Convex Optimization

The first section introduces the notion of convexity. The second section discusses the Lagrangian method, often used to solve the convex optimization exactly. The third section discusses the convergence of numerical methods to solve the convex optimization.

1 Convex Functions

Definition:

1. A set $S \subseteq \mathbb{R}^n$ is called a convex set if any two points in S contain their line, i.e. for any $\boldsymbol{x}, \boldsymbol{y} \in S$ we have that $\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y} \in S$ for any $\theta \in [0,1]$.
2. For a convex set $S \subseteq \mathbb{R}^n$, we say that a function $f : S \to \mathbb{R}$ is convex on S if for any two points $\boldsymbol{x}, \boldsymbol{y} \in S$ and any $\theta \in [0,1]$ we have that $f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$.

1.1 First-order Characterization

Theorem: Let S be an open convex subset of $\mathbb{R}^n$, and let $f : S \to \mathbb{R}$ be a differentiable function. Then, f is convex if and only if for any $\boldsymbol{x}, \boldsymbol{y} \in S$ we have that $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$.

1.2 Second-order Characterization

Theorem: Let $S \subseteq \mathbb{R}^n$ be open and convex, and let $f : S \to \mathbb{R}$ be twice continuously differentiable.

1. $H_f(\boldsymbol{x})$ is positive semi-definite for any $\boldsymbol{x} \in S \Leftrightarrow f$ is convex on S.
2. If $H_f(\boldsymbol{x})$ is positive definite for any $\boldsymbol{x} \in S$ then f is strictly convex on S. The opposite is not true, *e.g.*, for $f(x) = x^4$ at $x = 0$.

2 Convex Duality and Conjugate

2.1 Separating Hyperplanes

Definition: a hyperplane $H : n^\top x = \mu$ separates two sets A and B iff for any $a \in A$ and any $b \in B$, we have $n^\top a \geq \mu$ and $n^\top b \leq \mu$. If both equalities are strict, then we say H strictly separates A and B.

Theorem: For two disjoint convex sets $A, B \in \mathbb{R}^n$, there exists a separating hyperplane H. If A and B are bounded and closed as well, then there exists a strictly separating hyperplane H.

2.2 Lagrange Duality

Definition:

1. **Weak Duality**: the optima of the Lagrange dual problem is always a lower bound of the optima of the primal problem.
2. **Strong Duality**: the optima of the Lagrange dual problem equals the optima of the primal problem. This is implied by Slater's condition (sufficient condition). KKT condition is necessary and sufficient for strong duality.
3. **Slater's condition**: there is a strictly feasible point in the relative interior of the constraint domain. $\text{relint}(S) = \{x \in S : \text{for all } \boldsymbol{y} \in S \text{ there exists } \epsilon > 0 \text{ such that } \boldsymbol{x} - \epsilon(\boldsymbol{y} - \boldsymbol{x}) \in S\}$. The proof of the implication of the strong duality requires separating hyperplane theorem.
4. **KKT condition**: includes the following: (1) primal feasiblity, *i.e.*, the constraint of the primal; (2) dual feasiblity, *i.e.*, the constraint of the dual (nonnegative multiplier for inequality constraint), (3) complementary slackness, *i.e.*, for each inequality constraint, the product of the multiplier and the constraint must be zero, and (4) gradient condition, *i.e.*, the gradient of the Lagrange function is zero.

Idea behind KKT gradient condition: if the gradient of the object is not parallel to the gradient of the constraint, then we can take a step towards the negative gradient direction and project into the constraint set, thus decreasing the object without changing the constraint.

2.3 Fenchel Conjugates

Given a (convex) function $\mathcal{E} : S \subseteq \mathbb{R}^n \to \mathbb{R}$, its Fenchel conjugate is a function $\mathcal{E}^* : \mathbb{R}^n \to \mathbb{R}$ defined as $\mathcal{E}^*(\boldsymbol{z}) = \sup_{\boldsymbol{y} \in S} \langle \boldsymbol{z}, \boldsymbol{y} \rangle - \mathcal{E}(\boldsymbol{y})$.

Properties:

1. The Fenchel conjugate is convex even if $\mathcal{E}$ is not, as the sup of convex functions is convex.
2. For real-valued and continous $\mathcal{E}$ defined on a convex domain, $\mathcal{E}^{**} = \mathcal{E}$.

3 Solving Convex Optimization approximately

3.1 Solving via Gradient Descent

3.1.1 *Vanilla Gradient Descent*

Idea: approximate the object by its first order expansion, *i.e.*, $x_{t+1} = x_t - \alpha \nabla f(x_t)$.

Theorem: Let $f : \mathbb{R}^n \to \mathbb{R}$ be a β-gradient Lipschitz, convex function. Let $\boldsymbol{x}_0$ be a given starting point, and let $\boldsymbol{x}^* \in \arg\min_{x \in \mathbb{R}^n} f(\boldsymbol{x})$ be a minimizer of f. The Gradient Descent algorithm given by $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \frac{1}{\beta} \nabla f(\boldsymbol{x}_i)$ ensures that the kth iterate satisfies $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \leq \frac{2\beta \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{k+1}$.

3.1.2 *Accelerated Gradient Descent*

Theorem: Let $f : \mathbb{R}^n \to \mathbb{R}$ be a β-gradient Lipschitz, convex function. The Accelerated Gradient Descent algorithm given by

$$
\begin{aligned}
a_i &= \frac{i+1}{2}, A_i = \frac{(i+1)(i+2)}{4} \\
\boldsymbol{v}_0 &= \boldsymbol{x}_0 - \frac{1}{2\beta} \nabla f(\boldsymbol{x}_0) \\
\boldsymbol{y}_i &= \boldsymbol{x}_i - \frac{1}{\beta} \nabla f(\boldsymbol{x}_i) \\
\boldsymbol{x}_{i+1} &= \frac{A_i \boldsymbol{y}_i + a_{i+1} \boldsymbol{v}_i}{A_{i+1}} \\
\boldsymbol{v}_{i+1} &= \boldsymbol{v}_i - \frac{a_{i+1}}{\beta} \nabla f(\boldsymbol{x}_{i+1})
\end{aligned}
$$

ensures that the k-th iterate satisfies $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \leq \frac{2\beta \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{(k+1)(k+2)}$.

3.2 Solving via Newton's Method

Idea: approximate the object by its second order expansion, *i.e.*, $x_{t+1} = x_t - \alpha H_f^{-1}(x_t) \nabla f(x_t)$.

Definition:

- K-stable Hessian: we say $\mathcal{E}$ has a K-stable Hessian if there exists a constant matrix $\boldsymbol{A}$ s.t. for all $\boldsymbol{y}$ we have $\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}) \approx_K \boldsymbol{A}$, *i.e.*, $\frac{1}{1+K} \boldsymbol{A} \preceq \boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}) \preceq (1+K)\boldsymbol{A}$.

Theorem (unconstrained): Assume $\mathcal{E}$ has K-stable Hessian, then $\mathcal{E}\left(\boldsymbol{y}_k\right) - \mathcal{E}(\boldsymbol{y}^*) \leq \epsilon\left(\mathcal{E}\left(\boldsymbol{y}_0\right) - \mathcal{E}(\boldsymbol{y}^*)\right)$ when $k > (K+1)^6 \log(1/\epsilon)$ with $\alpha = \frac{1}{(1+K)^2}$. Note: $(K+1)^6$ can be reduced to $K+1$.

