)$ to each edge, such that

$$0 \leq f(i,j) \leq u(i,j) \tag{22}$$

this is, the flow respects capacities, and for each vertex j

$$\sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k) \tag{23}$$

this is, the flow entering each vertex equals the amount leaving, we define the value of the flow as the flow leaving the source, this is

$$|f| = \sum_{(s,i)\in E} f(s,i) - \sum_{(s,i)\in E} f(s,i)$$

the maximum flow problem consist on finding the flow f^* that maximizes $|f|$ over all flows $|f|$.

10.2 The multiplicative weights algorithm

The first approach will be to use the multiplicative weights algorithm, the idea of the algorithm is to make a series of choices given a number of steps $t = 1, ..., T$, at each step we choose one of m possible choices, after making a choice i in time t and before time $t+1$, we get a value $v_t(i) \in [0, 1]$ that is unknown before making the choice, the algorithm will try to maximize this result for each step, that is, it will try to find $\max_j \sum_{t=1}^{T} v_t(j)$.

The algorithm will set a fixed parameter ϵ with $0 \leq \epsilon \leq 1/2$, and will work by keeping a set of weights $w(i)$ for each choice i, at each step we make a choice proportional to weight w, and updates the weights by a factor $(1 + \epsilon v_t(i))$ for all i, which means that in the long term, choices that do better have a higher probability in the future.

We will use this algorithm to find a nearly feasible solution to a packing problem, given by

$$Mx \leq \mathbf{1}, \quad x \in Q \tag{24}$$

with Q a convex set, we assume that $x \geq 0$ for all $x \in Q$, we could define the problem as $Mx \leq b$ for $\mathbb{R}^n$, however without loss of generality we can transform to this problem into (24) by dividing each row of M by b_i, the algorithm will use an *oracle* subroutine that given p can return $x \in Q$ such that $p^T Mx \leq p^T \mathbf{1}$ if it exists, note that if there is no x satisfying $p^T Mx \leq p^T \mathbf{1}$ then (24) has no feasible solution, the algorithm will use the concept of width of the oracle, it is defined as

$$\rho \geq \max_i \max_{x \text{ returned by oracle}} M_i x$$

as we want $Mx \leq \mathbf{1}$, we can see that ρ is the maximum multiplicative factor by which an x returned by the oracle could exceed its desired bound.

PACKINGMULTIPLICATIVEWEIGHTS(M, ϵ)

1 $w(i) = 1$ for $i = 1, 2, ..., m$
2 $T = \frac{\rho}{\epsilon^2} \ln m$
3 **for** $t = 1, t \leq T$
4 $\quad W_t = \sum_{i=1}^{n} w(t)$
5 $\quad p_t(i) = w_t(i)/W_t$
6 $\quad$ Run oracle to find $x_t \in Q$ such that $p_t^T Mx_t \leq p_t^T \mathbf{1}$
7 $\quad v_t(i) = \frac{1}{\rho} M_i x_t$ for $i = 1, ..., m$
8 $\quad w_{t+1}(i) = w_t(i)(1 + \epsilon v_t(i))$ for $i = 1, ..., m$
9 **return** $\bar{x} = \frac{1}{T} \sum_{t=1}^{T} x_t$

We will now provide a result showing that the expected value gained by the algorithm, given by $\sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i)$ is almost as large as the maximum value of any decision j, this will be used later when analyzing the running time and error rate of the algorithm.

