ETH Zurich

Advanced Graph Algorithms and Optimization

Rasmus Kyng & Maximilian Probst Gutenberg

Spring 2024

These notes will be updated throughout the course. They are likely to contain typos, and they may have mistakes or lack clarity in places. Feedback and comments are welcome. Please send to kyng@inf.ethz.ch or, even better, submit a pull request at

`https://github.com/rjkyng/agao24_script`.

We want to thank scribes from the 2020 edition of the course who contributed to these notes: Hongjie Chen, Meher Chaitanya, Timon Knigge, and Tim Taubner – and we're grateful to all the readers who've submitted corrections, including Martin Kucera, Alejandro Cassis, and Luke Volpatti.

A important note: If you're a student browsing these notes to decide whether to take this course, please note that the current notes are incomplete. We will release parts later in the semester. You can take a look at last year's notes for an impression of the what the rest of the course will look like. Find them here:

`https://github.com/rjkyng/agao23_script/raw/main/agao23_script.pdf`

There will, however, be some changes to the content compared to last year.

Contents

Chapter 1

Course Introduction

1.1 Overview

This course will take us quite deep into modern approaches to graph algorithms using convex optimization techniques. By studying convex optimization through the lens of graph algorithms, we'll try to develop an understanding of fundamental phenomena in optimization. Much of our time will be devoted to flow problems on graphs. We will not only be studying these problems for their own sake, but also because they often provide a useful setting for thinking more broadly about optimization.

The course will cover some traditional discrete approaches to various graph problems, especially flow problems, and then contrast these approaches with modern, asymptotically faster methods based on combining convex optimization with spectral and combinatorial graph theory.

1.2 Electrical Flows and Voltages - a Graph Problem from Middle School?

We will dive right into graph problems by considering how electrical current moves through a network of resistors.

First, let us recall some middle school physics. If some of these things don't make sense to you, don't worry, in less than a paragraph from here, we'll be back to safely doing math.

Recall that a typical battery that one buys from Migros has two endpoints, and produces what is called a *voltage difference* between these endpoints.

One end of the battery will have a positive charge (I think that means an excess of positrons[1]), and the other a negative charge. If we connect the two endpoints with a wire, then a current will flow from one end of the battery to the other in an attempt to even out this imbalance of charge.

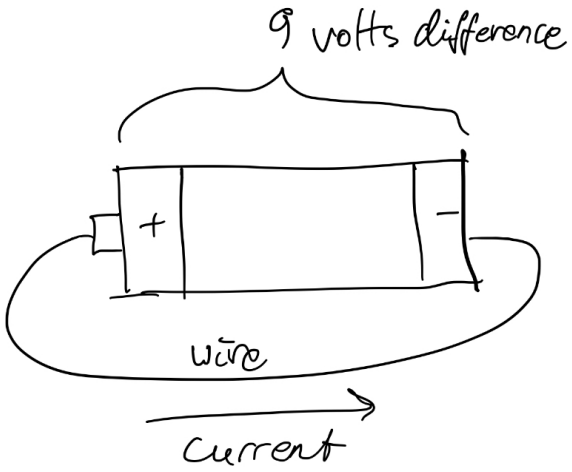


Figure 1.1: A 9 volts battery with a wire attached.

We can also imagine a kind of battery that tries to send a certain amount of current through the wires between its endpoints, e.g. 1 unit of charge per unit of time. This will be a little more convenient to work with, so let us focus on that case.

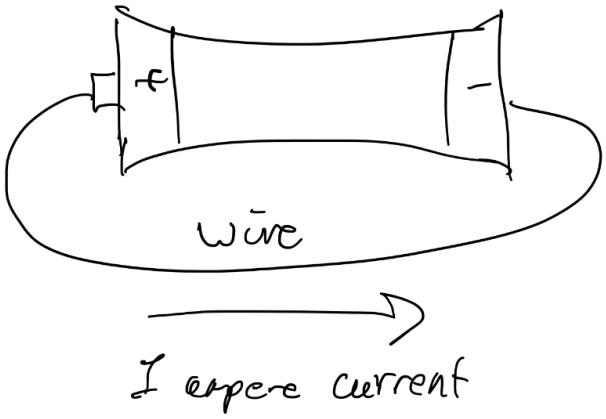


Figure 1.2: A 1 ampere battery with a wire attached.

A *resistor* is a piece of wire that connects two points u and v, and is completely described by a single number r called its *resistance*.

[1] I'm joking, of course! Try Wikipedia if you want to know more. However, you will not need it for this class.

If the voltage difference between the endpoints of the resistor is x, and the resistance is r then this will create a flow of charge per unit of time of $f = x/r$. This is called Ohm's Law.

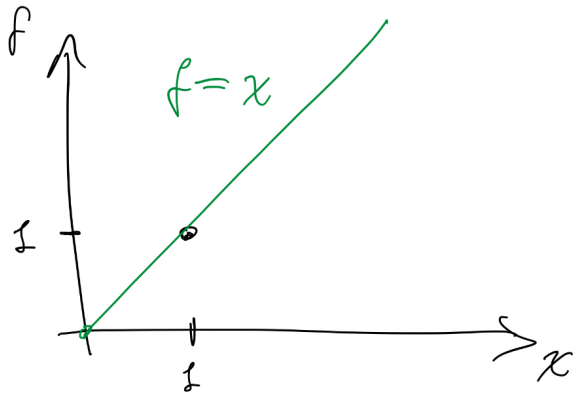


Figure 1.3: Ohm's Law for a resistor with resistance $r = 1$.

Suppose we set up a bunch of wires that route electricity from our current source s to our current sink t in some pattern:

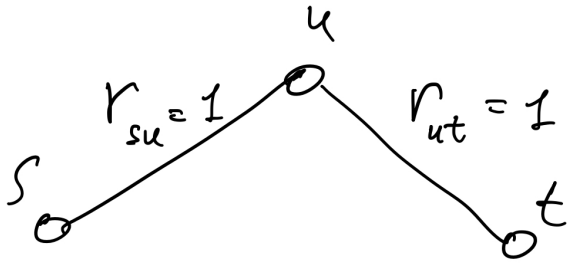


Figure 1.4: A path of two resistors.

We have one unit of charge flowing out of s per unit of time, and one unit coming into t. Because charge is conserved, the current flowing into any other point u must equal the amount flowing out of it. This is called Kirchhoff's Current Law.

To send one unit of current from s to t, we must be sending it first from s to u and then from u to t. So the current on edge (s, u) is 1 and the current on (u, t) is 1. By Ohm's Law, the voltage difference must also be 1 across each of the two wires. Thus, if the voltage is x at s, it must be $x + 1$ at u and $x + 2$ at t. What is x? It turns out it doesn't matter: We only care about the differences. So let us set $x = 0$.

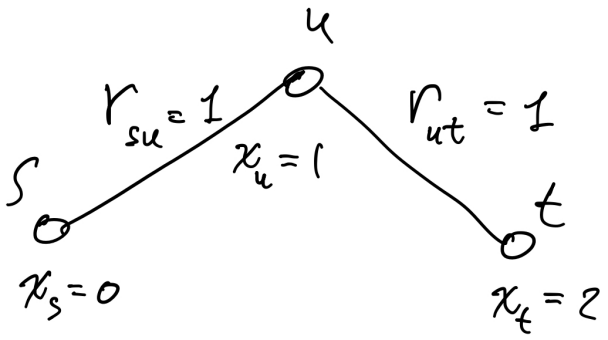


Figure 1.5: A path of two resistors.

Let us try one more example:

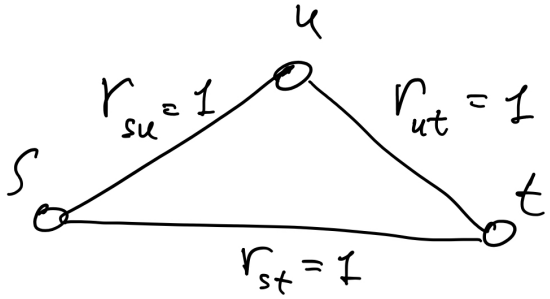


Figure 1.6: A network with three resistors.

How much flow will go directly from s to t and how much via u?

Well, we know what the net current flowing into and out of each vertex must be, and we can use that to set up some equations. Let us say the voltage at s is x_s, at u is x_u and at t is x_t.

- Net current at s: $-1 = (x_s - x_t) + (x_s - x_u)$
- Net current at u: $0 = (x_u - x_s) + (x_u - x_t)$
- Net current at t: $1 = (x_t - x_s) + (x_t - x_u)$

The following is a solution: $x_s = 0$, $x_u = \frac{1}{3}$, $x_t = \frac{2}{3}$. And as before, we can shift all the voltages by some constant x and get another solution $x_s = x + 0$, $x_u = x + \frac{1}{3}$, $x_t = x + \frac{2}{3}$. You might want to convince yourself that these are the only solutions.

Electrical flows in general graphs. Do we know enough to calculate the electrical flow in some other network of resistors? To answer this, let us think about the network as a graph. Consider an undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, and let us assume G is connected. Let's associate a resistance $\boldsymbol{r}(e) > 0$ with every edge $e \in E$.

To keep track of the direction of the flow on each edge, it will be useful to assign an arbitrary direction to every edge. So let's do that, but remember that this is just a bookkeeping tool that helps us track where flow is going.

A *flow* in the graph is a vector $\boldsymbol{f} : \mathbb{R}^E$. The *net flow* of $\boldsymbol{f}$ at a vertex $u \in V$ is defined as $\sum_{v \to u} \boldsymbol{f}(v,u) - \sum_{u \to v} \boldsymbol{f}(u,v)$.

We say a flow routes the demands $\boldsymbol{d} \in \mathbb{R}^V$ if the net flow at every vertex v is $\boldsymbol{d}(v)$.

We can assign a voltage to every vertex $\boldsymbol{x} \in R^V$. Ohm's Law says that the electrical flow induced by these voltages will be $\boldsymbol{f}(u,v) = \frac{1}{\boldsymbol{r}(u,v)}(\boldsymbol{x}(v) - \boldsymbol{x}(u))$.

Say we want to route one unit of current from vertex $s \in V$ to vertex $t \in V$. As before, we can write an equation for every vertex saying that the voltage differences must produce the desired net current:

- Net current at s: $-1 = \sum_{(s,v)} \frac{1}{\boldsymbol{r}(s,v)}(\boldsymbol{x}(v) - \boldsymbol{x}(s))$
- Net current at $u \in V \setminus \{s,t\}$: $0 = \sum_{(u,v)} \frac{1}{\boldsymbol{r}(u,v)}(\boldsymbol{x}(v) - \boldsymbol{x}(u))$
- Net current at t: $1 = \sum_{(t,v)} \frac{1}{\boldsymbol{r}(t,v)}(\boldsymbol{x}(v) - \boldsymbol{x}(t))$

This gives us n constraints, exactly as many as we have voltage variables. However we have to be a little careful when trying to conclude that a solution exists, yielding voltages $\boldsymbol{x}$ that gives induce an electrical flow routing the desired demand.

You will prove in the exercises (Week 1, Exercise 3) that a solution $\boldsymbol{x}$ exists. The proof requires two important observations: Firstly that the graph is connected, and secondly that summed over all vertices, the net demand is zero, i.e. as much flow is coming into the network as is leaving it.

The incidence matrix and the Laplacian matrix. To have a more compact notation for net flow constraints, we also introduce the *edge-vertex incidence matrix* of the graph, $\boldsymbol{B} \in \mathbb{R}^{V \times E}$.

$$\boldsymbol{B}(v,e) = \begin{cases} 1 & \text{if } e = (u,v) \\ -1 & \text{if } e = (v,u) \\ 0 & \text{o.w.} \end{cases}$$

Now we can express the net flow constraint that $\boldsymbol{f}$ routes $\boldsymbol{d}$ by

$$\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}.$$

This is also called a conservation constraint. In our examples so far, we have $\boldsymbol{d}(s) = -1$, $\boldsymbol{d}(t) = 1$ and $\boldsymbol{d}(u) = 0$ for all $u \in V \setminus \{s,t\}$.