To generalize the convergence to the linearly constrained case, notice that the constraint domain is a C-dimensional linear space, and thus can be connected to $\mathcal{R}^C$ via a projection matrix Π_C. Therefore, we can minimize $\mathcal{E}(\Pi_C(x))$ without constraint instead of the linearly constrained $\mathcal{E}(x)$. This does not change the K-stability of Hessian.

For the flow problem, the linear equation in the Newton's step is a Laplacian linear equation, which can be solved quickly.

Spectral Graph Theory

The first two sections discuss graph Laplacian in the spectral domain, by bounding its eigenvalues and introducing its importance (Cheeger's inequality). The random walk section provides a use case of graph spectral in the analysis of convergence, and introduces the importance of Laplacian linear equations. The next two sections show how to solve Laplacian linear equations exactly by applying pseudo inverse, as the Laplacian is non-invertible. The last two sections show how to approximate a graph Laplacian and solve the Laplacian linear equation approximately but more efficiently.

4 Spectral Domain of the Graph Laplacian

We assume the graph has n vertices and m edges. The vertex set is V and the edge set is E.

4.1 Courant-Fischer Theorem

1. **Eigenvalue version**: Let A be a symmetric matrix with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$, then

$$\lambda_i = \min_{\substack{\text{subspace } W \subseteq \mathbb{R}^n \\ \dim(W)=i}} \max_{\substack{x \in W \\ x \neq \mathbf{0}}} \frac{x^\top A x}{x^\top x} = \max_{\substack{\text{subspace } W \subseteq \mathbb{R}^n \\ \dim(W)=n+1-i}} \min_{\substack{x \in W \\ x \neq \mathbf{0}}} \frac{x^\top A x}{x^\top x}.$$

2. **Eigenbasis version**: Let A be a symmetric matrix with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ and corresponding orthonormal eigenvectors $x_1, \ldots, x_n$, then

$$\lambda_i = \min_{\substack{x \perp x_1, \ldots x_{i-1} \\ x \neq \mathbf{0}}} \frac{x^\top A x}{x^\top x} = \max_{\substack{x \perp x_{i+1}, \ldots x_n \\ x \neq \mathbf{0}}} \frac{x^\top A x}{x^\top x}.$$

Note that we also have $\lambda_i = \frac{x_i^\top A x_i}{x_i^\top x_i}$.

Applying Courant-Fischer theorem, we have $\lambda_2 x^\top x \leq x^\top L x \leq \lambda_n x^\top x$ for all $x \perp \mathbf{1}$, as $\mathbf{1}$ is the eigenvector of L corresponding to eigenvalue 0. For connected graphs, $\lambda_2 > 0$.

4.2 PSD Order (Loewner Order)

Defined only for symmetric matrices: $A \preceq B$ iff for all $x \in \mathbb{R}^n$, we have $x^\top A x \leq x^\top B x$. We also define $G \preceq H$ for two graphs G and H iff $L_G \preceq L_H$. We always have $G \succeq H$ if H is a subgraph of G.

Properties:

1. If $A \preceq B$ and $B \preceq C$, then $A \preceq C$.
2. If $A \preceq B$, then $A + C \preceq B + C$ for any symmetric C.
3. If $A \preceq B$ and $C \preceq D$, then $A + C \preceq B + D$.
4. If $A \succ 0$ and $\alpha \geq 1$, then $\frac{1}{\alpha} A \preceq A \preceq \alpha A$.
5. If $A \preceq B$, then $\lambda_i(A) \leq \lambda_i(B)$ for all i. Proof by Courant-Fischer theorem. The converse is not true.
6. For any matrix C, if $A \preceq B$, then $C^\top A C \preceq C^\top B C$.
7. If $0 \preceq A \preceq B$, then $B^{-1} \preceq A^{-1}$.

4.3 Bounding the λ_2 and λ_n

4.3.1 Test Vector Method

Since $\lambda_2 \leq \frac{y^\top L y}{y^\top y}$ for any $y \perp \mathbf{1}$, we can upper bound the λ_2 by any **test vector** y. Similarly, we can lower bound the λ_n by test vectors by $\lambda_n \geq \frac{y^\top L y}{y^\top y}$.

1. For a complete graph K_n, $L = n\boldsymbol{I} - \mathbf{1}\mathbf{1}^\top$ and for any $x \perp \mathbf{1}$ we have $x^\top L x = n x^\top x$. Therefore, $\lambda_2(K_n) = \cdots = \lambda_n(K_n) = n$ and any $x \perp \mathbf{1}$ is an eigenvector.
2. For a path graph P_n, let $x(i) = n + 1 - 2i$ be the test vector which satisfies $x \perp \mathbf{1}$, we get $\lambda_2(P_n) \leq \frac{12}{n^2}$. Let $x(1) = -1$, $x(n) = 1$ and $x(i) = 0$ for other i to be the test vector, we get $\lambda_n(P_n) \geq 1$.
3. For a complete binary tree T_n (depth equals zero for a single root), let $x(i) = 0$ for all non-leaf nodes, $x(i) = -1$ for even-numbered leaf nodes and $x(i) = 1$ for odd-numbered leaf nodes be the test vector, we get $\lambda_n(T_n) \geq 1$. Let $x(1) = 0$, $x(i) = 1$ for the left subtree of the root and $x(i) = -1$ for the right subtree of the root be the test vector, we get $\lambda_2(T_n) \leq \frac{2}{n-1}$.

4.3.2 Consequences of PSD Order

Since $x^\top (D - A) x = \sum_{(u,v)} w(u,v)(x(u) - x(v))^2 \geq 0$ and $x^\top (D + A) x = \sum_{(u,v)} w(u,v)(x(u) + x(v))^2 \geq 0$, we have $D \succeq A$ and $D \succeq -A$. In addition, we have $D \preceq (\max D_{i,i})\boldsymbol{I}$. Therefore, we have $L = D - A \preceq 2D \preceq (2 \max D_{i,i})\boldsymbol{I}$, which implies $\lambda_n \leq 2 \max D_{i,i}$ for any graph. For unit-weight graphs, this means $\lambda_n \leq 2 \max \operatorname{degree}(v)$. The bound is tight for a single-edge graph.

To get lower bounds of $\lambda_2(H)$, we first establish $f(n) H \succeq G$ for some G with known lower bounds on $\lambda_2(G)$. Usually $G = K_n$ because $\lambda_2(K_n) = n$. Then it follows that $\lambda_2(H) \geq \lambda_2(G)/f(n)$.