Lemma 16. *If $\epsilon < 1/2$, then for any $j = 1, ..., m$*

$$\sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i) \geq (1 - \epsilon) \sum_{i=1} v_t(j) - \frac{1}{\epsilon} \ln m$$

Proof. The argument will be a direct result of an upper bound and lower bound

on w_{T+1}, first the upper bound

$$\begin{aligned} w_{t+1} =& \sum_{i=1}^{m} w_{t+1}(i) \\ =& \sum_{i=1}^{m} w_t(i)(1+\epsilon v_t(i)) \\ =& W_t + \epsilon W_t \sum_{i=1}^{m} v_t(i) \frac{w_t(i)}{W_t} \\ =& W_t(1+\epsilon \sum_{i=1}^{m} v_t(i) p_t(i)) \\ \leq& \exp\left(\epsilon \sum_{i=1}^{m} v_t(i) p_t(i)\right) \end{aligned}$$

this last inequality comes from $1 + x \leq e^x$, then

$$\begin{aligned} W_{T+1} \leq& W_T \exp\left(\epsilon \sum_{i=1}^{m} v_T(i) p_T(i)\right) \\ \leq& W_1 \prod_{i=1}^{T} \exp\left(\epsilon \sum_{i=1}^{m} v_t(i) p_t(i)\right) \\ =& m \exp\left(\epsilon \sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i)\right) \end{aligned}$$

for the lower bound we get

$$W_{T+1} \geq w_{T+1}(j) = \prod_{t=1}^{T} (1+\epsilon v_t(j)) \geq (1+\epsilon)^{\sum_{t=1}^{m} v_t(j)}$$

last inequality uses $(1+\epsilon x) \geq (1+\epsilon)^x$ if $x \in [0,1]$, then combining both bounds we get

$$(1+\epsilon)^{\sum_{t=1}^{m} v_t(j)} \leq m \exp\left(\epsilon \sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i)\right)$$

we take log on both sides to obtain

$$\begin{aligned} \ln(1+\epsilon) \sum_{t=1}^{T} v_t(j) \leq& \ln m + \epsilon \sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i) \\ \sum_{t=1}^{T} \sum_{i=1}^{m} v_t(i) p_t(i) \geq& \frac{1}{\epsilon} \ln(1+\epsilon) \sum_{t=1}^{T} v_t(j) - \frac{1}{\epsilon} \ln m \end{aligned}$$

finally, using $\ln(1+\epsilon) \geq \epsilon - \epsilon^2$ for $\epsilon \in [0, 1/2]$

$$\sum_{t=1}^{T}\sum_{i=1}^{m} v_t(i)p_t(i) \geq (1-\epsilon)\sum_{i=1} v_t(j) - \frac{1}{\epsilon}\ln m$$

□

We can now give the analysis for the algorithm

Lemma 17. *If $\epsilon \leq \frac{1}{2}$ and the oracle returns a value on each iteration then* packingMultiplicativeWeights *returns a solution $\bar{x}$ such that $M\bar{x} \leq (1+4\epsilon)\mathbf{1}$ in $O(\frac{m\rho}{\epsilon^2}\ln m)$ time plus the time for $O(\frac{\rho}{\epsilon^2}\ln m)$ matrix vector products Mx and $O(\frac{\rho}{\epsilon^2}\ln m)$ oracle calls.*

Proof. The running time bound comes from the fact that we run the main loop $\frac{\rho}{\epsilon^2}\ln m$ times, and on each iteration we spend $O(m)$ time updating v_t and w_{t+1}.

For the error, we use the definition of $v_t(i)$ and the fact that the oracle returns a value to obtain

$$\sum_{i=1}^{m} v_t(i)p_t(i) = \frac{1}{\rho}p_t^T M x_t \leq \frac{1}{\rho}p_t^T\mathbf{1} = \frac{1}{\rho}$$

last inequality comes from the fact that p_t makes a probability distribution, then

$$\sum_{t=1}^{T}\sum_{i=1}^{m} v_t(i)p_t(i) \leq \frac{T}{\rho}$$

using lemma 16, for any $j \in \{1, ..., m\}$

$$\begin{aligned}\sum_{t=1}^{T}\sum_{i=1}^{m} v_t(i)p_t(i) \geq& (1-\epsilon)\sum_{i=1} v_t(j) - \frac{1}{\epsilon}\ln m \\ =& (1-\epsilon)\sum_{t=1}^{T}\frac{1}{\rho}M_j x_t - \frac{1}{\epsilon}\ln m \\ =& (1-\epsilon)\frac{T}{\rho}M_j\bar{x} - \frac{1}{\epsilon}\ln m\end{aligned}$$