If we let $\boldsymbol{R} = \mathrm{diag}_{e \in E}\, \boldsymbol{r}(e)$ then Ohm's law tells us that $\boldsymbol{f} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x}$. Putting these observations together, we have $\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x} = \boldsymbol{d}$. The voltages $\boldsymbol{x}$ that induce $\boldsymbol{f}$ must solve this system of linear equations, and we can use that to compute both $\boldsymbol{x}$ and $\boldsymbol{f}$. It is exactly

the same linear equation as the one we considered earlier. We can show that for a connected graph, a solution $\boldsymbol{x}$ exists if and only if the flow into the graph equals the net flow out, which we can express as $\sum_v \boldsymbol{d}(v) = 0$ or $\mathbf{1}^\top \boldsymbol{d} = 0$. You will show this as part of Exercise 3. This also implies that an electrical flow routing $\boldsymbol{d}$ exists if and only if the net flow into the graph equals the net flow out, which we can express as $\mathbf{1}^\top \boldsymbol{d} = 0$.

The matrix $\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top$ is called the *Laplacian* of the graph and is usually denoted by $\boldsymbol{L}$.

An optimization problem in disguise. So far, we have looked at electrical voltages and flows as arising from a set of linear equations – and it might not be apparent that this has anything to do with optimization. But transporting current through a resistor requires energy, which will be dissipated as heat by the resistor (i.e. it will get hot!). If we send a current of f across a resistor with a potential drop of x, then the amount of energy spent per unit of time by the resistor will be $f \cdot x$. This is called Joule's Law. Applying Ohm's law to a resistor with resistance r, we can also express this energy per unit of time as $f \cdot x = x^2/r = r \cdot f^2$. Since we aren't bothering with units, we will even forget about time, and refer to these quantities as "energy", even though a physicist would call them "power".

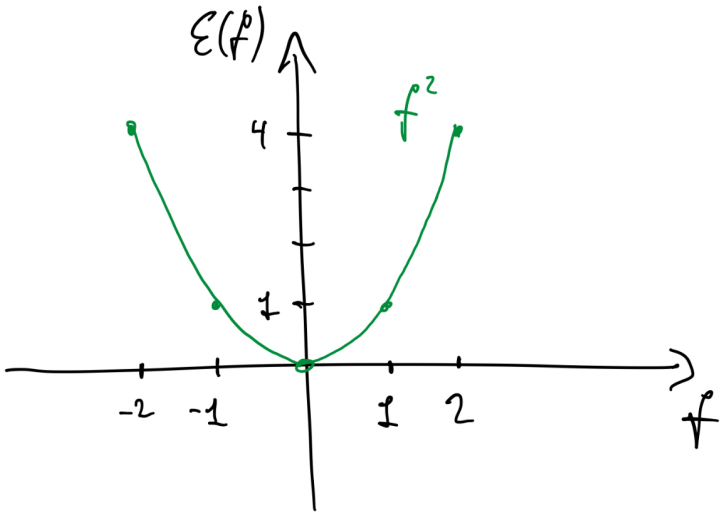


Figure 1.7: Energy as a function of flow in a resistor with resistance $r = 1$.

Now, another interesting question would seem to be: If we want to find a flow routing a certain demand $\boldsymbol{d}$, how should the flow behave in order to minimize the electrical energy spent routing the flow? The electrical energy of a flow vector $\boldsymbol{f}$ is $\mathcal{E}(\boldsymbol{f}) \stackrel{\text{def}}{=} \sum_e \boldsymbol{r}(e)\boldsymbol{f}(e)^2$. We can phrase this as an optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{f} \in \mathbb{R}^E} \mathcal{E}(\boldsymbol{f}) \\ &\text{s.t. } \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}. \end{aligned}$$

We call this problem *electrical energy-minimizing flow*. As we will prove later, the flow $\boldsymbol{f}^*$ that minimizes the electrical energy among all flows that satisfy $\boldsymbol{Bf} = \boldsymbol{d}$ is precisely the electrical flow.

A pair of problems. What about our voltages, can we also get them from some optimization problem? Well, we can work backwards from the fact that our voltages solve the equation $\boldsymbol{Lx} = \boldsymbol{d}$. Consider the function $c(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Lx} - \boldsymbol{x}^\top \boldsymbol{d}$. We should ask ourselves some questions about this function $c : \mathbb{R}^V \to \mathbb{R}$. Is it continuous and continuously differentiable? The answer to this is yes, and that is not hard to see. Does the function have a minimum? This is maybe not immediately clear, but the minimum does indeed exist.

When this is minimized, the derivative of $c(\boldsymbol{x})$ with respect to each coordinate of $\boldsymbol{x}$ must be zero. This condition yields exactly the system of linear equations $\boldsymbol{Lx} = \boldsymbol{d}$. You will confirm this in Exercise 4 of the first exercise sheet.

Based on our derivative condition for the optimum, we can also express the electrical voltages as the solution to an optimization problem, namely

$$\min_{\boldsymbol{x} \in \mathbb{R}^V} c(\boldsymbol{x})$$

As you are probably aware, having the derivative of each coordinate equal zero is not a sufficient condition for being at the optimum of a function[2].

It is also interesting to know whether *all* solutions to $\boldsymbol{Lx} = \boldsymbol{d}$ are in fact minimizers of c. The answer is yes, and we will see some very general tools for proving statements like this in Chapter 2.

Altogether, we can see that routing electrical current through a network of resistors leads to a *pair* of optimization problems, let's call them $\boldsymbol{f}^*$ and $\boldsymbol{x}^*$, and that the solutions to the two problems are related, in our case through the equation $\boldsymbol{f}^* = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x}^*$ (Ohm's Law). But why and how are these two optimization problems related?

Instead of minimizing $c(\boldsymbol{x})$, we can equivalently think about maximizing $-c(\boldsymbol{x})$, which gives the following optimization problem: $\max_{\boldsymbol{x} \in \mathbb{R}^V} -c(\boldsymbol{x})$. In fact, as you will show in the exercises for Week 1, we have $\mathcal{E}(\boldsymbol{f}^*) = -c(\boldsymbol{x}^*)$, so the minimum electrical energy is exactly the maximum value of $-\boldsymbol{c}(\boldsymbol{x})$. More generally for *any* flow that routes $\boldsymbol{d}$ and *any* voltages $\boldsymbol{x}$, we have $\mathcal{E}(\boldsymbol{f}) \geq -c(\boldsymbol{x})$. So, for any $\boldsymbol{x}$, the value of $-c(\boldsymbol{x})$ is a lower bound on the minimum energy $\mathcal{E}(\boldsymbol{f}^*)$.

This turns out to be an instance of a much broader phenomenon, known as Lagrangian duality, which allows us to learn a lot about many optimization problems by studying two related pairs of problems, a minimization problem, and a related maximization problem that gives lower bounds on the optimal value of the minimization problem.

[2]Consider the function in one variable $c(x) = x^3$.

Solving $\boldsymbol{Lx} = \boldsymbol{d}$. Given a graph G with resistances for the edges, and some net flow vector $\boldsymbol{d}$, how quickly can we compute $\boldsymbol{x}$? Broadly speaking, there are two very different families of algorithms we could use to try to solve this problem.

We could solve the linear equation using something like *Gaussian Elimination* to compute an exact solution.

Alternatively, we could start with a guess at a solution, e.g. $\boldsymbol{x}_0 = \mathbf{0}$, and then we could try to make a change to $\boldsymbol{x}_0$ to reach a new point $\boldsymbol{x}_1$ with a lower value of $c(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{d}$, i.e. $c(\boldsymbol{x}_1) < c(\boldsymbol{x}_0)$. If we repeat a process like that for enough steps, say t, hopefully we eventually reach $\boldsymbol{x}_t$ with $c(\boldsymbol{x}_t)$ close to $c(\boldsymbol{x}^*)$, where $\boldsymbol{x}^*$ is a minimizer of $c(\boldsymbol{x})$ and hence $\boldsymbol{L}\boldsymbol{x}^* = \boldsymbol{d}$. Now, we also need to make sure that $c(\boldsymbol{x}_t) \approx c(\boldsymbol{x}^*)$ implies that $\boldsymbol{L}\boldsymbol{x}_t \approx \boldsymbol{d}$ in some useful sense.

One of the most basic algorithms in this framework of "guess and adjust" is called *Gradient Descent*, which we will study in Week 2. The rough idea is the following: if we make a very small step from $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{\delta}$, then a multivariate Taylor expansion suggests that $c(\boldsymbol{x} + \boldsymbol{\delta}) - c(\boldsymbol{x}) \approx \sum_{v \in V} \boldsymbol{\delta}(v) \frac{\partial c(\boldsymbol{x})}{\partial \boldsymbol{x}(v)}$.

If we are dealing with smooth convex function, this quantity is negative if we let $\boldsymbol{\delta}(v) = -\epsilon \cdot \frac{\partial c(\boldsymbol{x})}{\partial \boldsymbol{x}(v)}$ for some small enough ϵ so the approximation holds well. So we should be able to make progress by taking a small step in this direction. That's Gradient Descent! The name comes from the vector of partial derivatives, which is called the gradient.

As we will see later in this course, understanding electrical problems from an optimization perspective is crucial to develop fast algorithms for computing electrical flows and voltages, but to do very well, we also need to borrow some ideas from Gaussian Elimination.

What running times do different approaches get?

1. Using Gaussian Elimination, we can find $\boldsymbol{x}$ s.t. $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$ in $O(n^3)$ time and with asymptotically faster algorithms based on matrix multiplication, we can bring this down to roughly $O(n^{2.372})$.

2. Meanwhile Gradient Descent will get a running time of $O(n^3 m)$ or so – at least this is a what a simple analysis suggests.

3. However, we can do much better: By combining ideas from both algorithms, and a bit more, we can get $\boldsymbol{x}$ up to very high accuracy in time $O(m \log^c n)$ where c is some small constant.

1.3 Convex Optimization

Recall our plot in Figure 1.7 of the energy required to route a flow f across a resistor with resistance r, which was $\mathcal{E}(f) = r \cdot f^2$. We see that the function has a special structure: the

graph of the function sits below the line joining any two points $(f, \mathcal{E}(f))$ and $(g, \mathcal{E}(g))$. A function $\mathcal{E} : \mathbb{R} \to \mathbb{R}$ that has this property is said to be convex.

Figure 1.8 shows the energy as a function of flow, along with two points $(f, \mathcal{E}(f))$ and $(g, \mathcal{E}(g))$. We see the function sits below the line segment between these points.

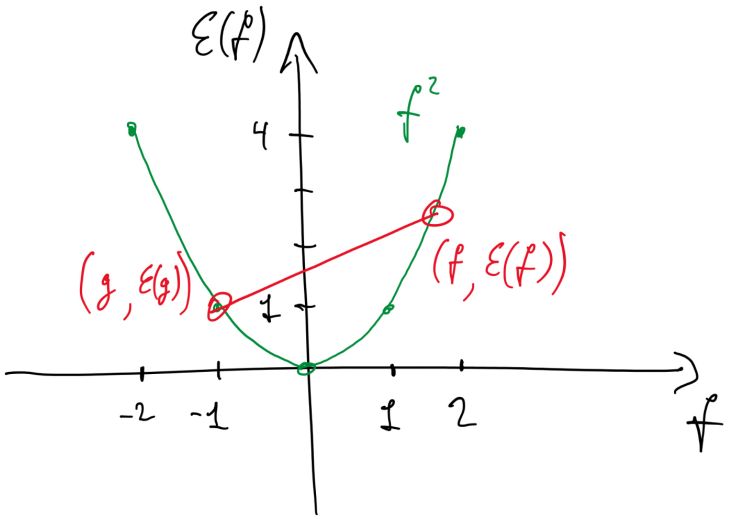


Figure 1.8: Energy as a function of flow in a resistor with resistance $r = 1$. The function is convex.

We can also interpret this condition as saying that for all $\theta \in [0, 1]$

$$\mathcal{E}(\theta f + (1 - \theta)g) \leq \theta \mathcal{E}(f) + (1 - \theta)\mathcal{E}(g).$$

This immediately generalizes to functions $\mathcal{E} : \mathbb{R}^m \to \mathbb{R}$.