1. **Path Graph** P_n: Let $G_{i,j}$ denote a unit-weight graph consisting of one edge (i,j) and P_n be the path graph connecting 1 and n. Then $(n-1)P_n \succeq G_{1,n}$. Proof follows from applying Cauchy-Schwartz for $\delta_i := x(i+1) - x(i)$. For weighted paths, we have $G_{1,n} \preceq \left(\sum_{i=1}^{n-1} \frac{1}{w_i}\right) \sum_{i=1}^{n-1} w_i G_{i,i+1}$. Applying path inequality, we have $K_n = \sum_{i<j} G_{i,j} \preceq \sum_{i<j} (j-i) P_{i,j} \preceq \sum_{i<j} (j-i) P_n \preceq n^3 P_n$, which implies $\lambda_2(P_n) \geq \lambda_2(K_n)/n^3 = 1/n^2$.
2. **Any unit-weight graph** G: Define the diameter D of a graph G to be the maximum length of the shortest paths between any two nodes. Let $G_{i,j}^s$ be the shortest path from i to j. Applying path inequality, we have $K_n = \sum_{i<j} G_{i,j} \preceq \sum_{i<j} D G_{i,j}^s \preceq \sum_{i<j} DG \preceq n^2 DG$, which implies $\lambda_2(G) \geq \frac{1}{nD}$.
3. **Complete Binary Tree** T_n: Define G_e be the single-edge graph with edge e, and $T_{i,j}$ be the unique path between i and j. Applying the weighted path inequality, we have $K_n = \sum_{i<j} G_{i,j} \preceq \sum_{i<j} \left(\left(\sum_{e \in T^{i,j}} \frac{1}{w(e)} \right) \left(\sum_{e \in T^{i,j}} w(e) G_e \right) \right) \preceq \left(\max_{i<j} \sum_{e \in T^{i,j}} \frac{1}{w(e)} \right) \left(\sum_{i<j} \sum_{e \in T^{i,j}} w(e) G_e \right)$. For e connecting level i and $i+1$ for $i \in [d-1]$, we set $w(e) = 2^i$. Then $\max_{i<j} \sum_{e \in T^{i,j}} \frac{1}{w(e)} \leq 4$. Since the number of occurrence of e in $T^{i,j}$ for any $i < j$ is upper bounded by $n^2 2^{-i}$, we have $\sum_{i<j} \sum_{e \in T^{i,j}} w(e) G_e \preceq \sum_e w(e) n^2 2^{-i} G_e = \sum_e n^2 G_e = n^2 T_n$. Therefore, $K_n \preceq 4n^2 T_n$, which implies $\lambda_2(T_n) \geq \frac{1}{4n}$.

5 Conductance

Definitions:

1. **Conductance of a vertex subset**: Given $\emptyset \subset S \subset V$, the conductance $\phi(S) := \phi(S) = \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}$, where $\mathbf{vol}(S) := \sum_{v \in S} \operatorname{degree}(v)$. Define $\mathbf{1}_S$ to be the n-dimensional vector with only 1 for the vertices of S and 0 for the vertices of $V \setminus S$. Assuming $\mathbf{vol}(S) \leq \mathbf{vol}(V)/2$, thus $|E(V, V \setminus S)| = \sum_{(u,v) \in E} (\mathbf{1}_S(u) - \mathbf{1}_S(v))^2 = \mathbf{1}_S^\top L \mathbf{1}_S$ and $\mathbf{vol}(S) = \mathbf{1}_S^\top D \mathbf{1}_S$. Then $\phi(S) = \frac{\mathbf{1}_S^\top L \mathbf{1}_S}{\mathbf{1}_S^\top D \mathbf{1}_S}$.
2. **Conductance of a graph**: The conductance $\phi(G) := \min_{\emptyset \subset S \subset V} \phi(S) = \min_{\substack{\emptyset \subset S \subset V \\ \mathbf{vol}(S) \leq \mathbf{vol}(V)/2}} \phi(S)$.
3. ϕ**-expander**: For any $\phi \in (0, 1]$, we call a graph G to be a ϕ-expander if $\phi(G) \geq \phi$.
4. ϕ**-expander decomposition of quality** q: A partition $\{X_i\}$ of the vertex set V is called a ϕ-expander decomposition of quality q if (1) each induced graph $G[X_i]$ is a ϕ-expander, and (2.i) #edges not contained in any $G[X_i]$ is at most $q \cdot \phi \cdot m$. The second condition is equivalent to (2.ii) The partition removes at most $q \cdot \phi \cdot m$ edges.
5. **Normalized Laplacian**: We define the *normalized Laplacian* to be $N := D^{-1/2} L D^{-1/2}$. N is still PSD, with first eigenvalue equals 0 associated with eigenvector $D^{1/2}\mathbf{1}$. By Courant-Fischer theorem, $\lambda_2(N) = \min_{x \perp D^{1/2}\mathbf{1}} \frac{x^\top N x}{x^\top x} = \min_{z \perp d} \frac{z^\top L z}{z^\top D z}$.

5.1 Cheeger's Inequality

Notice that the $\lambda_2(N)$ has similar forms to $\phi(G)$. Cheeger's inequality aims to bound $\phi(G)$ by $\lambda_2(N)$.

Cheeger's Inequality: $\frac{\lambda_2(N)}{2} \leq \phi(G) \leq \sqrt{2\lambda_2(N)}$.

The lower bound is proved by restricting the minimum in $\lambda_2(N)$ to be $z_S = \mathbf{1}_S - \alpha\mathbf{1}$ for some α such that $z_S \perp d$. The upper bound is proved by constructing S for any $z \perp d$ such that $\frac{\mathbf{1}_S^\top L \mathbf{1}_S}{\mathbf{1}_S^\top D \mathbf{1}_S} \leq \sqrt{2 \frac{z^\top L z}{z^\top D z}}$.

6 Random Walks on a Graph

A **random walk on a graph** G is a Markov Chain with transition probability $\mathbb{P}(v_{t+1} = v \mid v_t = u) = w(u,v)/d(u)$ iff $(u,v) \in E$ and 0 otherwise. The transition matrix is thus $W = AD^{-1} = I - D^{1/2} N D^{-1/2}$ and $p_t = W^t p_0$. Define $\boldsymbol{\pi} = \frac{d}{\mathbf{1}^\top d}$, thus $\boldsymbol{\pi} = W\boldsymbol{\pi}$ for any G, so every G has a stationary distribution.

6.1 Lazy Random Walks

A **lazy random walk on a graph** G is a random walk, but has half probability to not move for every step. Assuming that G is connected, the lazy random walk guarantees ergodicity of the Markov Chain, and thus convergence to the stationary distribution. The transition matrix is $\tilde{W} = \frac{1}{2}(I + W) = I - \frac{1}{2}D^{1/2}ND^{-1/2}$.

Relation between lazy random walk and normalized Laplacian: For the i-th eigenvalue ν_i of N associated with eigenvector ψ_i, the $\tilde{W}$ has an eigenvalue $1 - \frac{1}{2}\nu_i$ associated with eigenvector $D^{1/2}\psi_i$. Since $0 \preceq L \preceq 2D$, we have $0 \preceq N \preceq 2\boldsymbol{I}$ and thus $0 \le \lambda_i(N) \le 2$. Therefore, we conclude that all eigenvalues of $\tilde{W} \in [0,1]$.

Dynamics of lazy random walk: Expanding the starting distribution p_0 by the eigenvectors of $\tilde{W}$, we have for some $\{\alpha_i\}$ that $p_0 = \sum_{i=1}^n \alpha_i D^{1/2}\psi_i$. Therefore, we have $p_t = \tilde{W}^t p_0 = \sum_{i=1}^n \alpha_i (1 - \frac{1}{2}\nu_i)^t D^{1/2}\psi_i \to \alpha_1 D^{1/2}\psi_1$ as $\nu_1 = 0$ and $\nu_i > 0$ for $i \neq 1$. Since $\psi_1 \propto D^{1/2}\mathbf{1}$, we have $\psi_1 = \frac{d^{1/2}}{(\mathbf{1}^\top d)^{1/2}}$, thus $\alpha_1 = \psi_1^\top D^{-1/2} p_0 = \frac{\mathbf{1}^\top p_0}{(\mathbf{1}^\top d)^{1/2}} = \frac{1}{(\mathbf{1}^\top d)^{1/2}}$ and $\alpha_1 D^{1/2}\psi_1 = \pi$, which implies $p_t \to \pi$, the stationary distribution.