combining the previous two inequalities we obtain

$$M_j\bar{x} \leq \frac{1}{1-\epsilon}\big(1 + \frac{\rho\ln m}{\epsilon T}\big)$$

then as $T = \frac{\rho}{\epsilon^2}\ln m$ and $0 \leq \epsilon \leq 1/2$

$$M_j\bar{x} \leq \frac{1+\epsilon}{1-\epsilon} \leq 1 + 4\epsilon$$

thus, as the inequality holds for any $j \in \{1, ..., m\}$ we conclude

$$M\bar{x} \leq (1+4\epsilon)\mathbf{1}$$

□

10.2.1 Applying multiplicative weights to the maximum flow problem

For simplicity we will assume that the capacities $u(i,j) = 1$ for all edges (i,j), this will mean that the optimal value of the flow $|f| \in [0, m]$, then we can use an algorithm that gives a flow of value nearly k if it exists, or correctly states that it doesn't exist, then we will be able to use binary search to find this optimal value k. This algorithm will be an application of the multiplicative weights for the packing problem we gave earlier, it finds a feasible solution over a convex region, in this case the region will be the region given by (22), (23) and $|f| = k$, recall that PACKINGMULTIPLICATIVEWEIGHTS used an oracle subroutine that found an x such that $p_t^T M x_t \leq p_t^T \mathbf{1}$, we will now show how finding an $s-t$ electrical flow can be used as an oracle, we want a flow that satisfies

$$\sum_{(i,j)\in E} p_t(i,j) f_t(i,j) \leq \sum_{(i,j)\in E} p_t(i,j) = 1$$

multiplying by W_t we get

$$\sum_{(i,j)\in E} w_t(i,j) f_t(i,j) \leq W_t$$

we will instead give a weaker result that could be dealt with by a rescaling on the flow f

Lemma 18. *For the electrical flow computed in iteration t of the max flow multiplicative weights algorithm*

$$\sum_{(i,j)\in E} w_t(i,j) f_t(i,j) \leq \sqrt{1+\epsilon}\, W_t$$

Proof. Let f^* be a maximum s-t flow, from lemma 6 we have

$$\begin{aligned}
\sum_{(i,,j)\in E} f_t^2(i,j) r_t(i,j) &\leq \sum_{(i,j)\in E} (f^*(i,j))^2 r_t(i,j) \\
&\leq \sum_{(i,j)\in E} r_t(i,j) \\
&= \sum_{(i,j)\in E} (w_t(i,j) + \frac{\epsilon W_t}{m}) \\
&= (1+\epsilon) W_t
\end{aligned} \tag{25}$$

the second inequality comes from assuming unit capacities, now using Cauchy-Schwarz inequality we have

$$\begin{aligned}
\Big(\sum_{(i,j)\in E} w_t(i,j) f_t(i,j) \Big)^2 &\leq (\sum_{(i,j)\in E} f_t^2 w_t(i,j))(\sum_{(i,j)\in E} w_t(i,j)) \\
&\leq (\sum_{(i,j)\in E} f_t^2(i,j) r_t(i,j)) W_t
\end{aligned}$$

Combining both results we get

$$\Big(\sum_{(i,j)\in E} w_t(i,j) f_t(i,j) \Big)^2 \leq (1+\epsilon) W_t^2$$

from which the lemma follows.