A *convex set* is a subset of $S \subseteq \mathbb{R}^m$ s.t. if $\boldsymbol{f}, \boldsymbol{g} \in S$ then for all $\theta \in [0, 1]$ we have $\theta \boldsymbol{f} + (1-\theta)\boldsymbol{g} \in S$.

Figure 1.9 shows some examples of sets that are and aren't convex.

Convex functions and convex sets are central to optimization, because for most problems of minimization of a convex function over a convex set, we can develop fast algorithms [3].

So why convex functions and convex sets? One important reason is that for a convex function defined over a convex feasible set, any local minimum is also a global minimum, and this fact makes searching for an optimal solution computationally easier. In fact, this is closely related to why Gradient Descent works well on many convex functions.

Notice that the set $\{\boldsymbol{f} : \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}\}$ is convex, i.e. the set of all flows that route a fixed demand

[3] There are some convex optimization problems that are NP-hard. That said, polynomial time algorithms exist for almost any convex problem you can come up with. The most general polynomial time algorithm for convex optimization is probably the Ellipsoid Method.

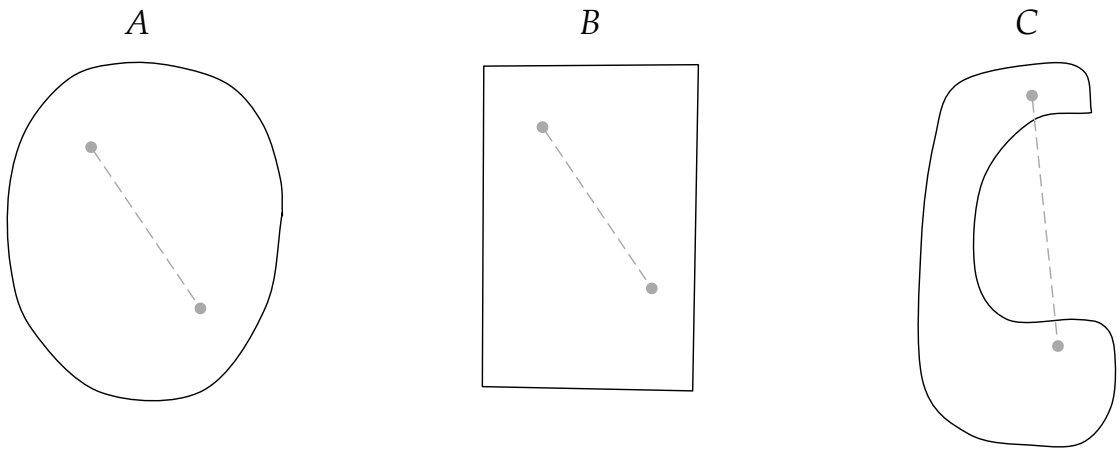


Figure 1.9: A depiction of convex and non-convex sets. The sets A and B are convex since the straight line between any two points inside them is also in the set. The set C is not convex.

$\boldsymbol{d}$ is convex. It is also easy to verify that $\mathcal{E}(\boldsymbol{f}) = \sum_e \boldsymbol{r}(e)\boldsymbol{f}(e)^2$ is a convex function, and hence finding an electrical flow is an instance of convex minmization:

1.4 More Graph Optimization Problems

Maximum flow. Again, let $G = (V, E)$ be an undirected, connected graph with n vertices and m edges. Suppose we want to find a flow $\boldsymbol{f} \in \mathbb{R}^E$ that routes $\boldsymbol{d}$, but instead of trying to minimize electrical energy, we try to pick an $\boldsymbol{f}$ that minimizes the largest amount of flow on any edge, i.e. $\max_e |\boldsymbol{f}_e|$ – which we also denote by $\|\boldsymbol{f}\|_\infty$. We can write this problem as

$$\begin{aligned} &\min_{\boldsymbol{f} \in \mathbb{R}^E} \|\boldsymbol{f}\|_\infty \\ &\text{s.t. } \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d} \end{aligned}$$

This problem is known as the Minimum Congested Flow Problem[4]. It is equivalent to the more famous Maximum Flow Problem.

The behavior of this kind of flow is very different than electrical flow. Consider the question of whether a certain demand can be routed $\|\boldsymbol{f}\|_\infty \leq 1$. Imagine sending goods from a source s to a destination t using a network of train lines that all have the same capacity and asking whether the network is able to route the goods at the rate you want: This boils down to whether routing exists with $\|\boldsymbol{f}\|_\infty \leq 1$, if we set it up right.

We have a very fast, convex optimization-based algorithm for Minimum Congested Flow: In $m\epsilon^{-1}\log^{O(1)} n$ time, we can find a flow $\tilde{\boldsymbol{f}}$ s.t. $\boldsymbol{B}\tilde{\boldsymbol{f}} = \boldsymbol{d}$ and $\left\|\tilde{\boldsymbol{f}}\right\|_\infty \leq (1+\epsilon)\|\boldsymbol{f}^*\|_\infty$, where $\boldsymbol{f}^*$

[4]This version is called undirected, because the graph is undirected, and *uncapacitated* because we are aiming for the same bound on the flow on all edges.

is an optimal solution, i.e. an actual minimum congestion flow routing $\boldsymbol{d}$.

But what if we want ϵ to be very small, e.g. $1/m$? Then this running time isn't so good anymore. But, in this case, we can use other algorithms that find flow $\boldsymbol{f}^*$ *exactly*. Unfortunately, these algorithms take time roughly $m^{4/3+o(1)}$.

Just as the electrical flow problem had a dual voltage problem, so maximum flow has a dual voltage problem, which is know as the s-t minimum cut problem.

Maximum flow, with directions and capacities. We can make the maximum flow problem harder by introducing directed edges: To do so, we allow edges to exist in both directions between vertices, and we require that the flow on a directed edge is always non-negative. So now $G = (V, E)$ is a directed graph. We can also make the problem harder by introducing capacities. We define a capacity vector $\boldsymbol{c} \in \mathbb{R}^E \geq \mathbf{0}$ and try to minimize $\left\| \boldsymbol{C}^{-1}\boldsymbol{f} \right\|_\infty$, where $\boldsymbol{C} = \operatorname{diag}_{e \in E} \boldsymbol{c}(e)$. Then our problem becomes

$$\begin{aligned} \min_{\boldsymbol{f} \in \mathbb{R}^E} & \left\| \boldsymbol{C}^{-1}\boldsymbol{f} \right\|_\infty \\ \text{s.t. } & \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d} \\ & \boldsymbol{f} \geq \mathbf{0}. \end{aligned}$$

For this capacitated, directed maximum flow problem, our best algorithms run in about $O(m\sqrt{n})$ time in sparse graphs and $O(m^{1.483})$ in dense graphs[5], even if we are willing to accept fairly low accuracy solution. If the capacities are allowed to be exponentially large, the best running time we can get is $O(mn)$. For this problem, we do not yet know how to improve over classical combinatorial algorithms using convex optimization.

Multi-commodity flow. We can make the problem even harder still, by simultaneously trying to route two types of flow (imagine pipes with Coke and Pepsi). Our problem now looks like

$$\begin{aligned} \min_{\boldsymbol{f}_1, \boldsymbol{f}_2 \in \mathbb{R}^E} & \left\| \boldsymbol{C}^{-1}(\boldsymbol{f}_1 + \boldsymbol{f}_2) \right\|_\infty \\ \text{s.t. } & \boldsymbol{B}\boldsymbol{f}_1 = \boldsymbol{d}_1 \\ & \boldsymbol{B}\boldsymbol{f}_2 = \boldsymbol{d}_2 \\ & \boldsymbol{f}_1, \boldsymbol{f}_2 \geq \mathbf{0}. \end{aligned}$$

Solving this problem to high accuracy is essentially as hard as solving a general linear program! We should see later in the course how to make this statement precise.

If we in the above problem additionally require that our flows must be integer valued, i.e. $\boldsymbol{f}_1, \boldsymbol{f}_2 \in \mathbb{N}_0$, then the problem becomes NP-complete.

[5]Provided the capacities are integers satisfying a condition like $\boldsymbol{c} \leq n^{100}\mathbf{1}$.

Random walks in a graph. Google famously uses[6] the PageRank problem to help decide how to rank their search results. This problem essentially boils down to computing the *stable distribution* of a random walk on a graph. Suppose $G = (V, E)$ is a directed graph where each outgoing edge (v, u), which we will define as going from u to v, has a transition probability $p_{(v,u)} > 0$ s.t. $\sum_{z \leftarrow u} p_{(z,u)} = 1$. We can take a step of a random walk on the vertex set by starting at some vertex $u_0 = u$, and then randomly pick one of the outgoing edges (v, u) with probability $p_{(v,u)}$ and move to the chosen vertex $u_1 = v$. Repeating this procedure, to take a step from the next vertex u_1, gives us a *random walk* in the graph, a sequence of vertices $u_0, u_1, u_2 \ldots, u_k$.

We let $\boldsymbol{P} \in \mathbb{R}^{V \times V}$ be the matrix of transition probabilities given by

$$\boldsymbol{P}_{vu} = \begin{cases} p_{(v,u)} & \text{for } (u, v) \in E \\ 0 & \text{o.w.} \end{cases}$$

Any probability distribution over the vertices can be specified by a vector $\boldsymbol{p} \in \mathbb{R}^V$ where $\boldsymbol{p} \geq \boldsymbol{0}$ and $\sum_v \boldsymbol{p}(v) = 1$. We say that probability distribution $\boldsymbol{\pi}$ on the vertices is a *stable distribution* of the random walk if $\boldsymbol{\pi} = \boldsymbol{P}\boldsymbol{\pi}$. A strongly connected graph always has exactly one stable distribution.

How quickly can we compute the stable distribution of a general random walk? Under some mild conditions on the stable distribution[7], we can find a high accuracy approximation of $\boldsymbol{\pi}$ in time $O(m \log^c n)$ for some constant c.

This problem does not easily fit in a framework of convex optimization, but nonetheless, our fastest algorithms for it use ideas from convex optimization.

Topics in this Course

In this course, we will try to address the following questions.

1. What are the fundamental tools of fast convex optimization?
2. What are some problems we can solve quickly on graphs using optimization?
3. What can graphs teach us about convex optimization?
4. What algorithm design techniques are good for getting algorithms that quickly find a crude approximate solution? And what techniques are best when we need to get a highly accurate answer?
5. What is special about flow problems?

[6] At least they did at some point.

[7] Roughly something like $\max_v 1/\boldsymbol{\pi}(v) \leq n^{100}$.

Part I

Introduction to Convex Optimization

Chapter 2

Some Basic Optimization, Convex Geometry, and Linear Algebra

2.1 Overview

In this chapter, we will

1. Start with an overview (i.e. this list).
2. Learn some basic terminology and facts about optimization.
3. Recall our definition of convex functions and see how convex functions can also be understood in terms of a characterization based on first derivatives.
4. See how the first derivatives of a convex function can certify that we are at a global minimum.

2.2 Optimization Problems

Focusing for now on optimization over $\boldsymbol{x} \in \mathbb{R}^n$, we usually write optimization problems as:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \text{ (or } \max) \; & f(\boldsymbol{x}) \\ s.t. \; & g_1(\boldsymbol{x}) \leq b_1 \\ & . \\ & . \\ & . \\ & g_m(\boldsymbol{x}) \leq b_m \end{aligned}$$

where $\{g_i(\boldsymbol{x})\}_{i=1}^m$ encode the constraints. For example, in the following optimization problem from the previous chapter

$$\min_{\boldsymbol{f} \in \mathbb{R}^E} \sum_e \boldsymbol{r}(e)\boldsymbol{f}(e)^2$$
$$\text{s.t. } \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}$$

we have the constraint $\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}$. Notice that we can rewrite this constraint as $\boldsymbol{B}\boldsymbol{f} \leq \boldsymbol{d}$ and $-\boldsymbol{B}\boldsymbol{f} \leq -\boldsymbol{d}$ to match the above setting. The set of points which respect the constraints is called the *feasible set*.