Rate of Convergence: For any unit-weight connected graph G and any starting distribution p_0, we have $\|p_t - \pi\|_\infty \le e^{-\nu_2 t/2}\sqrt{n}$. Therefore, a larger ν_2 and smaller vertex set means faster convergence, and the convergence rate is exponential. This can be viewed as larger ν_2 implies larger conductance by Cheeger's inequality, which means better connectedness.

6.2 Hitting Time

The expected hitting time from a to s is defined by $\mathbb{E}H_{a,s}$, where $H_{a,s} = \mathbf{argmin}_t\{v_t = s \mid v_0 = a\}$. We want $\mathbb{E}H_{a,s}$ for all vertices a and denote the vector as h, *e.g.*, $h(s) = 0$.

By one-step analysis, we have $h(a) = 1 + \sum_{(a,b)\in E} \frac{w(a,b)}{d(a)} h(b) = 1 + \mathbf{1}_a^\top W^\top h$, and thus $1 = \mathbf{1}_a^\top (\boldsymbol{I} - W^\top)h$. Combining the equation for all vertices except s, we have $\mathbf{1} - \alpha \mathbf{1}_s = (\boldsymbol{I} - W^\top)h$, where α represents the extra freedom from the $n-1$ equations. Multiplying both side by D, we get $d - \alpha d(s)\mathbf{1}_s = (D - A)h = Lh$, which only have solution when $d - \alpha d(s)\mathbf{1}_s \perp \mathbf{1}$. Therefore, $\alpha = \|d\|_1 / d(s)$.

To summarize, by solving $Lh = d - \|d\|_1 \mathbf{1}$, we can get the expected hitting time from all vertices to s. Note that the solution has one extra freedom because $\dim(\ker(L)) = 1$, and the correct expected hitting time is $h - h(s)\mathbf{1}$ to enforce the constraint that $h(s) = 1$. The equation can be solved in $\tilde{O}(m)$.

7 Pseudo-Inverse and Effective Resistance

Given a Laplacian L, its (Moore-Penrose) pseudo inverse is defined to be either of the two equivalents:

1. A matrix L^+ that is (1) symmetric, (2) $L^+ v = 0$ for $v \in \ker(L)$, and (3) $L^+ L v = L L^+ v = v$ for $v \in \ker(L)$.
2. Let λ_i, v_i be the i-th eigenvalue and eigenvector. Then $L^+ = \sum_{\lambda_i \neq 0} \lambda_i^{-1} v_i v_i^\top$.

Property:

- Assume $M = XYX^\top$, where X is real and invertible, and Y is real and symmetric. Let Π_M be the orthogonal projection to the image of M. Then $M^+ = \Pi_M (X^\top)^{-1} Y^+ X^{-1} \Pi_M$.
- For symmetric L, $\Pi_L := \sum_{\lambda_i \neq 0} v_i v_i^\top = L^{+/2} L L^{+/2} = L^+ L = L L^+$ is the orthogonal projection to the image of L, *i.e.*, $\Pi_L v = 0$ for any $v \in \ker(L)$ and $\Pi_v = v$ for any $v \in \mathbf{im}(L)$. For connected G, $\Pi_{L_G} = \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$.

The effective resistance between vertex a and b is defined to be the cost (energy lost) to routing one unit (of positive electric charge) from a to b: $R_{\text{eff}}(a,b) = \min_{Bf = \mathbf{1}_b - \mathbf{1}_a} f^\top R f = \tilde{f}^\top R \tilde{f}$, where $\tilde{f}$ is the electric flow. Let $\tilde{x}$ be the electric voltages, we also have $L\tilde{x} = \mathbf{1}_b - \mathbf{1}_a$, and thus $R_{\text{eff}}(a,b) = \tilde{x}^\top L \tilde{x} = (\mathbf{1}_b - \mathbf{1}_a)^\top L^+ (\mathbf{1}_b - \mathbf{1}_a) = \|L^{+/2}(\mathbf{1}_b - \mathbf{1}_a)\|_2^2$.

Effective Resistance is a distance defined on the vertex pairs, i.e. $R_{\text{eff}}(a,c) \le R_{\text{eff}}(a,b) + R_{\text{eff}}(b,c)$.

8 Solving Laplacian Linear Equations Exactly

8.1 Optimization View

Solving $Lx = d$ is equivalent to solving $\mathbf{argmin}_x -d^\top x + \frac{1}{2}x^\top L x$. By iteratively optimize over x_i, we get a series of similar optimizations. The final optimization is straightforward, then we can back substitute to get x.

8.2 Additive View

Given an invertible square lower/upper triangular matrix M, we can solve $Mx = d$ by back substitution in $O(\mathbf{nnz}(M))$, where $\mathbf{nnz}(M)$ means the number of non-zeros in M. Therefore, if we know the **Cholesky decomposition** $L = MM^\top$ (requires $O(n^3)$), then we can solve $Lx = d = M(M^\top x)$ by (1) solving $My = d$ then (2) solving $M^\top x = y$ in $O(\mathbf{nnz}(M))$.

However, the Laplacian is non-invertible, leading to one diagonal of M equals 0. Therefore, we need to play a trick. Define $\hat{M}$ equals M but has value 1 for the zero diagonal, and $\hat{D}$ be a diagonal matrix that has value 0 at the zero diagonal of M and 1 otherwise. Then $L = \hat{M}\hat{D}\hat{M}^\top$, and each $\hat{M}$ is now invertible. Since $\hat{D}^+ = \hat{D}$, we can find a special solution of $Lx = d$ by (1) solving $\hat{M}z = d$, (2) computing $y = \hat{D}z$, and (3) solving $\hat{M}^\top x = y$. The solution space is obtained by adding a subspace spanned by $\mathbf{1}$.