□

We can now provide the modified multiplicative weights algorithm that uses a Laplacian solver as oracle.

maxFlowMultiplicativeWeights(M, ϵ)

```
1  w_1(i,j) = 1 for all (i,j) ∈ E
2  T = (ρ/ε²) ln m
3  for t = 1, t ≤ T
4       W_t = Σ_{(i,j)∈E} w_t(i,j)
5       p_t(i,j) = w_t(i,j)/W_t for all (i,j) ∈ E
6       r_t(i,j) = w_t(i,j) + (ε/m) W_t for all (i,j) ∈ E
7       Compute s-t electrical flow f_t of value k using resistances r_t
8       v_t(i,j) = (1/ρ) f_t(i,j) for all (i,j) ∈ E
9       w_{t+1}(i,j) = w_t(i,j)(1 + ε v_t(i,t)) for all (i,j) ∈ E
10 return f̄ = (1/T) Σ_{t=1}^{T} f_t
```

to analyze the algorithm we will need to find the width ρ, by the way we defined the algorithm this turns out to be an upper bound on the maximum value of $f_t(i,j)$ over all t and $(i,j) \in E$, to do this we will once again use the energy of the resulting flow.

Lemma 19. *The width of the oracle computing electrical flows in the max flow multiplicative weights algorithm is at most* $\sqrt{2m/\epsilon}$ *for* $\epsilon \leq 1$.

Proof. To give an upper bound on f_t returned by the oracle, we can use the definition of $r_t = w_t(i,j) + \frac{\epsilon}{m} W_t$ to get

$$f_t^2(i,j) r_t(i,j) \geq f_t^2(i,j) \frac{\epsilon W_t}{m}$$

then by (25)

$$f_t^2(i,j) \leq \frac{(1+\epsilon)m}{\epsilon}$$

and using the fact that $\epsilon \leq 1$ we get $f_t \leq \sqrt{2m/\epsilon}$ □

Then we can use lemma 17, lemma 19 and theorem 11 to give a running time for maximum flow.

Theorem 12. maxFlowMultiplicativeWeights *computes an s-t flow* f *of value* $k/\sqrt{1+\epsilon}$ *with* $f(i,j) \leq (1+4\epsilon)$ *for all* $(i,j) \in E$ *in* $\tilde{O}(m^{1.5}/\epsilon^{2.5})$*, then using bisection we can find a maximum s-t flow in* $\tilde{O}(m^{1.5}/\epsilon^{2.5})$.

11 Solving the minimum cost flow problem.

We conclude this monograph with one final application of Laplacian solvers, in this case we will solve a more general problem, the minimum cost flow problem, it can be shown that by having an efficient way to find minimum cost flows we can efficiently find a solution to the bipartite matching and max flow problems. The algorithm will be based on the interior point method, that uses newtons method for finding roots of polynomials, the algorithm will use the Laplacian solver as a subroutine to quickly calculate newton's method "steps".

11.1 An interior point method

Consider the convex optimization problem

$$\min_{x \in K} f(x)$$

with $f(x)$ a convex function and K a convex set, for simplicity we will assume f to be linear, that is $f(x) = c^T x$ and K to be bounded, we want to transform this problem to an unconstrained one, we will do this by moving the constraints defining k to the objective function in a way that punishes breaking them, so we define the family of functions

$$f_\eta(x) = \eta c^T x + F(x)$$

where $F(x)$ is a convex function defined in the interior of K and takes value ∞ outside of K, we will call this F a *barrier function*, $\eta > 0$ can be seen as a parameter that limits the influence of $F(x)$ in the solution, so that $\lim_{\eta \to \infty} f^*_\eta(x) = f^*(x)$, for the problem we will be interested in we have that the set K is a polyhedron of the form $P = \{x | Ax \leq b\}$, with A an $m \times n$ matrix and $b \in \mathbb{R}^m$, then we can use a logarithmic barrier function

$$F(x) = -\sum_{i=1}^{m} \log(b(i) - a_i^T x) \tag{26}$$

with a_i the ith row of A, note that F is convex and tends to infinity as x gets closer to the boundary of P. We denote x^*_η as the minimizer of f_η in the interior of P, given by int(P)= $\{x | Ax < b\}$, then we define the central path as