Definition 2.2.1. For a given optimization problem the set $\mathcal{F} = \{\boldsymbol{x} \in \mathbb{R}^n \ : \ g_i(\boldsymbol{x}) \leq b_i, \forall i \in [m]\}$ is called the **feasible set**. A point $\boldsymbol{x} \in \mathcal{F}$ is called a **feasible point**, and a point $\boldsymbol{x}' \notin \mathcal{F}$ is called an **infeasible point**.

Ideally, we would like to find optimal solutions for the optimization problems we consider. Let's define what we mean exactly.

Definition 2.2.2. For a *maximization* problem $\boldsymbol{x}^\star$ is called an **optimal solution** if $f(\boldsymbol{x}^\star) \geq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{F}$. Similarly, for a *minimization* problem $\boldsymbol{x}^\star$ is an optimal solution if $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{F}$.

What happens if there are *no feasible points*? In this case, an optimal solution cannot exist, and we say the problem is infeasible.

Definition 2.2.3. If $\mathcal{F} = \emptyset$ we say that the optimization problem is **infeasible**. If $\mathcal{F} \neq \emptyset$ we say the optimization problem is **feasible**.

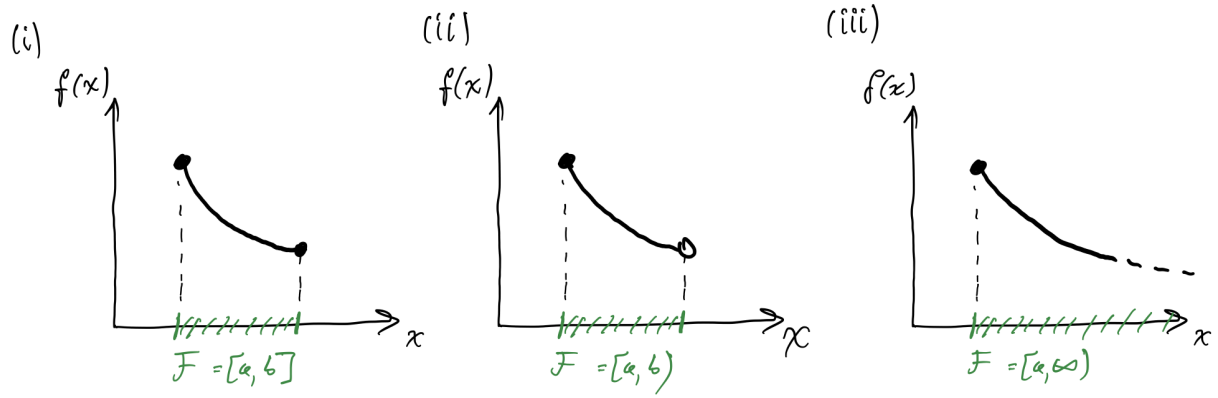


Figure 2.1

Consider three examples depicted in Figure 2.1:

(i) $\mathcal{F} = [a, b]$

(ii) $\mathcal{F} = [a, b)$

(iii) $\mathcal{F} = [a, \infty)$

In the first example, the minimum of the function is attained at b. In the second case the region is open and therefore there is no minimum function value, since for every point we will choose, there will always be another point with a smaller function value. Lastly, in the third example, the region is unbounded and the function decreasing, thus again there will always be another point with a smaller function value.

Sufficient Condition for Optimality. The following theorem, which is a fundamental theorem in real analysis, gives us a sufficient (though not necessary) condition for optimality.

Theorem (Extreme Value Theorem)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function and $\mathcal{F} \subseteq \mathbb{R}^n$ be nonempty, bounded, and closed. Then, the optimization problem $\min f(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{F}$ has an optimal solution.

2.3 A Characterization of Convex Functions

Recall the definitions of convex sets and convex functions that we introduced in Chapter 1:

Definition 2.3.1. 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 $\theta \in [0, 1]$.

Definition 2.3.2. 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:

$$f\left(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}\right) \leq \theta f(\boldsymbol{x}) + \Big(1-\theta\Big) f(\boldsymbol{y}).$$

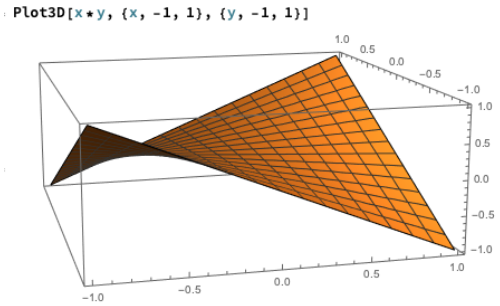


Figure 2.2: This plot shows the function $f(x, y) = xy$. For any fixed y_0, the function $h(x) = f(x, y_0) = xy_0$ is linear in x, and so is a convex function in x. But is f convex?

We will first give an important characterization of convex function. To do so, we need to characterize multivariate functions via their Taylor expansion.

Notation for this section. In the rest of this section, we frequently consider a multivariate function f whose domain is a set $S \subseteq \mathbb{R}^n$, which we will require to be open. When we additionally require that S is convex, we will specify this. Note that $S = \mathbb{R}^n$ is both open and convex and it suffices to keep this case in mind. Things sometimes get more complicated if S is not open, e.g. when the domain of f has a boundary. We will leave those complications for another time.

2.3.1 First-order Taylor Approximation

Definition 2.3.3. The **gradient** of a function $f : S \to \mathbb{R}$ at point $\boldsymbol{x} \in S$ is denoted $\boldsymbol{\nabla} f(\boldsymbol{x})$ and is defined as

$$\boldsymbol{\nabla} f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}(1)}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}(n)}\right]^\top$$

First-order Taylor expansion. For a function $f : \mathbb{R} \to \mathbb{R}$ of a single variable, differentiable at $x \in \mathbb{R}$

$$f(x + \delta) = f(x) + f'(x)\delta + o(|\delta|)$$

where by definition:

$$\lim_{\delta \to 0} \frac{o(|\delta|)}{|\delta|} = 0.$$

Similarly, a multivariate function $f : S \to \mathbb{R}$ is said to be *(Fréchet) differentiable* at $\boldsymbol{x} \in S$ when there exists $\boldsymbol{\nabla} f(\boldsymbol{x}) \in \mathbb{R}^n$ s.t.

$$\lim_{\boldsymbol{\delta} \to \mathbf{0}} \frac{\left\| f(\boldsymbol{x} + \boldsymbol{\delta}) - f(\boldsymbol{x}) - \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{\delta} \right\|_2}{\|\boldsymbol{\delta}\|_2} = 0.$$

Note that this is equivalent to saying that $f(\boldsymbol{x} + \boldsymbol{\delta}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)$.

We say that f is *continuously differentiable* on a set $S \subseteq \mathbb{R}^n$ if it is differentiable and in addition the gradient is continuous on S. A differentiable convex function whose domain is an open convex set $S \subseteq \mathbb{R}^n$ is always continuously differentiable[1].

Remark. In this course, we will generally err on the side of being informal about functional analysis when we can afford to, and we will not worry too much about the details of different notions of differentiability (e.g. Fréchet and Gateaux differentiability), except when it turns out to be important.

Theorem 2.3.4 (Taylor's Theorem, multivariate first-order remainder form)**.** *If $f : S \to \mathbb{R}$ is continuously differentiable over $[\boldsymbol{x}, \boldsymbol{y}]$, then for some $\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]$,*

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{z})^\top (\boldsymbol{y} - \boldsymbol{x}).$$

[1] See p. 248, Corollary 25.5.1 in *Convex Analysis* by Rockafellar (my version is the Second print, 1972). Rockefellar's corollary concerns finite convex functions, because he otherwise allows convex functions that may take on the values $\pm\infty$.

This theorem is useful for showing that the function f can be approximated by the affine function $\boldsymbol{y} \to f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$ when $\boldsymbol{y}$ is "close to" $\boldsymbol{x}$ in some sense.

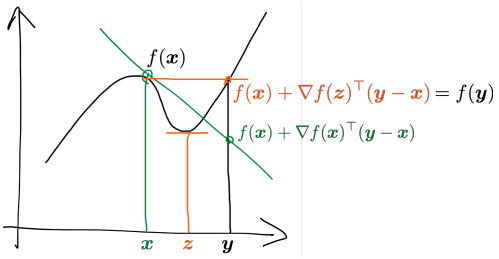


Figure 2.3: The function $f(\boldsymbol{y})$ equals its linear approximation based at x, with gradient coming from an intermediate point z, i.e. $f(\boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{z})^\top (\boldsymbol{y} - \boldsymbol{x})$.

2.3.2 Directional Derivatives

Definition 2.3.5. Let $f : S \to \mathbb{R}$ be a function differentiable at $\boldsymbol{x} \in S$ and let us consider $\boldsymbol{d} \in \mathbb{R}^n$. We define the **derivative of f at $\boldsymbol{x}$ in direction $\boldsymbol{d}$** as:

$$Df(\boldsymbol{x})[\boldsymbol{d}] = \lim_{\lambda \to \boldsymbol{0}} \frac{f(\boldsymbol{x} + \lambda \boldsymbol{d}) - f(\boldsymbol{x})}{\lambda}$$

Proposition 2.3.6. $Df(\boldsymbol{x})[\boldsymbol{d}] = \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{d}$.

Proof. Using the first order expansion of f at $\boldsymbol{x}$:

$$f(\boldsymbol{x} + \lambda \boldsymbol{d}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\lambda \boldsymbol{d}) + o(\|\lambda \boldsymbol{d}\|_2)$$

hence, dividing by λ (and noticing that $\|\lambda \boldsymbol{d}\|_2 = \lambda \|\boldsymbol{d}\|_2$):

$$\frac{f(\boldsymbol{x} + \lambda \boldsymbol{d}) - f(\boldsymbol{x})}{\lambda} = \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{d} + \frac{o(\lambda \|\boldsymbol{d}\|_2)}{\lambda}$$

letting λ go to 0 concludes the proof. □

2.3.3 Lower Bounding Convex Functions with Affine Functions

In order to prove the characterization of convex functions in the next section we will need the following lemma. This lemma says that any differentiable convex function can be lower bounded by an affine function.

Theorem 2.3.7. *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 $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$.*

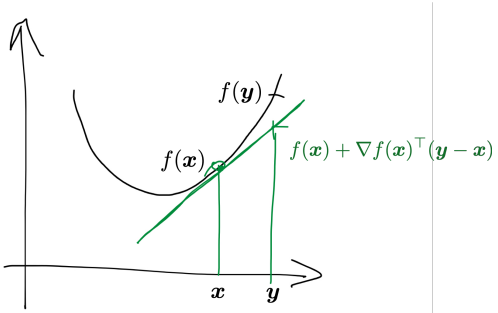


Figure 2.4: The convex function $f(\boldsymbol{y})$ sits above the linear function in $\boldsymbol{y}$ given by $f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x})$.