9 Approximating a Dense Graph in the Spectral Domain

9.1 Concentration of Random Matrices

1. **Chernoff Bound for Bounded independent variables**: Suppose $\{X_i \in \mathbb{R}\}$ are independent random variables and $0 \le X_i \le R$. Let $X = \sum_i X_i$ and $\mu = \mathbb{E}X$. Then for any $0 < \epsilon \le 1$, we have $\mathbb{P}(X \ge (1+\epsilon)\mu) \le \exp(-\frac{\epsilon^2 \mu}{4R})$ and $\mathbb{P}(X \le (1-\epsilon)\mu) \le \exp(-\frac{\epsilon^2 \mu}{4R})$.
2. **Bernstein Bound for independent, zero-mean and bounded variables**: Suppose $\{X_i \in \mathbb{R}\}$ are independent, zero-mean random variables and $|X_i| \le R$. Let $X = \sum_i X_i$, and $\sigma^2 = \mathbf{Var}(X)$. Then for any $t > 0$, we have $\mathbb{P}(|X| \ge t) \le 2\exp(\frac{-t^2}{2Rt + 4\sigma^2})$.
   The proof is similar to Chernoff bound. (1) $\mathbb{P}(X \ge t) = \mathbb{P}(\exp(\theta X) \ge \exp(\theta t)) \le \exp(-\theta t)\mathbb{E}(\exp(\theta X))$, (2) upper bound $\mathbb{E}(\exp(\theta X)) \le \exp(\theta^2 \sigma^2)$ given $\theta \in (0, \frac{1}{R}]$, which allows $\exp(\theta X_i) \le 1 + \theta X_i + (\theta X_i)^2$, and (3) take the minimum among $\theta \in (0, \frac{1}{R}]$.
3. **Bernstein Bound for independent, zero-mean and bounded symmetric matrices**: Suppose $\{X_i \in \mathbb{R}^{n\times n}\}$ are independent, zero-mean, symmetric random matrices and $\|X_i\| \le R$, where $\|\cdot\|$ is the spectral norm (the largest singular value). Let $X = \sum_i X_i$, and $\sigma^2 = \|\sum_{i=1}^n \mathbb{E}X_i^2\|$. Then $\mathbb{P}(\|X\| \ge t) \le 2n\exp(\frac{-t^2}{2Rt + 4\sigma^2})$.

9.2 Matrix Functions

Given a real-valued function $f: \mathbb{R} \to \mathbb{R}$ and a symmetric matrix A with eigen-decomposition $A = V\Lambda V^\top$, we define $f(A) = Vf(\Lambda)V^\top$. This is compatible to the Taylor expansion $f(x) = \sum_i \alpha_i x^i$, as $f(A) = \sum_i \alpha A^i = V(\sum_i \alpha_i f(\Lambda))V^\top = Vf(\Lambda)V^\top$.

Monotonicity: Given $f: \mathcal{D} \to \mathcal{C}$ and partial orders $\le_{\mathcal{C}}$ and $\le_{\mathcal{D}}$, we call f is monotonically increasing w.r.t. these orders iff for all $d_1 \le_{\mathcal{D}} d_2 \in \mathcal{D}$ we have $f(d_1) \le f(d_2)$. For matrix functions, we use the PSD order as the ordering.

Property:

- If the scalar function f is monotonically increasing, then the matrix function $X \to \mathbf{Tr}(f(X))$ is monotonically increasing.
- $\log(\cdot)$ is monotonically increasing.
- The matrix function $(\cdot)^2$ and $\exp(\cdot)$ is **not** monotone.
- $\exp(A) \preceq \boldsymbol{I} + A + A^2$ for $\|A\| \le 1$.
- $\log(\boldsymbol{I} + A) \preceq A$ for $A \succ -\boldsymbol{I}$.
- (Lieb's theorem): $f(A) := \mathbf{Tr}(\exp(H + \log(A)))$ for some symmetric H is concave in the domain of PSD matrices. This is in particular useful with Markov inequality because $\mathbb{P}(\|X\| \ge t) = \mathbb{P}(\lambda_n \ge t) \le \mathbb{P}(\mathbf{Tr}(\exp(\theta X)) \ge \exp(\theta t)) \le \exp(-\theta t)\mathbb{E}(\mathbf{Tr}(\exp(\theta X)))$.

9.3 Spectral Sparsifiers

Given PD matrices A, B and $\epsilon > 0$, we say $A \approx_\epsilon B$ iff $\frac{1}{1+\epsilon}A \preceq B \preceq (1+\epsilon)A$. If $L_G \approx_\epsilon L_{\tilde{G}}$ and $|\tilde{E}| \ll |E|$, we call $\tilde{G}$ a spectral sparsifier of G.
Properties:

- Define $c_G(T) := \sum_{e \in E \cap (T \times V \setminus T)} w(e)$ to be the value of the cut $(T, V \setminus T)$. If $L_G \approx_\epsilon L_{\tilde{G}}$, then for all $T \subset V$, we have $\frac{1}{1+\epsilon} c_G(T) \le c_{\tilde{G}}(T) \le (1+\epsilon) c_G(T)$. The proof is by noticing $c_G(T) = 1_T^\top L_G 1_T$.
- $L \approx_\epsilon \tilde{L} \Leftrightarrow \Pi_L \approx_\epsilon L^{+/2}\tilde{L}L^{+/2}$, as $A \preceq B$ implies $C^\top AC \preceq C^\top BC$ for any $C \in \mathbb{R}^{n\times n}$.
- For $\epsilon \le 1$, if $\|\Pi_L - L^{+/2}\tilde{L}L^{+/2}\| \le \epsilon/2$, then $\Pi_L \approx_\epsilon L^{+/2}\tilde{L}L^{+/2}$.

Theorem: Consider a connected graph $G = (V, E, w)$, with $n = |V|$. For any $0 < \epsilon < 1$ and $0 < \delta < 1$, there exist sampling probabilities p_e for each edge $e \in E$ s.t. if we include each edge e in $\tilde{E}$ independently with probabilty p_e and set its weight $\tilde{w}(e) = \frac{1}{p_e} w(e)$, then with probability at least $1 - \delta$ the graph $\tilde{G} = (V, \tilde{E}, \tilde{w})$ satisfies $L_G \approx_\epsilon L_{\tilde{G}}$ and $|\tilde{E}| \le O\left(n\epsilon^{-2}\log(n/\delta)\right)$. The proof uses Bernstein bounds to prove the concentration of the constructed random graph.

10 Solving Laplacian Linear Equations Approximately

Idea: solving Laplacian linear equations requires $O(n^3)$ to get the Cholesky decomposition, which is expensive when the graph is large. By approximating the Laplacian, we can get an approximation of the solution quickly, especially in sparse graphs.
Given PSD matrix M and $d \in \mathbf{im}(M)$, let $Mx^* = d$. We say that $\tilde{x}$ is an ϵ-approximate solution to $Mx = d$ iff $\|\tilde{x} - x^*\|_M^2 \le \epsilon\|x^*\|_M^2$, where $\|x\|_M^2 = x^\top Mx$. Note that any solution to $Mx = d$ has the same $\|\cdot\|_M^2$, as they differ by a vector in the kernel of M.
Theorem: Given a Laplacian $\boldsymbol{L}$ of a weighted undirected graph $G = (V, E, \boldsymbol{w})$ with $|E| = m$ and $|V| = n$ and a demand vector $\boldsymbol{d} \in \mathbb{R}^V$, we can find $\tilde{\boldsymbol{x}}$ that is an ϵ-approximate solution to $\boldsymbol{Lx} = \boldsymbol{d}$, using an algorithm that takes time $O(m\log^c n \log(1/\epsilon))$ for some fixed constant c and succeeds with probability $1 - 1/n^{10}$. Note that without knowing the Cholesky decomposition in advance, the exact solution requires $O(n^3)$ and $m \le n^2/2$.
Idea: during the exact Cholesky decomposition, we minus a $l_i l_i^T$ term. However, the Laplacian given by $l_i l_i^T$ at each step equals a star graph minus a clique graph, thus a clique is added to the graph every time, making the graph denser and denser. With sampling, we can get a sparse approximation of such cliques and add these approximated cliques instead so that l_i is sparse, resulting in sparse approximated Cholesky decomposition.