$$\Gamma_c = \{x^*_\eta | \eta \geq 0\}$$

thus we have that the central path starts at the *analytic center* x^*_0 and approaches x^*, that is $\lim_{\eta \to \infty} x^*_\eta$, the idea of the algorithm will be to take steps following the central path to get to x^*. On each step we take a *newton step* given by

$$x_{t+1} = x_t - n_\eta(x_t) \quad \text{with} \quad n_\eta(x_t) = (\nabla^2 f(x_t))^{-1} \nabla f(x_t) \tag{27}$$

as $c^T x$ is a linear function we have $\nabla^2 f_\eta(x_t) = \nabla^2 F(x_t)$, so we the resulting algorithm is

IPM$(P = (A, b), c)$

find initial η_0 and x_0 with $||n_{\eta_0}(x_0)||_{x_0} < 1/6$
Let T be such that $\eta_T = \eta_0(1 + \frac{1}{20\sqrt{m}}) > \frac{m}{\epsilon}$
for $i = 0, i \leq T$
    $x_{t+1} = x_t + n_{\eta_t}(x_t)$
    $\eta_{t+1} = \eta_t(1 + \frac{1}{20\sqrt{m}})$
Find $\hat{x}$ by performing to newton with respect to f_{η_T} steps from x_T
return $\hat{x}$

Where $||v||_{x_0} = \sqrt{v^T \nabla^2 f(x_0) v}$, we will skip the analysis of the algorithm and only establish a result regarding convergence and running time.

Theorem 13. *The algorithm* IPM *returns a point* $\hat{x}$ *with*

$$c^T \hat{x} \leq c^T x^* + 2\epsilon$$

in $O(\sqrt{m} \log \frac{m}{\epsilon \eta_0})$ *iterations, where we solve a system of the form* $\nabla^2 F(x) y = z$.

11.2 The problem

Once again we start with a graph $G = (V, E)$, a source vertex s and a sink vertex t, we also have associated with edge edge (i, j) a capacity $u(i, j)$ and a cost $c(i, j)$, and a flow value F, we will try to find a flow f of value F that satisfies capacities as in (22), that minimizes the cost

$$\sum_{(i,j) \in E} f(i, j) c(i, j)$$

this can be expressed as a linear programming problem by

$$\begin{aligned} \min \quad & c^T x \\ s.t \quad & Bx = b \\ & 0 \leq x(i, j) \leq u(i, j) \end{aligned}$$

where $b = F(e_s - e_t)$, we will solve this problem using a interior point method, which needs us to express the problem in a different way, we set slack variables s_i that allow us to write capacity the problem with only equality constraints, we obtain

$$\begin{aligned} \min \quad & c^T x \\ s.t \quad & \begin{vmatrix} B & 0 \\ I & I \end{vmatrix} \begin{vmatrix} x \\ s \end{vmatrix} = \begin{vmatrix} b \\ u \end{vmatrix} \\ & x, y \geq 0 \end{aligned} \tag{28}$$

11.3 Adapting the IPM for the min-cost flow problem

The interior point method we developed requires the polyhedron to be full rank and of the form $\{x|Ax \leq b\}$, we will now adapt this method to problems like (28), more generally, of the form

$$\begin{aligned} \min \quad & c^T x \\ s.t \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{29}$$

Let $Q = \{x|Ax = b\}$, $E = \{x|Ax = 0\}$ and $\{v_1, ..., v_k\}$ be an orthonormal basis for E, we will use H, respectively g to denote the hessian, respectively gradient of f at point x, that is $H(x) = \nabla^2 f(x)$ and $g(x) = \nabla f(x)$, to make our method work on problems like (29) we will restrict the domain of f to Q, we will denote it $\tilde{f}$, then we need to find $\tilde{H}(x)$ and $\tilde{g}(x)$ so that by taking a newton step