Proof. [$\implies$] Assume f is convex, then for all $\mathbf{x}, \mathbf{y} \in S$ and $\theta \in [0, 1]$, if we let $\mathbf{z} = \theta\boldsymbol{y} + (1 - \theta)\boldsymbol{x}$, we have that

$$f(\mathbf{z}) = f((1 - \theta)\boldsymbol{x} + \theta\boldsymbol{y}) \leq (1 - \theta)f(\boldsymbol{x}) + \theta f(\boldsymbol{y})$$

and therefore by subtracting $f(\boldsymbol{x})$ from both sides we get:

$$\begin{aligned} f\left(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})\right) - f(\boldsymbol{x}) &\leq \theta f(\boldsymbol{y}) + (1 - \theta)f(\boldsymbol{x}) - f(\boldsymbol{x}) \\ &= \theta f(\boldsymbol{y}) - \theta f(\boldsymbol{x}). \end{aligned}$$

Thus we get that (for $\theta > 0$):

$$\frac{f\left(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})\right) - f(\boldsymbol{x})}{\theta} \leq f(\boldsymbol{y}) - f(\boldsymbol{x})$$

Applying Proposition 2.3.6 with $\boldsymbol{d} = \boldsymbol{y} - \boldsymbol{x}$ we have that:

$$\boldsymbol{\nabla} f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x}) = \lim_{\theta \to 0^+} \frac{f(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})) - f(\boldsymbol{x})}{\theta} \leq f(\boldsymbol{y}) - f(\boldsymbol{x}).$$

[$\impliedby$] Assume that $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x})$ for all $\mathbf{x}, \mathbf{y} \in S$ and show that f is convex. Let $\mathbf{x}, \mathbf{y} \in S$ and $\mathbf{z} = \theta\boldsymbol{y} + (1 - \theta)\boldsymbol{x}$. By our assumption we have that:

$$f(\boldsymbol{y}) \geq f(\boldsymbol{z}) + \boldsymbol{\nabla} f(\boldsymbol{z})^\top(\boldsymbol{y} - \mathbf{z}) \tag{2.1}$$

$$f(\boldsymbol{x}) \geq f(\boldsymbol{z}) + \boldsymbol{\nabla} f(\boldsymbol{z})^\top(\boldsymbol{x} - \mathbf{z}) \tag{2.2}$$

Observe that $\boldsymbol{y} - \boldsymbol{z} = (1 - \theta)(\boldsymbol{y} - \boldsymbol{x})$ and $\boldsymbol{x} - \boldsymbol{z} = \theta(\boldsymbol{x} - \boldsymbol{y})$. Thus adding θ times (2.1) to $(1 - \theta)$ times (2.2) gives cancellation of the vectors multiplying the gradient, yielding

$$\begin{aligned} \theta f(\boldsymbol{y}) + (1 - \theta)f(\boldsymbol{x}) &\geq f(\boldsymbol{z}) + \boldsymbol{\nabla} f(\boldsymbol{z})^\top \mathbf{0} \\ &= f(\theta\boldsymbol{y} + (1 - \theta)\boldsymbol{x}) \end{aligned}$$

This is exactly the definition of convexity. □

2.4 Conditions for Optimality

We now want to find necessary and sufficient conditions for local optimality.

Definition 2.4.1. Consider a differentiable function $f : S \to \mathbb{R}$. A point $\boldsymbol{x} \in S$ at which $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$ is called a **stationary point**.

Proposition 2.4.2. *If $\boldsymbol{x}$ is a local extremum of a differentiable function $f : S \to \mathbb{R}$ then $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$.*

Proof. Let us assume that $\boldsymbol{x}$ is a local minimum for f. Then for all $\boldsymbol{d} \in \mathbb{R}^n$, $f(\boldsymbol{x}) \leq f(\boldsymbol{x}+\lambda \boldsymbol{d})$ for λ small enough. Hence:

$$0 \leq f(\boldsymbol{x} + \lambda \boldsymbol{d}) - f(\boldsymbol{x}) = \lambda \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{d} + o(\|\lambda \boldsymbol{d}\|)$$

dividing by $\lambda > 0$ and letting $\lambda \to 0^+$, we obtain $0 \leq \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{d}$. But, taking $\boldsymbol{d} = -\boldsymbol{\nabla} f(\boldsymbol{x})$, we get $0 \leq -\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_2^2$. This implies that $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$.

The case where $\boldsymbol{x}$ is a local maximum can be dealt with similarly. □

Remark 2.4.3. For this proposition to hold, it is important that S is open.

For convex functions however it turns out that a stationary point necessarily implies that the function is at its minimum. Together with the proposition above, this says that for a convex function on $\mathbb{R}^n$ a point is optimal if and only if it is stationary.

Proposition 2.4.4. *Let $S \subseteq \mathbb{R}^n$ be an open convex set and let $f : S \to \mathbb{R}$ be a differentiable and convex function. If $\boldsymbol{x}$ is a stationary point then $\boldsymbol{x}$ is a global minimum.*

Proof. From Theorem 2.3.7 we know that for all $\boldsymbol{x}, \boldsymbol{y} \in S$: $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x})$. Since $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$ this implies that $f(\boldsymbol{y}) \geq f(\boldsymbol{x})$. As this holds for any $\boldsymbol{y} \in S$, $\boldsymbol{x}$ is a global minimum.

□

Chapter 3

Convexity, Second Derivatives and Gradient Descent

Notation for this chapter. In this chapter, we sometimes consider a multivariate function f whose domain is a set $S \subseteq \mathbb{R}^n$, which we will require to be open. When we additionally require that S is convex, we will specify this. Note that $S = \mathbb{R}^n$ is both open and convex and it suffices to keep this case in mind. Things sometimes get more complicated if S is not open, e.g. when the domain of f has a boundary. We will leave those complications for another time.

3.1 A Review of Linear Algebra

Semi-definiteness of a matrix. The following classification of symmetric matrices will be useful.

Definition 3.1.1. Let $\boldsymbol{A}$ by a symmetric matrix in $\mathbb{R}^{n\times n}$. We say that $\boldsymbol{A}$ is:

1. *positive definite* iff $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$ for all $x \in \mathbb{R}^n \setminus \{0\}$;
2. *positive semidefinite* iff $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$ for all $x \in \mathbb{R}^n$;
3. If neither $\boldsymbol{A}$ nor $-\boldsymbol{A}$ is positive semi-definite, we say that $\boldsymbol{A}$ is *indefinite*.

Example: indefinite matrix. Consider the following matrix $\boldsymbol{A}$:

$$\boldsymbol{A} := \begin{bmatrix} +4 & -1 \\ -1 & -2 \end{bmatrix}$$

For $\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, we have $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 4 > 0$. For $\boldsymbol{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ we have $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = -2 < 0$. $\boldsymbol{A}$ is therefore indefinite.

The following theorem gives a useful characterization of (semi)definite matrices.

Theorem 3.1.2. *Let $\boldsymbol{A}$ be a symmetric matrix in $\mathbb{R}^{n\times n}$.*

1. *$\boldsymbol{A}$ is positive definite iff all its eigenvalues are positive;*
2. *$\boldsymbol{A}$ is positive semidefinite iff all its eigenvalues are non-negative;*

In order to prove this theorem, let us first recall the Spectral Theorem for symmetric matrices.

Theorem 3.1.3 (The Spectral Theorem for Symmetric Matrices)**.** *For all symmetric $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ there exist $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ s.t.*

1. *$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$.*
2. *$\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}$ (the $n \times n$ identity matrix). I.e. the columns of $\boldsymbol{V}$ form an orthonormal basis. Furthermore, $\boldsymbol{v}_i$ is an eigenvector of $\lambda_i(\boldsymbol{A})$, the ith eigenvalue of $\boldsymbol{A}$.*
3. *$\boldsymbol{\Lambda}_{ii} = \lambda_i(\boldsymbol{A})$.*

Using the Spectral Theorem, we can show the following result:

Theorem 3.1.4 (The Courant-Fischer Theorem)**.** *Let $\boldsymbol{A}$ be a symmetric matrix in $\mathbb{R}^{n\times n}$, with eigenvalues $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$. Then*

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

Theorem 3.1.2 is an immediate corollary of Theorem 3.1.4, since we can see that the minimum value of the quadratic form $\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x}$ over $\boldsymbol{x} \in W = \mathbb{R}^n$ is $\lambda_1(\boldsymbol{A})\|\boldsymbol{x}\|_2^2$.

Proof of Theorem 3.1.4. We start by showing Part 1.

Consider letting $W = \operatorname{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_i\}$, and normalize $\boldsymbol{x} \in W$ so that $\|\boldsymbol{x}\|_2 = 1$. Then $\boldsymbol{x} = \sum_{j=1}^{i} \boldsymbol{c}(j)\boldsymbol{v}_j$ for some vector $\boldsymbol{c} \in \mathbb{R}^i$ with $\|\boldsymbol{c}\|_2 = 1$.

Using the decomposition from Theorem 3.1.3 $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$ where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues of $\boldsymbol{A}$, which we take to be sorted in increasing order. Then $\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^\top\boldsymbol{V}^\top\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{x} = (\boldsymbol{V}\boldsymbol{x})^\top\boldsymbol{\Lambda}(\boldsymbol{V}\boldsymbol{x}) = \sum_{j=1}^{i}\lambda_j\boldsymbol{c}(j)^2 \le \lambda_i\|\boldsymbol{c}\|_2^2 = \lambda_i$. So this choice of W ensures the maximizer cannot achieve a value above λ_i.

But is it possible that the "minimizer" can do better by choosing a different W? Let $T = \text{span}\{\boldsymbol{v}_i, \ldots, \boldsymbol{v}_n\}$. As $\dim(T) = n+1-i$ and $\dim(W) = i$, we must have $\dim(W \cap T) \geq 1$, by a standard property of subspaces. Hence for any W of this dimension,

$$\begin{aligned}\max_{\boldsymbol{x} \in W, \boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} &\geq \max_{\boldsymbol{x} \in W \cap T, \boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} \\ &\geq \min_{\substack{\text{subspace } V \subseteq T \\ \dim(V)=1}} \max_{\boldsymbol{x} \in V, \boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \lambda_i,\end{aligned}$$

where the last equality follows from a similar calculation to our first one. Thus, λ_i can always be achieved by the "maximizer" for all W of this dimension.

Part 2 can be dealt with similarly.

□

Example: a positive semidefinite matrix. Consider the following matrix $\boldsymbol{A}$:

$$\boldsymbol{A} := \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

For $\boldsymbol{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, we have $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$, so $\lambda = 0$ is an eigenvalue of $\boldsymbol{A}$. For $\boldsymbol{x} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, we have $\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} 2 \\ -2 \end{pmatrix} = 2\boldsymbol{x}$, so $\lambda = 2$ is the other eigenvalue of $\boldsymbol{A}$. As both are non-negative, by the theorem above, $\boldsymbol{A}$ is positive semidefinite.

Since we are learning about symmetric matrices, there is one more fact that everyone should know about them. We'll use $\lambda_{\max}(\boldsymbol{A})$ denote maximum eigenvalue of a matrix $\boldsymbol{A}$, and $\lambda_{\min}(\boldsymbol{A})$ the minimum.

Claim 3.1.5. *For a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\|\boldsymbol{A}\| = \max(|\lambda_{\max}(\boldsymbol{A})|, |\lambda_{\min}(\boldsymbol{A})|)$.*

3.2 Characterizations of Convexity and Optimality via Second Derivatives

We will now use the second derivatives of a function to obtain characterizations of convexity and optimality. We will begin by introducing the *Hessian*, the matrix of pairwise second derivatives of a function. We will see that it plays a role in approximating a function via a second-order Taylor expansion. We will then use *semi-definiteness* of the Hessian matrix to characterize both conditions of optimality as well as the convexity of a function.

Definition 3.2.1. Given a function $f : S \to \mathbb{R}$ its **Hessian** matrix at point $\boldsymbol{x} \in S$ denoted $\boldsymbol{H}_f(\boldsymbol{x})$ (also sometimes denoted $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$) is:

$$\boldsymbol{H}_f(\boldsymbol{x}) := \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(1)^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(1)\partial \boldsymbol{x}(2)} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(1)\partial \boldsymbol{x}(n)} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(2)\partial \boldsymbol{x}(1)} & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(2)^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(2)\partial \boldsymbol{x}(n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(n)\partial \boldsymbol{x}(1)} & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(n)\partial \boldsymbol{x}(2)} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial \boldsymbol{x}(n)^2} \end{bmatrix}$$

Second-order Taylor expansion. When f is twice differentiable it is possible to obtain an approximation of f by quadratic functions. Our definition of $f : S \to \mathbb{R}$ being twice (Fréchet) differentiable at $\boldsymbol{x} \in S$ is that there exists $\boldsymbol{\nabla} f(\boldsymbol{x}) \in \mathbb{R}^n$ and $\boldsymbol{H}_f(\boldsymbol{x}) \in R^{n \times n}$ s.t.

$$\lim_{\boldsymbol{\delta} \to \boldsymbol{0}} \frac{\left\| f(\boldsymbol{x} + \boldsymbol{\delta}) - f(\boldsymbol{x}) - \left(\boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{\delta} \right) \right\|_2}{\|\boldsymbol{\delta}\|_2^2} = 0.$$

This is equivalent to saying that for all $\boldsymbol{\delta}$

$$f(\boldsymbol{x} + \boldsymbol{\delta}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2^2).$$

where by definition:

$$\lim_{\boldsymbol{\delta} \to \boldsymbol{0}} \frac{o(\|\boldsymbol{\delta}\|^2)}{\|\boldsymbol{\delta}\|_2^2} = 0$$

We say that f is *twice continuously differentiable* on a set $S \subseteq \mathbb{R}^n$ if it is twice differentiable and in addition the gradient and Hessian are continuous on S.