Combinatorial Graph Algorithms

The first section introduces the max flow problem and its duality with min cut, and the Ford-Fulkerson's algorithm which may not terminate in a general graph. The second section introduces the Dinic's algorithm, which can solve the max flow in a general graph. The third section introduces the link-cut tree, and how this can be adopted to speed up the Dinic's algorithm. The fourth section introduces how solving max flow can potentially help to solve other problems. The fifth section introduces interior point method, based on solving max flows. The last second briefly discusses how the distance oracle can be used to approximately solve the shortest distance queries.

11 Maximum Flows and Minimum Cuts

11.1 Flows

Definition:

1. *s-t* **flow**: an *s-t* flow is a flow such that $\boldsymbol{Bf} = F(-\mathbf{1}_s + \mathbf{1}_t)$ for some $F \ge 0$, *i.e.*, routes some unit from s to t. F is defined to be $\mathbf{val}(\boldsymbol{f})$.
2. **Maximum flow problem**: Given a source vertex and a sink vertex, maximize F such that $\boldsymbol{Bf} = F(-\mathbf{1}_s + \mathbf{1}_t)$ and $0 \le \boldsymbol{f} \le \boldsymbol{c}$.
3. **Path flow**: an *s-t* path flow is an *s-t* flow that only uses a simple path from s to t.
4. **Cycle flow**: A cycle flow is a flow that only uses a simple cycle, so it does not create net-in or net-out.

Path-cycle decomposition lemma: Any *s-t* flow can be decomposed to a sum of *s-t* path flows and cycle flows such that the summation has at most $\mathbf{nnz}(f)$ terms.
There is always an optimal flow that can be decomposed to only path flows, as the cycle flow does not route anything from *s-t* and removing all cycle flows in an optimal flow creates another optimal flow.

11.2 Cuts

Definition:

1. *s-t* **cuts**: an *s-t* cut is a cut $(S, V \setminus S)$ such that $s \in S$ and $t \in V \setminus S$.
2. **Minimum cut problem**: Given two vertices s and t, minimize the cut value $c_G(S) = \sum_{e \in E \cap (S \times V \setminus S)} w(e)$ such that $s \in S$ and $t \in V \setminus S$.

If there is no feasible *s-t* flow, then define S to be the set of vertices reachable from s, $(S, V \setminus S)$ is an *s-t* cut.

11.3 Solving Maximum Flow

Greedily adding flows on the original graph G leads to problems, but greedily adding flows on the residual graph G_f is optimal. This is because residual graph allows to cancel some part of the added flow in order to increase the unit routed.
The algorithm: (1) initialize $\boldsymbol{f} = 0$, (2) repeatly find an *s-t* flow $\tilde{\boldsymbol{f}}$ such that $-\boldsymbol{f} \le \tilde{\boldsymbol{f}} \le \boldsymbol{c} + \boldsymbol{f}$ and set $\boldsymbol{f} = \boldsymbol{f} + \tilde{\boldsymbol{f}}$.
Property:

1. Assume $\boldsymbol{f}$ is feasible in G. Then $\tilde{\boldsymbol{f}}$ is feasible in $G_{\boldsymbol{f}} \Leftrightarrow \tilde{\boldsymbol{f}} + \boldsymbol{f}$ is feasible in G. Proof follows from definition.
2. A feasible $\boldsymbol{f}$ is optimal iff there is no feasible *s-t* flow in $G_{\boldsymbol{f}}$. Proof by contradiction.

Ford-Fulkerson Algorithm

We call the minimum capacity of all edges in an *s-t* flow to be the bottleneck capacity.
Algorithm: find an arbitrage *s-t* path flow in G_f, augment it to route the bottleneck, then add it to the current flow, repeatly.
For irrational capacities this algorithm may not terminate. For integer capacities (rational capacities can be translated to integer capacities to multiplication), each round must increase the capacity of current flow by at least 1, so it terminates in F^*) augmentations, which is $O(mF^*)$) time.
Modified algorithm (may be faster in some cases): find the *s-t* path flow with the maximum bottleneck capacity, then add it to the current flow, repeatly.
Using binary search on the threshold of bottleneck capacity (only use edges with capacity greater than the threshold), we can find the maximum bottleneck capacity in $O(m\log n)$. This path flow carries at least $\frac{1}{m}$ fraction of the remaining flows in G_f, as there are at most m path flows. Therefore, it terminates when $(1 - \frac{1}{m})^T F^* < 1$, which means $T = O(m\log F^*)$. The total time is $O(Tm\log n) = O(m^2 \log n \log F^*)$.

11.4 Duality of Max Flow and Min Cut

- **Max flow** $\le$ **Min cut**: For any feasible *s-t* flow and any *s-t* cut, we have $\mathbf{val}(f) \le c_G(S)$. To see this, simply observe that this flow must cross the cut, so the maximum value that can be routed is bounded by the maximum capacity allowed by the cut. In particular, the maximum *s-t* flow is bounded by the minimum *s-t* cut.
- **Max flow** $\ge$ **Min cut**: let $\boldsymbol{f}$ be the maximum flow composed of only *s-t* path flows, then t is not reachable from s in $G_{\boldsymbol{f}}$. Define S to be the vertex set that is reachable from s. Then $\boldsymbol{f}$ saturates every edge in $E \cap \{S \times V \setminus S\}$. There is no edge in $\boldsymbol{f}$ that is directed from $V \setminus S$ to S, as there is no edge from S to $V \setminus S$ in $G_{\boldsymbol{f}}$. This implies $\mathbf{val}(\boldsymbol{f}) \ge c_G(S)$, and thus the maximum flow is greater than the minimum cut.

Combining this two, we estabilish that strong duality between the maximum flow and the minimum cut holds, *i.e.*, max flow = min cut.

12 Dinic's Algorithm for Maximum Flow

Definition:

1. **Level of a vertex**: given a source vertex s, the level of a vertex u is defined to be the length of the shortest path from s to u.
2. **Admissible Edges**: an edge (u,v) is called admissible if $l(u)+1 = l(v)$, *i.e.*, it is in one of the shortest paths from s to v.
3. **Level Graph**: the level graph of G is the subgraph induced by only the admissible edges, *i.e.*, only keep edges relevant to the shortest paths. Inferring the level graph is in $O(m)$.
4. **Blocking flow**: a blocking flow in G is a feasible flow in the level graph of G such that (1) only uses admissible edges, and (2) saturates at least one edge for any s-t path in the level graph of G, *i.e.*, any s-t path in the level graph is blocked by such a flow.

Dinic's algorithm: starting from an empty flow, then repeatly augment the current flow by a blocking flow in the residual graph G_f until no more s-t path exists in G_f.

12.1 #Iterations of the Dinic's Algorithm

At each iteration, the target vertex's level in the residual graph is increased by at least 1, as the original shortest path is blocked. As the level of any vertex is at most n, Dinic's algorithm terminates in $O(n)$ iterations.