$$\tilde{n} = \tilde{H}(x)^{-1}\tilde{g}(x) \tag{30}$$

we stay in Q, thus minimizing $\tilde{f}$. We will use the directional derivatives with respect to the basis $\{v_1, ..., v_k\}$ to restrict the gradient and hessian to Q, so $\tilde{g}(x)$ takes the value

$$\begin{aligned} \tilde{g}(x) =& \sum_{i=1}^{k} v_i \nabla_{v_i} f(x) \\ =& \sum_{i=1}^{k} v_i v_i^T \nabla f(x) \\ =& \Pi_E \nabla f(x) \end{aligned}$$

for the hessian we have that for entry indexed by i and j

$$\begin{aligned} \tilde{H}(x)(i,j) =& \nabla^2_{v_i, v_j} f(x) \\ =& v_i^T H(x) v_j \\ =& (\Pi_E^T H(x) \Pi_E)(i,j) \end{aligned}$$

we can now find $\tilde{n}$ using (30), we can rewrite it as the solution to the linear system

$$\begin{aligned} \tilde{H}(x)h =& -g(x) \\ Ah =& 0 \end{aligned}$$

as Π_E is symmetric and $h \in E$, the first equation becomes

$$\begin{aligned} \Pi_E H(x)h &= -\Pi_E g(x) \\ \Pi_E(H(x)h + g(x)) &= 0 \end{aligned}$$

meaning that $H(x)h + g(x)$ is in the orthogonal complement of E, this is

$$H(x)h + g(x) \in E^{\perp} = \{y | A^T\lambda = y, \lambda \in \mathbb{R}^n\}$$

then we have that

$$\begin{aligned} H(x)h + g(x) &= A^T\lambda \\ h + H(x)^{-1}g(x) &= H(x)^{-1}A^T\lambda \\ Ah + AH(x)^{-1}g(x) &= AH^{-1}A^T\lambda \end{aligned}$$

using (26) and since $\nabla^2 f = \nabla^2 F$, $H(x)$ is a diagonal matrix, and as A is an adjacency matrix we can see that $AH^{-1}A^T$ is a Laplacian matrix, finally, using the fact that $Ah = 0$ we can find λ by solving a Laplacian system

$$\tilde{L}\lambda = \tilde{b}$$

with $\tilde{L} = AH(x)^{-1}A^T$ and $\tilde{b} = AH(x)^{-1}g(x)$, then $\tilde{n}$ can be found by

$$\tilde{n} = H(x)^{-1}(A^T\lambda - g(x))$$

, thus combining 11 and 13 we get the following result.

Theorem 14. *We can solve the minimum cost flow problem using the interior point method in $\tilde{O}(\sqrt{m})$ steps, solving one Laplacian system on every step, for a running time of $\tilde{O}(m^{3/2})$.*

References

[Bol98] Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 1998.

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster Approximate Lossy Generalized Flow via Interior Point Algorithms, April 2008. arXiv:0803.0988.

[GKS23] Yuan Gao, Rasmus Kyng, and Daniel A. Spielman. Robust and Practical Solution of Laplacian Equations by Approximate Elimination, June 2023. arXiv:2303.00709.

[Kyn] Rasmus Kyng. Advanced Graph Algorithms and Optimization, An Interior Point Method for Maximum Flow [lecture notes].

[Li] Jason Li. Advanced Algorithms, low stretch spanning trees [lecture notes].

[She] Jonathan Richard Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.

[Spi] Daniel Spielman. *Spectral and algebraic graph theory.*

[Tro19] Joel A. Tropp. *Matrix Concentration & Computational Linear Algebra.* Caltech CMS Lecture Notes 2019-01. Pasadena, July 2019.

[Vis13] Nisheeth K. Vishnoi. Lx = b. *Foundations and Trends® in Theoretical Computer Science*, 8(1–2):1–141, 2013.

[Vis21] Nisheeth K. Vishnoi. *Algorithms for convex optimization.* Cambridge University Press, New York, 2021.

[Wil19] David P. Williamson. *Network flow algorithms.* Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA, 2019.