As for first order expansions, we have a Taylor's Theorem, which we state in the so-called remainder form.

Theorem 3.2.2 (Taylor's Theorem, multivariate second-order remainder form)**.** *If $f : S \to \mathbb{R}$ is twice continuously differentiable over $[\boldsymbol{x}, \boldsymbol{y}]$, then for some $\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]$,*

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2} (\boldsymbol{y} - \boldsymbol{x})^\top \boldsymbol{H}_f(\boldsymbol{z}) (\boldsymbol{y} - \boldsymbol{x})$$

3.2.1 A Necessary Condition for Local Extrema

Recall that in the previous chapter, we show the following proposition.

Proposition 3.2.3. *If $\boldsymbol{x}$ is a local extremum of a differentiable function $f : S \to \mathbb{R}$ then $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$.*

We can now give the second-order necessary conditions for local extrema via the Hessian.

Theorem 3.2.4. *Let $f : S \to \mathbb{R}$ be a function twice differentiable at $\boldsymbol{x} \in S$. If $\boldsymbol{x}$ is a local minimum, then $\boldsymbol{H}_f(\boldsymbol{x})$ is positive semidefinite.*

Proof. Let us assume that $\boldsymbol{x}$ is a local minimum. We know from Proposition 3.2.3 that $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$, hence the second-order expansion at $\boldsymbol{x}$ takes the form:

$$f(\boldsymbol{x} + \lambda \boldsymbol{d}) = f(\boldsymbol{x}) + \lambda^2 \frac{1}{2} \boldsymbol{d}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{d} + o(\lambda^2 \left\| \boldsymbol{d} \right\|_2^2)$$

Because $\boldsymbol{x}$ is a local minimum, we can then derive

$$0 \leq \lim_{\lambda \to 0^+} \frac{f(\boldsymbol{x} + \lambda \boldsymbol{d}) - f(\boldsymbol{x})}{\lambda^2} = \frac{1}{2} \boldsymbol{d}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{d}$$

This is true for any $\boldsymbol{d}$, hence $\boldsymbol{H}_f(\boldsymbol{x})$ is positive semidefinite. □

Remark 3.2.5. Again, for this proposition to hold, it is important that S is open.

3.2.2 A sufficient condition for local extrema

A local minimum thus is a stationary point and has a positive semi-definite Hessian. The converse is almost true, but we need to strengthen the Hessian condition slightly.

Theorem 3.2.6. *Let $f : S \to \mathbb{R}$ be a function twice differentiable at a stationary point $\boldsymbol{x} \in S$. If $\boldsymbol{H}_f(\boldsymbol{x})$ is positive definite then $\boldsymbol{x}$ is a local minimum.*

Proof. Let us assume that $\boldsymbol{H}_f(\boldsymbol{x})$ is positive definite. We know that $\boldsymbol{x}$ is a stationary point. We can write the second-order expansion at $\boldsymbol{x}$:

$$f(\boldsymbol{x} + \boldsymbol{\delta}) = f(\boldsymbol{x}) + \frac{1}{2} \boldsymbol{\delta}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{\delta} + o(\left\| \boldsymbol{\delta} \right\|_2^2)$$

Because the Hessian is positive definite, it has a strictly positive minimum eigenvalue $\lambda_{\min}$, we can conclude that $\boldsymbol{\delta}^\top \boldsymbol{H}_f(\boldsymbol{x}) \boldsymbol{\delta} \geq \lambda_{\min} \left\| \boldsymbol{\delta} \right\|_2^2$. From this, we conclude that when $\left\| \boldsymbol{\delta} \right\|_2^2$ is small enough, $f(\boldsymbol{x} + \boldsymbol{\delta}) - f(\boldsymbol{x}) \geq \frac{1}{4} \lambda_{\min} \left\| \boldsymbol{\delta} \right\|_2^2 > 0$. This proves that $\boldsymbol{x}$ is a local minimum. □

Remark 3.2.7. When $\boldsymbol{H}_f(\boldsymbol{x})$ is indefinite at a stationary point $\boldsymbol{x}$, we have what is known as a *saddle point*: $\boldsymbol{x}$ will be a minimum along the eigenvectors of $\boldsymbol{H}_f(\boldsymbol{x})$ for which the eigenvalues are positive and a maximum along the eigenvectors of $\boldsymbol{H}_f(\boldsymbol{x})$ for which the eigenvalues are negative.

3.2.3 Characterization of convexity

Definition 3.2.8. For a convex set $S \subseteq R^n$, we say that a function $f : S \to \mathbb{R}$ is **strictly convex on** S if for any two points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$ and any $\theta \in (0, 1)$ we have that:

$$f\left(\theta \boldsymbol{x}_1 + (1 - \theta) \boldsymbol{x}_2\right) < \theta f(\boldsymbol{x}_1) + \Big(1 - \theta\Big) f(\boldsymbol{x}_2).$$

Theorem 3.2.9. *Let $S \subseteq \mathbb{R}^n$ be open and convex, and let $f : S \to \mathbb{R}$ be twice continuously differentiable.*

1. *If $H_f(\boldsymbol{x})$ is positive semi-definite for any $\boldsymbol{x} \in S$ then 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.*
3. *If f is convex, then $H_f(\boldsymbol{x})$ is positive semi-definite $\forall \boldsymbol{x} \in S$.*

Proof.

1. By applying Theorem 3.2.2, we find that for some $\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]$:

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}\Big((\boldsymbol{y} - \boldsymbol{x})^\top H_f(\boldsymbol{z})(\boldsymbol{y} - \boldsymbol{x})\Big)$$

   If $H_f(\boldsymbol{z})$ is positive semi-definite, this necessarily implies that:

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$$

   and from Theorem 2.3.7 we get that f is convex.

2. if $H_f(\boldsymbol{x})$ is positive definite, we have that:

$$f(\boldsymbol{y}) > f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}).$$

   Applying the same idea as in Theorem 2.3.7 we can show that in this case f is **strictly** convex.

3. Let f be a convex function. For $\boldsymbol{x} \in S$, and some small $\lambda > 0$, for any $\mathbf{d} \in \mathbb{R}^n$ we have that $\boldsymbol{x} + \lambda \mathbf{d} \in S$. From the Taylor expansion of f we get:

$$f(\boldsymbol{x} + \lambda \mathbf{d}) = f(\boldsymbol{x}) + \lambda \boldsymbol{\nabla} f(\boldsymbol{x})^\top \mathbf{d} + \frac{\lambda^2}{2} \mathbf{d}^\top H_f(\boldsymbol{x}) \mathbf{d} + o(\lambda^2 \left\| \boldsymbol{d} \right\|_2^2).$$

   From Lemma 2.3.7 we get that if f is convex then:

$$f(\boldsymbol{x} + \lambda \mathbf{d}) \geq f(\boldsymbol{x}) + \lambda \boldsymbol{\nabla} f(\boldsymbol{x})^\top \mathbf{d}.$$

   Therefore, we have that for any $\mathbf{d} \in \mathbb{R}^n$:

$$\frac{\lambda^2}{2} \mathbf{d}^\top H_f(\boldsymbol{x}) \mathbf{d} + o(||\lambda \mathbf{d}||^2) \geq 0$$

   Dividing by λ^2 and taking $\lambda \to 0^+$ gives us that for any $\mathbf{d} \in \mathbb{R}^n$: $\mathbf{d}^\top H_f(\boldsymbol{x}) \mathbf{d} \geq 0$. □

Remark 3.2.10. It is important to note that if S is open and f is strictly convex, then $\boldsymbol{H}_f(\boldsymbol{x})$ may still (only) be positive semi-definite $\forall \boldsymbol{x} \in S$. Consider $f(x) = x^4$ which is strictly convex, then the Hessian is $\boldsymbol{H}_f(x) = 12x^2$ which equals 0 at $x = 0$.

3.3 Gradient Descent - An Approach to Optimization?

We have begun to develop an understanding of convex functions, and what we have learned already suggests a way for us to try to find an approximate minimizer of a given convex function.

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and we want to solve

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

We would like to find $\boldsymbol{x}^*$, a global minimizer of f. Suppose we start with some initial guess $\boldsymbol{x}_0$, and we want to update it to $\boldsymbol{x}_1$ with $f(\boldsymbol{x}_1) < f(\boldsymbol{x}_0)$. If we can repeatedly make updates like this, maybe we eventually find a point with nearly minimum function value, i.e. some $\tilde{\boldsymbol{x}}$ with $f(\tilde{\boldsymbol{x}}) \approx f(\boldsymbol{x}^*)$?

Recall that $f(\boldsymbol{x}_0 + \boldsymbol{\delta}) = f(\boldsymbol{x}_0) + \boldsymbol{\nabla} f(\boldsymbol{x}_0)^\top \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)$. This means that if we choose $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \lambda \boldsymbol{\nabla} f(\boldsymbol{x}_0)$, we get

$$f(\boldsymbol{x}_0 - \lambda \boldsymbol{\nabla} f(\boldsymbol{x}_0)) = f(\boldsymbol{x}_0) - \lambda \|\boldsymbol{\nabla} f(\boldsymbol{x}_0)\|_2^2 + o(\lambda \|\boldsymbol{\nabla} f(\boldsymbol{x}_0)\|_2)$$

And because f is convex, we know that $\boldsymbol{\nabla} f(\boldsymbol{x}_0) \neq \mathbf{0}$ unless we are already at a global minimum. So, for some small enough $\lambda > 0$, we should get $f(\boldsymbol{x}_1) < f(\boldsymbol{x}_0)$ unless we're already at a global minimizer. This idea of taking a step in the direction of $-\boldsymbol{\nabla} f(\boldsymbol{x}_0)$ is what is called *Gradient Descent*. But how do we choose λ each time? And does this lead to an algorithm that quickly reaches a point with close to minimal function value? To get good answers to these questions, we need to assume more about the function f that we are trying to minimize.

In the following subsection, we will see some conditions that suffice. But there are also many other settings where one can show that some form of gradient descent converges.

3.3.1 A Quantitative Bound on Changes in the Gradient

Definition 3.3.1. Let $f : S \to \mathbb{R}$ be a differentiable function, where $S \subseteq \mathbb{R}^n$ is convex and open. We say that f is *β-gradient Lipschitz* iff for all $\boldsymbol{x}, \boldsymbol{y} \in S$

$$\|\boldsymbol{\nabla} f(\boldsymbol{x}) - \boldsymbol{\nabla} f(\boldsymbol{y})\|_2 \leq \beta \|\boldsymbol{x} - \boldsymbol{y}\|_2 .$$

We also refer to this as f being *β-smooth*.

Proposition 3.3.2. *Consider a twice continuously differentiable $f : S \to \mathbb{R}$. Then f is β-gradient Lipschitz if and only if for all $\boldsymbol{x} \in S$, $\|\boldsymbol{H}_f(\boldsymbol{x})\| \leq \beta$.*

You will prove this in Exercise 13 (Week 2) of the first exercise set.

Proposition 3.3.3. *Let $f : S \to \mathbb{R}$ be a β-gradient Lipschitz function. Then for all $\boldsymbol{x}, \boldsymbol{y} \in S$,*

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \frac{\beta}{2} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2$$

To prove this proposition, we need the following result from multi-variate calculus. This is a restricted form of the fundamental theorem of calculus for line integrals.