For unit-weight graphs, Dinic's algorithm can be proven to terminate in $O(\min\{m^{1/2}, n^{2/3}\})$ iterations. This is because now after k iterations, the next iteration would erase at least k edges, as the level of the target vertex is now at least k in the residual graph. (1) This implies the value of the blocking flow cannot exceed m/k, which implies termination after at most another m/k iterations. Setting $k = m^{1/2}$ gives the first bound. (2) By pigeonhole theorem, since the level graph has at most $n-1$ vertices, there are strictly more than $k/2$ of the level sets that has #vertices less than $2n/k$. Using pigeonhole theorem again, there is at least two adjacent level set such that both have #vertices less than $2n/k$. This implies there are at most $4n^2/k^2$ crossing edges between these two levels, and thus the algorithm terminates after at most another $4n^2/k^2$ iterations. Setting $k = 2n^{2/3}$ gives the second bound.

12.2 Finding Blocking Flow by Depth-First Seach

Using depth-first search in the level graph, we are able to find a blocking flow in $O(nm)$, thus the total complexity of Dinic's algorithm is $O(n^2m)$. In the unit-weight graph, depth-first seach is in $O(m)$, thus the total complexity is $O(m\min\{m^{1/2}, n^{2/3}\})$.

13 Link-Cut Trees

Definition:

1. **Dynamic Graph**: a graph that is constantly changing by edge insertion/deletion. No vertex changes.
2. **Dynamic rooted forest**: for every edge change, the graph remains a directed forest, and each tree in the forest has a single root. The root can be reached from any vertex in this tree.

A link-cut tree is a data structure that speeds up dynamic rooted forest changes, *i.e.*, it always represents a uniquely determined dynamic rooted forest, but can execute edge changes in less amortized time. This can be used to speed up the process of finding blocking flows, thus making the Dinic's algorithm faster. **Note**: the link-cut tree is designed to carry weight on its vertices but not edges. However, any edge-weighted graph can be converted to be vertex-weighted, by adding a dummy vertex in the middle of each edge with the same weight, and setting the weight of all original vertices to be $+\infty$. We choose $+\infty$ so that this does not change the max flow. Other values may be chosen for other usages.

The link-cut tree supports the following operations:

1. *Initialize(G)*: creates a link-cut tree that refers to an empty dynamic rooted forest with the same vertices of G but no edges, *i.e.*, every vertex is its own root.
2. *FindRoot(v)*: find the root of v in the current dynamic rooted forest.
3. *AddCost(v, Δ)*: add Δ to the cost of every vertex on the path from v to its root.
4. *FindMin(v)*: returns the first min-cost vertex on the path from v to its root and its associated cost.
5. *Link(u, v)*: add an edge (u,v), assuming u to be a root vertex and v to be in another tree. Note that the required property maintains the graph to be rooted forest and merges two trees into one.
6. *Cut(u, v)*: cuts a current edge (u,v). This splits one tree into two, with u being one of the root.

Theorem: The link-cut tree can realize any sequence of m operations in total expected time $O(m\log^2 n + n)$.

13.1 Implementation of Link-Cut Trees

The implementation relies on the treap structure (basically search property of binary search trees for one key + heap order for another independent key). Basically, we first construct these operations restricted to path trees, encoded by balanced treaps. The path is encoded such that one key (the searching key) of the treap stores the "order", *i.e.*, v is always at the right of u if u is the ancestor of v, and the other key (the heap key) stores a random value for constructing balanced binary trees with high probability.

The weight changes are boosted by associating the difference between the min-cost of current vertex and the min-cost of its parent, and the difference between the cost of current vertex and the min-cost of it. We first call PCut and Plink, if necessary, to make v have no precessor. When the current vertex has no precessor, the PathAddCost only needs to adjust the root's min-cost, and the PFindMin only needs to follow the child with $\Delta\min = 0$. As the depth is $O(\log n)$, these operations are $O(\log n)$ as well.

To implement the general link-cut tree, we decompose each tree into paths so that each vertex only occur in exactly one path and each internal vertex has exactly one incoming edge. By swtiching between the different path decompositions (requires $O(\log n)$), we are able to make sure the tree under changing is always a path. This is possible because all these operations actually only changes a specific path.

13.2 Boosting Blocking Flows by Link-Cut Trees

First, as described before, we convert the level graph of current residual graph G_f into vertex-weighted by adding dummy vertices. The change is that we now use the operations provided by the link-cut tree to do the DFS, which is faster than the naive DFS.

14 The Cut-Matching Game

Definition:

1. **Sparsity of a vertex subset**: Given $\emptyset \subset S \subset V$, the conductance $\psi(S) := \frac{|E(S,V\setminus S)|}{\min\{|S|,|V\setminus S|\}}$. This is different to the conductance $\phi(S)$ in the denominator. Since $\mathbf{vol}\,S \geq |S|$, it is guaranteed that $\psi(S) \geq \phi(S)$.
2. **Sparsity of a graph**: $\psi(G) := \min_{\emptyset\subset S\subset V} \psi(S)$. We say G is a ψ-expander w.r.t. sparsity iff $\psi(G) \geq \psi$. The cut that achieves the minimum is called the sparsest cut.

The cut-matching game is an algorithm that involves interaction of the cut player and the matching player, designed to follow a specific strategy, so that the result of such a game could certfify the sparsity of a graph.

14.1 Certifying via Embedding

Given graphs H and G defined on the same vertex set, we say a function is an embedding of H into G if it maps each edge $(u,v) \in H$ to a u-to-v path in G. We define the congestion of such an embedding to be the maximum number of times that any edge in G appears on any embedding path.

Property: given a ψ-expander graph H and an embedding of H into G with congestion C, then G is a ψ/C-expander.

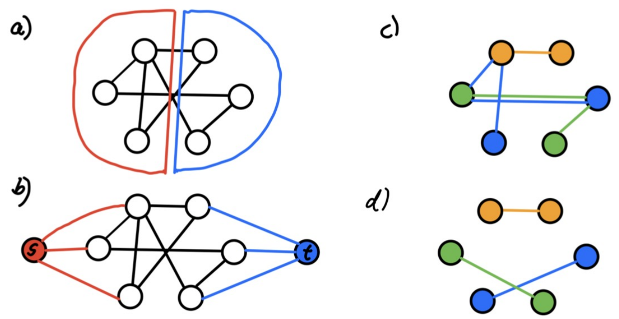
Proof. For any cut $(S, V\setminus S)$ such that $|S| \leq n/2$, we have $|E_H(S,V\setminus S)| \geq \psi|S|$. For every edge (u,v) in $|E_H(S,V\setminus S)|$, there is a path from u to v in G which crosses the cut. Since each edge crossing the cut can be used at most C times, we have that $|E_G(S,V\setminus S)| \geq \psi|S|/C$, which implies that G is a ψ/C-expander. □

14.2 Certifying ψ-expanders via Max Flows

Theorem 14.1.1. There is an algorithm CertifyOrCut(G, ψ) that given a graph G and a parameter $0 < \psi \le 1$, either:

- Certifies that G is a $\Omega\left(\psi/\log^2 n\right)$-expander w.r.t. sparsity.
- Presents a cut S such that $\psi(S) \le O(\psi)$.