Proposition 3.3.4. *Let $f : S \to \mathbb{R}$ be a differentiable function, and consider $\boldsymbol{x}, \boldsymbol{y}$ such that $[\boldsymbol{x}, \boldsymbol{y}] \in S$. Let $\boldsymbol{x}_\theta = \boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})$. Then*

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \int_{\theta=0}^{1} \boldsymbol{\nabla} f(\boldsymbol{x}_\theta)^\top (\boldsymbol{y} - \boldsymbol{x}) d\theta$$

Now, we're in a position to show Proposition 3.3.3

Proof of Proposition 3.3.3. Let $f : S \to \mathbb{R}$ be a β-gradient Lipschitz function. Consider arbitrary $\boldsymbol{x}, \boldsymbol{y} \in S$ such that $[\boldsymbol{x}, \boldsymbol{y}] \in S$

$$\begin{aligned}
f(\boldsymbol{y}) &= f(\boldsymbol{x}) + \int_{\theta=0}^{1} \boldsymbol{\nabla} f(\boldsymbol{x}_\theta)^\top (\boldsymbol{y} - \boldsymbol{x}) d\theta \\
&= f(\boldsymbol{x}) + \int_{\theta=0}^{1} \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) d\theta + \int_{\theta=0}^{1} (\boldsymbol{\nabla} f(\boldsymbol{x}_\theta) - \boldsymbol{\nabla} f(\boldsymbol{x}))^\top (\boldsymbol{y} - \boldsymbol{x}) d\theta \\
&\qquad \text{Next we use Cauchy-Schwarz pointwise.} \\
&\qquad \text{We also evaluate the first integral.} \\
&\leq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \int_{\theta=0}^{1} \|\boldsymbol{\nabla} f(\boldsymbol{x}_\theta) - \boldsymbol{\nabla} f(\boldsymbol{x})\| \, \|\boldsymbol{y} - \boldsymbol{x}\| \, d\theta \\
&\qquad \text{Then we apply } \beta\text{-gradient Lipschitz and note } \boldsymbol{x}_\theta - \boldsymbol{x} = \theta(\boldsymbol{y} - \boldsymbol{x}). \\
&\leq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \int_{\theta=0}^{1} \beta\theta \, \|\boldsymbol{y} - \boldsymbol{x}\|^2 \, d\theta. \\
&= f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \frac{\beta}{2} \|\boldsymbol{y} - \boldsymbol{x}\|^2 .
\end{aligned}$$

□

3.3.2 Analyzing Gradient Descent

It turns out that just knowing a function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and β-gradient Lipschitz is enough to let us figure out a reasonable step size for Gradient Descent and let us analyze its convergence.

We start at a point $\boldsymbol{x}_0 \in \mathbb{R}^n$, and we try to find a point $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{\delta}$ with lower function value. We will let our upper bound from Proposition 3.3.3 guide us, in fact, we could ask, what is the *best* update for minimizing this upper bound, i.e. a $\boldsymbol{\delta}$ solving

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^n} f(\boldsymbol{x}_0) + \boldsymbol{\nabla} f(\boldsymbol{x}_0)^\top \boldsymbol{\delta} + \frac{\beta}{2} \|\boldsymbol{\delta}\|^2$$

We can compute the best according to this upper bound by noting first that function is convex and continuously differentiable, and hence will be minimized at any point where the gradient is zero. Thus we want $\mathbf{0} = \boldsymbol{\nabla}_{\boldsymbol{\delta}} \left(f(\boldsymbol{x}_0) + \boldsymbol{\nabla} f(\boldsymbol{x}_0)^\top \boldsymbol{\delta} + \frac{\beta}{2} \|\boldsymbol{\delta}\|^2 \right) = \boldsymbol{\nabla} f(\boldsymbol{x}_0) + \beta \boldsymbol{\delta}$, which occurs at $\boldsymbol{\delta} = -\frac{1}{\beta} \boldsymbol{\nabla} f(\boldsymbol{x}_0)$.

Plugging in this value into the upper bound, we get that $f(\boldsymbol{x}_1) \leq f(\boldsymbol{x}_0) - \|\boldsymbol{\nabla} f(\boldsymbol{x}_0)\|_2^2 / 2\beta$.

Now, as our algorithm, we will start with some guess $\boldsymbol{x}_0$, and then at every step we will update our guess using the best step based on our Proposition 3.3.3 upper bound on f at $\boldsymbol{x}_i$, and so we get

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \frac{1}{\beta} \boldsymbol{\nabla} f(\boldsymbol{x}_i) \text{ and } f(\boldsymbol{x}_{i+1}) \leq f(\boldsymbol{x}_i) - \frac{\|\boldsymbol{\nabla} f(\boldsymbol{x}_i)\|_2^2}{2\beta}. \tag{3.1}$$

Let us try to prove that our algorithm converges toward an $\boldsymbol{x}$ with low function value.

We will measure this by looking at

$$\text{gap}_i = f(\boldsymbol{x}_i) - f(\boldsymbol{x}^*)$$

where $\boldsymbol{x}^*$ is a global minimizer of f (note that there may not be a unique minimizer of f). We will try to show that this function value gap grows small. Using $f(\boldsymbol{x}_{i+1}) - f(\boldsymbol{x}_i) = \text{gap}_{i+1} - \text{gap}_i$, we get

$$\text{gap}_{i+1} - \text{gap}_i \leq -\frac{\|\boldsymbol{\nabla} f(\boldsymbol{x}_i)\|_2^2}{2\beta} \tag{3.2}$$

If the gap_i value is never too much bigger than $\frac{\|\boldsymbol{\nabla} f(\boldsymbol{x}_i)\|_2^2}{2\beta}$, then this should help us show we are making progress. But how much can they differ? We will now try to show a limit on this.

Recall that in the previous chapter we showed the following theorem.

Theorem 3.3.5. *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 $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$.*

Using the convexity of f and the lower bound on convex functions given by Theorem 3.3.5, we have that

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}_i) + \boldsymbol{\nabla} f(\boldsymbol{x}_i)^\top (\boldsymbol{x}^* - \boldsymbol{x}_i) \tag{3.3}$$

Rearranging gets us

$$\begin{aligned} \text{gap}_i &\leq \boldsymbol{\nabla} f(\boldsymbol{x}_i)^\top (\boldsymbol{x}_i - \boldsymbol{x}^*) && (3.4) \\ &\leq \|\boldsymbol{\nabla} f(\boldsymbol{x}_i)\|_2 \|\boldsymbol{x}_i - \boldsymbol{x}^*\|_2 && \text{by Cauchy-Schwarz.} \end{aligned}$$

At this point, we are essentially ready to connect Equation (3.2) with Equation (3.4) and analyze the convergence rate of our algorithm.

However, at the moment, we see that the change $\text{gap}_{i+1} - \text{gap}_i$ in how close we are to the optimum function value is governed by the norm of the gradient $\|\boldsymbol{\nabla} f(\boldsymbol{x}_i)\|_2$, while the size of the gap is related to *both* this quantity and the distance $\|\boldsymbol{x}_i - \boldsymbol{x}^*\|_2$ between the current solution $\boldsymbol{x}_i$ and an optimum $\boldsymbol{x}^*$. Do we need both or can we get rid of, say, the distance? Unfortunately, with this algorithm and for this class of functions, a dependence on the distance is necessary. However, we can simplify things considerably using the following observation, which you will prove in the exercises (Exercise 2):

Claim 3.3.6. *When running Gradient Descent as given by the step in Equation* (3.1)*, for all* i $\|\boldsymbol{x}_i - \boldsymbol{x}^*\|_2 \leq \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2$.

Combining this Claim with Equation (3.2) and Equation (3.4),

$$\text{gap}_{i+1} - \text{gap}_i \leq -\frac{1}{2\beta} \cdot \left(\frac{\text{gap}_i}{\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2} \right)^2 \tag{3.5}$$

At this point, a simple induction will complete the proof of following result.

Theorem 3.3.7. *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* $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$ *be a minimizer of* f*. The Gradient Descent algorithm given by*

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \frac{1}{\beta} \boldsymbol{\nabla} f(\boldsymbol{x}_i)$$

ensures that the k*th iterate satisfies*

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

Carrying out this induction is one of the exercises in problem set 2.

Part II

Spectral Graph Theory

Chapter 4

Introduction to Spectral Graph Theory

In this chapter, we will study graphs through linear algebra. This approach is known as Spectral Graph Theory and turns out to be surprisingly powerful. An in-depth treatment of many topics in this area can be found in [Spi19].

4.1 Recap: Incidence and Adjacency Matrices, the Laplacian Matrix and Electrical Energy

To exploit tools from linear algebra, we first have to introduce some interesting matrices that capture important properties of the graph G. In Chapter 1, we have already seen some of these matrices. Let us recall them and also revisit some additional material from the lectures before. For the rest of the lecture, we let $G = (V, E, \boldsymbol{w})$ be an undirected graph, with $n = |V|$ vertices and $m = |E|$ edges, where $\boldsymbol{w} \in \mathbb{R}_+^E$ assigns a positive weight for every edge. We also assume that G is connected!

Fundamental Matrices. We start by re-introducing the *edge-vertex incidence matrix* $\boldsymbol{B}$ of the graph G. Therefore, we first have to associate an arbitrary direction to every edge. We then let $\boldsymbol{B} \in \mathbb{R}^{V \times E}$.

$$\boldsymbol{B}(v, e) = \begin{cases} 1 & \text{if } e = (u, v) \\ -1 & \text{if } e = (v, u) \\ 0 & \text{o.w.} \end{cases}$$

The edge directions are only there to help us track the meaning of signs of quantities defined on edges: The math we do should not depend on the choice of sign.

We define the *weight matrix* $\boldsymbol{W} \in \mathbb{R}^{E \times E}$ be the diagonal matrix given by $\boldsymbol{W} = \operatorname{diag}(\boldsymbol{w})$, i.e $\boldsymbol{W}(e, e) = \boldsymbol{w}(e)$. The weighted degree of a vertex is defined as $\boldsymbol{d}(v) = \sum_{\{u,v\} \in E} \boldsymbol{w}(u, v)$.

Again we treat the edges as undirected. Let $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{d})$ be the *degree matrix* that is a diagonal matrix in $\mathbb{R}^{V \times V}$ with weighted degrees on the diagonal.

We define the *weighted adjacency matrix* $\boldsymbol{A} \in \mathbb{R}^{V \times V}$ of a graph by

$$\boldsymbol{A}(u,v) = \begin{cases} \boldsymbol{w}(u,v) & \text{if } \{u,v\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

Note that we treat the edges as undirected here, so $\boldsymbol{A}^\top = \boldsymbol{A}$.

Finally, we define the *Laplacian* of the graph as $\boldsymbol{L} = \boldsymbol{B}\,\boldsymbol{W}\,\boldsymbol{B}^\top$. Note that in the first chapter, we defined the Laplacian as $\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top$, where $\boldsymbol{R}$ is the diagonal matrix with edge resistances on the diagonal. We want to think of high *weight* on an edge as expressing that two vertices are highly connected, whereas we think of high resistance on an edge as expressing that the two vertices are poorly connected, so we let $\boldsymbol{w}(e) = 1/\boldsymbol{R}(e,e)$. In Problem Set 1, you showed that $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, and that for $\boldsymbol{x} \in \mathbb{R}^V$,

$$\boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x} = \sum_{\{a,b\} \in E} \boldsymbol{w}(a,b)(\boldsymbol{x}(a) - \boldsymbol{x}(b))^2.$$

Flows and Voltages. Let us next briefly review flows and voltages. Given the edge-vertex incidence matrix $\boldsymbol{B}$, we can now express the net flow constraint that $\boldsymbol{f}$ routes a demand $\boldsymbol{d}$ by

$$\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}.$$

This is also called a conservation constraint. In our examples so far, we have $\boldsymbol{d}(s) = -1$, $\boldsymbol{d}(t) = 1$ and $\boldsymbol{d}(u) = 0$ for all $u \in V \setminus \{s,t\}$. Note that, unfortunately, the demand vector is typically denoted by $\boldsymbol{d}$, but so is the degree vector. Since it should always be clear from context which is which, we will also overload $\boldsymbol{d}$ in this way!