The algorithm runs in time $O\left(\log^2 n\right) \cdot T_{\text{max\_flow}}(G) + \tilde{O}(m)$ where $T_{\text{max-flow}}(G)$ is the time it takes to solve a Max Flow problem on G.
Illustration of one iteration of the Algorithm:



In a), a bi-partition $(S_i, \overline{S_i})$ of V is found (requires random walk on G). In b), the bi-partition is used to obtain a flow problem where we inject one unit of flow to each vertex in S_i via super-source s and extract one unit of flow from each vertex in $\overline{S_i}$ via super-sink t. Every edge is set to have capacity $1/\psi$. Then we solve this problem to get a flow f with $\mathbf{val}(f) = n/2$. If such flow does not exist, then we return the min-cut of this flow problem, removing the dummy source and sink. c) If such flow exists, we construct a path flow decomposition. For each path, the first vertex is in S and the last vertex in $\overline{S}$. d) We find M_i to be the one-to-one matching between endpoints in S and $\overline{S}$ defined by the path flows.
It can be proven that after $T = \Theta(\log^2 n)$ iterations, the union of the T matchings is a 1/2-expander and G can be embedded into H with congestion $O(\log^2 n/\psi)$, which certifies that G is a $O(\psi/\log^2 n)$-expander. If one of the iterations presents a cut, then it can be proven that the sparsity of this cut is $O(\psi)$.

15 Interior Point Method for Max Flow: Barrier Method

Undirected max flow: Given a source vertex and a sink vertex, maximize F such that $\boldsymbol{B}\boldsymbol{f} = F(-\mathbf{1}_s + \mathbf{1}_t)$ and $-\boldsymbol{c} \le \boldsymbol{f} \le \boldsymbol{c}$.
Core idea: convert the inequality constraints to the object function via logrithm, as the logrithm only takes positive inputs. Then, if the object has K-stable Hessians, we can quickly solve it using Newton's method.
For the undirected max flow, we define the barrier function: $V(\boldsymbol{f}) = \sum_e -\log(\boldsymbol{c}(e) - \boldsymbol{f}(e)) - \log(\boldsymbol{c}(e) + \boldsymbol{f}(e))$. Given a constant $0 \le \alpha < 1$ and the value for the max flow F^*, we define the barrier problem to be minimizing $V(\boldsymbol{f})$ such that $B\boldsymbol{f} = \alpha F^*(-\mathbf{1}_s + \mathbf{1}_t)$. The KKT condition for the barrier problem is: (1) primal feasibility, *i.e.*, $B\boldsymbol{f} = \alpha F^*(-\mathbf{1}_s + \mathbf{1}_t)$ and $-\boldsymbol{c} \le \boldsymbol{f} \le \boldsymbol{c}$, (2) gradient condition, *i.e.*, $\nabla V(\boldsymbol{f}) = B^\top \boldsymbol{x}$ for the multiplier $\boldsymbol{x}$.
Let $\boldsymbol{f}_\alpha$ and $\boldsymbol{x}_\alpha$ be the optima of the barrier problem. The interior point method is based on the fact that given $\boldsymbol{f}_\alpha$ and $\boldsymbol{x}_\alpha$, we can quickly compute the optimal flow at $\alpha + \alpha'$ for some $\alpha' < 1 - \alpha$ using Newton's method. By repeatly doing this, we can get the optima for $\alpha > 1 - \epsilon$, thus a good approximation of the max flow. If the approximation is good enough for interger-weighted graphs, then we get the exact solution. Note that we can binary search for F^* in log time.

15.1 Updates using Divergence

The update problem is to minimize $V(\boldsymbol{f} + \boldsymbol{\delta})$ such that $B\boldsymbol{\delta} = \alpha' F^*(-\mathbf{1}_s + \mathbf{1}_t)$. Actually, as $V(\boldsymbol{\delta} + \boldsymbol{f}^*_\alpha) - (V(\boldsymbol{f}^*_\alpha) + \langle \nabla V(\boldsymbol{f}^*_\alpha), \boldsymbol{\delta}\rangle) = V(\boldsymbol{\delta} + \boldsymbol{f}^*_\alpha) - (V(\boldsymbol{f}^*_\alpha) + \langle \boldsymbol{x}^*_\alpha, \alpha' F^* \boldsymbol{b}_{st}\rangle)$ by the KKT gradient condition, the **divergence update problem** to minimize $V(\boldsymbol{\delta} + \boldsymbol{f}) - (V(\boldsymbol{f}) + \langle \nabla V(\boldsymbol{f}), \boldsymbol{\delta}\rangle)$ such that $\boldsymbol{B}\boldsymbol{\delta} = \alpha' F^* \boldsymbol{b}_{st}$ has the same optima as the original update problem. The divergence update problem is easier to analyze as we remove the first order information.

15.2 Fast Divergence Updates via Smoothing

Theorem: Suppose $S \subseteq \mathbb{R}^n$ is a convex set, and let $f, g : S \to \mathbb{R}$ be convex functions. Let $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in S} f(\boldsymbol{x})$. Suppose f, g agree on a neighborhood of $\boldsymbol{x}^*$ in S (i.e. an open set containing $\boldsymbol{x}^*$). Then $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in S} g(\boldsymbol{x})$. The proof is straightforward as local optimum is global for convex functions.

Smoothing is to find another (smoothed) convex function which agrees with the original convex function in the neighborhood of the optima, but with K-stable Hessian. Note that our divergence function obtains minimum at zero (here we do not care about constraints). Therefore, we can choose the smoothed function to be the second Taylor expansion at some point for large values, and use the divergence of the log function $-\log(1-x) - x$ for the neighborhood of zero. Then Newton's method (with linear constraints) can solve this in $\tilde{O}(m)$ time.

16 Distance Oracles

Distance oracle is a data structure that quickly computes the shortest path between all vertex pairs in an undirected graph. While the Floyd-Warshall algorithm achieves this in $O(n^3)$ preprocessing time, $O(n^2)$ space and $O(1)$ query time, distance oracles are much faster and use less space in the cost of accuracy.

Theorem: There is an algorithm that, for any integer $k \ge 1$ and undirected graph $G = (V, E)$, computes a data structure that can be stored using $\tilde{O}\left(kn^{1+1/k}\right)$ space, and returns in $O(k)$ time a distance estimate $\widetilde{\text{dist}}(u, v)$ on querying any two vertices $u, v \in V$ such that $\text{dist}(u, v) \le \widetilde{\text{dist}}(u, v) \le (2k-1) \cdot \text{dist}(u, v)$. The algorithm computes the data structure in expected time $\tilde{O}\left(kmn^{1/k}\right)$.

Note: when $k = 1$, this is the same performance as using Floyd-Warshall to preprocess and a matrix to store the distance pairs.

Key Idea behind the Distance Oracle ($k = 2$): sample a small subset of vertices S with probability $n^{-1/2}$ for each vertex, then compute the shortest path from all $u \in S$ to all the vertices $v \in V$ and store $(u, v, d(u, v))$. For the vertex u that are not in the selection, (1) find the min-distance selected vertex $p(u) \in S$ and store the matching $(u, p(u))$, (2) find the vertex set $\{v \mid d(u, v) < d(u, p(u))\}$ and store $(u, v, d(u, v))$. Therefore, the stored distances are exactly the shortest. For the rest distance pairs (u, v) where $u \notin S$ and $v \notin S$, return an estimation $\tilde{d}(u, v) = d(u, p(u)) + d(p(u), v)$. By triangular inequality, we have $d(p(u), v) \le d(u, p(u)) + d(u, v)$, thus $\tilde{d}(u, v) \le 2d(u, p(u)) + d(u, v) \le 3d(u, v)$.

To generalize, we simply sequentially sample from the previous selection with probability $n^{-1/k}$, with the first selection being the whole set.