We then define for any flow $\boldsymbol{f} \in \mathbb{R}^E$ the electrical energy by

$$\mathcal{E}(\boldsymbol{f}) = \sum_e \boldsymbol{r}(e)\boldsymbol{f}(e)^2 = \boldsymbol{f}^\top \boldsymbol{R} \boldsymbol{f}$$

and for any electrical voltages $\boldsymbol{x} \in \mathbb{R}^E$

$$\mathcal{E}(\boldsymbol{f}) = \boldsymbol{f}^\top \boldsymbol{R} \boldsymbol{f} = \boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x}.$$

While the former definition is standard, the second definition is motivated by the fact that by Ohm's law, where we use $\boldsymbol{R} = \mathrm{diag}_{e \in E}\, \boldsymbol{r}(e)$, we have that electrical voltages $\boldsymbol{x}$ will induce an electrical flow $\boldsymbol{f} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x}$. And for such a pair $\boldsymbol{f}$ and $\boldsymbol{x}$, we then have that the energy associated with $\boldsymbol{x}$ is equal to the energy consumed by the flow $\boldsymbol{f}$ induced by $\boldsymbol{x}$:

$$\mathcal{E}(\boldsymbol{f}) = \boldsymbol{f}^\top \boldsymbol{R} \boldsymbol{f} = \boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x} = \mathcal{E}(\boldsymbol{x}).$$

The Courant-Fisher Theorem. Let us also recall the Courant-Fischer theorem, which we proved in Chapter 3 (Theorem 3.1.4). Here we add an additional characterization of λ_i. Note that the second equality for both versions to obtain λ_i can be extracted from our proof of the Courant-Fischer theorem in Chapter 3.

Theorem 4.1.1 (The Courant-Fischer Theorem, eigenbasis version)**.** *Let $\boldsymbol{A}$ be a symmetric matrix in $\mathbb{R}^{n\times n}$, with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, and corresponding eigenvectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ which form an othernormal basis. Then*

1. $$\lambda_i = \min_{\substack{\text{subspace } W\subseteq\mathbb{R}^n \\ \dim(W)=i}} \max_{\boldsymbol{x}\in W, \boldsymbol{x}\neq\boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}} = \min_{\substack{\boldsymbol{x}\perp \boldsymbol{x}_1,\ldots\boldsymbol{x}_{i-1} \\ \boldsymbol{x}\neq\boldsymbol{0}}} \frac{\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}}$$
2. $$\lambda_i = \max_{\substack{\text{subspace } W\subseteq\mathbb{R}^n \\ \dim(W)=n+1-i}} \min_{\boldsymbol{x}\in W, \boldsymbol{x}\neq\boldsymbol{0}} \frac{\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}} = \max_{\substack{\boldsymbol{x}\perp \boldsymbol{x}_{i+1},\ldots\boldsymbol{x}_{n} \\ \boldsymbol{x}\neq\boldsymbol{0}}} \frac{\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}}$$

Of course, we also have $\lambda_i(\boldsymbol{A}) = \frac{\boldsymbol{x}_i^\top \boldsymbol{A}\boldsymbol{x}_i}{\boldsymbol{x}_i^\top\boldsymbol{x}_i}$.

4.2 Understanding Eigenvalues of the Laplacian

We would like to understand the eigenvalues of the Laplacian matrix of a graph.

But first, why should we care? It turns out that Laplacian eigenvalues can help us understand many properties of a graph. But we are going to start with a simple motivating observation: Electrical voltages $\boldsymbol{x} \in \mathbb{R}^V$ consume electrical energy $\mathcal{E}(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}$. This means that by the Courant-Fischer Theorem

$$\mathcal{E}(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} \leq \lambda_n(L)\boldsymbol{x}^\top\boldsymbol{x}$$

And, for any voltages $\boldsymbol{x} \perp \boldsymbol{1}$,

$$\mathcal{E}(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} \geq \lambda_2(L)\boldsymbol{x}^\top\boldsymbol{x}.$$

Thus, we can use the eigenvalues to give upper and lower bounds on how much electrical energy will be consumed by the flow induced by $\boldsymbol{x}$, in terms compared to $\boldsymbol{x}^\top\boldsymbol{x} = \|\boldsymbol{x}\|_2^2$.

In a couple of chapters, we will also prove the following claim, which shows that the Laplacian eigenvalues can directly tell us about the electrical energy that is required to route a given demand.

Claim 4.2.1. *Given a demand vector $\boldsymbol{d} \in \mathbb{R}^V$ such that $\boldsymbol{d} \perp \boldsymbol{1}$, the electrical voltages $\boldsymbol{x}$ that route $\boldsymbol{d}$ satisfy $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$ and the electrical energy of these voltages satisfies*

$$\frac{\|\boldsymbol{d}\|_2^2}{\lambda_n} \leq \mathcal{E}(\boldsymbol{x}) \leq \frac{\|\boldsymbol{d}\|_2^2}{\lambda_2}$$

While the techniques in this section can be used to analyze various Laplacian matrices, we focus on understanding the eigenvalues of certain graph families that are particularly instructive. In this section, we obtain the following results.

Graph Family	λ_2 lower bound	λ_2 upper bound	λ_n lower bound	λ_n upper bound
Complete Graph K_n	n	n	n	n
Path Graph P_n	$\frac{1}{n^2}$	$\frac{12}{n^2}$	1	4
Binary Tree T_n	$\frac{1}{2n\log_2(n)}$	$\frac{2}{n-1}$	1	6

4.3 The Complete Graph

To get a sense of how Laplacian eigenvalues behave, let us start by considering the n-vertex complete graph with unit weights, which we denote by K_n. The adjacency matrix of K_n is $\boldsymbol{A} = \mathbf{1}\mathbf{1}^\top - \boldsymbol{I}$, since it has ones everywhere, except for the diagonal, where entries are zero. The degree matrix $\boldsymbol{D} = (n-1)\boldsymbol{I}$. Thus the Laplacian is $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} = n\boldsymbol{I} - \mathbf{1}\mathbf{1}^\top$.

Thus for any $\boldsymbol{y} \perp \mathbf{1}$, we have $\boldsymbol{y}^\top \boldsymbol{L}\boldsymbol{y} = n\boldsymbol{y}^\top\boldsymbol{y} - (\mathbf{1}^\top\boldsymbol{y})^2 = n\boldsymbol{y}^\top\boldsymbol{y}$.

From this, we can conclude that any $\boldsymbol{y} \perp \mathbf{1}$ is an eigenvector of eigenvalue n, and that all $\lambda_2 = \lambda_3 = \ldots = \lambda_n = n$.

4.4 The Path Graph

Next, let us try to understand λ_2 and λ_n for P_n, the n vertex path graph with unit weight edges. I.e. the graph has edges $E = \{\{i, i+1\} \text{ for } i = 1 \text{ to } (n-1)\}$. This is in a sense the least well-connected unit weight graph on n vertices, whereas K_n is the most well-connected.

4.4.1 Upper Bounding $\lambda_2(P_n)$ via Test Vectors

We can use the eigenbasis version of the Courant-Fisher theorem to observe that the second-smallest eigenvalue of the Laplacian is given by

$$\lambda_2(\boldsymbol{L}) = \min_{\substack{\boldsymbol{x}\neq\mathbf{0} \\ \boldsymbol{x}^\top\mathbf{1}=0}} \frac{\boldsymbol{x}^\top\boldsymbol{L}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}}. \tag{4.1}$$

We can get a better understanding of this particular case through a couple of simple observations. Suppose $\boldsymbol{x} = \boldsymbol{y} + \alpha\mathbf{1}$, where $\boldsymbol{y} \perp \mathbf{1}$. Then $\boldsymbol{x}^\top\boldsymbol{L}\boldsymbol{x} = \boldsymbol{y}^\top\boldsymbol{L}\boldsymbol{y}$, and $\|\boldsymbol{x}\|_2^2 = \|\boldsymbol{y}\|^2 + \alpha^2\|\mathbf{1}\|^2$.

So for any given vector, you can increase the value of $\frac{\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}}$, by replacing $\boldsymbol{x}$ with its component orthogonal to $\mathbf{1}$, which we denoted by $\boldsymbol{y}$.

We can conclude from Equation (4.1) that for *any* vector $\boldsymbol{y} \perp \mathbf{1}$,

$$\lambda_2 \leq \frac{\boldsymbol{y}^\top \boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{y}}$$

When we use a vector $\boldsymbol{y}$ in this way to prove a bound on an eigenvalue, we call it a *test vector*.

Now, we'll use a test vector to give an upper bound on $\lambda_2(\boldsymbol{L}_{P_n})$. Let $\boldsymbol{x} \in \mathbb{R}^V$ be given by $\boldsymbol{x}(i) = (n+1) - 2i$, for $i \in [n]$. This vector satisfies $\boldsymbol{x} \perp \mathbf{1}$. We picked this because we wanted a sequence of values growing linearly along the path, while also making sure that the vector is orthogonal to $\mathbf{1}$. Now

$$\begin{aligned}
\lambda_2(\boldsymbol{L}_{P_n}) &\leq \frac{\sum_{i \in [n-1]} (\boldsymbol{x}(i) - \boldsymbol{x}(i+1))^2}{\sum_{i=1}^{n} \boldsymbol{x}(i)^2} \\
&= \frac{\sum_{i=1}^{n-1} 2^2}{\sum_{i=1}^{n} (n+1-2i)^2} \\
&= \frac{4(n-1)}{(n+1)n(n-1)/3} \\
&= \frac{12}{n(n+1)} \leq \frac{12}{n^2}.
\end{aligned}$$

Later, we will prove a lower bound that shows this value is right up to a constant factor. But the test vector approach based on the Courant-Fischer theorem doesn't immediately work when we want to prove lower bounds on $\lambda_2(\boldsymbol{L})$.

4.4.2 Lower Bounding $\lambda_n(P_n)$ via Test Vectors

We can see from either version of the Courant-Fischer theorem that

$$\lambda_n(\boldsymbol{L}) = \max_{\boldsymbol{v} \neq \mathbf{0}} \frac{\boldsymbol{v}^\top \boldsymbol{L}\boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}}. \tag{4.2}$$

Thus for *any* vector $\boldsymbol{y} \neq 0$,

$$\lambda_n \geq \frac{\boldsymbol{y}^\top \boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{y}}.$$

This means we can get a test vector-based lower bound on λ_n. Let us apply this to the Laplacian of P_n. We'll try the vector $\boldsymbol{x} \in \mathbb{R}^V$ given by $\boldsymbol{x}(1) = -1$, and $\boldsymbol{x}(n) = 1$ and $\boldsymbol{x}(i) = 0$ for $i \neq 1, n$.

Here we get

$$\lambda_n(\boldsymbol{L}_{P_n}) \geq \frac{\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \frac{2}{2} = 1. \tag{4.3}$$

Again, it's not clear how to use the Courant-Fischer theorem to prove an upper bound on $\lambda_n(\boldsymbol{L})$. But, later we'll see how to prove an upper that shows that for P_n, the lower bound we obtained is right up to constant factors.

4.4.3 Upper bounding $\lambda_n(P_n)$ via the Loewner Order

In the previous sections, we first saw a complete characterization of the eigenvalues and eigenvectors of the unit weight complete graph on n vertices, K_n. Namely, $\boldsymbol{L}_{K_n} = n\boldsymbol{I} - \boldsymbol{1}\boldsymbol{1}^\top$, and this means that *every* vector $\boldsymbol{y} \perp \boldsymbol{1}$ is an eigenvector of eigenvalue n. Ideally, we would like to prove an almost matching lower bound on $\lambda_2(P_n)$ and an almost matching upper bound on $\lambda_n(P_n)$, but it is not clear how to get that from the Courant-Fischer theorem.

To get there, we need to introduce some more tools. We now introduce an ordering on symmetric matrices called the *Loewner order*, which I also like to just call the positive semi-definite order. As we will see in a moment, it is a partial order on symmetric matrices, we denote it by "$\preceq$". For convenience, we allow ourselves to both write $\boldsymbol{A} \preceq \boldsymbol{B}$ and equivalently $\boldsymbol{B} \succeq \boldsymbol{A}$.

The Loewner Order, aka. the Positive Semi-Definite Order. For a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ we define that

$$\boldsymbol{A} \succeq \boldsymbol{0}$$

if and only if $\boldsymbol{A}$ is positive semi-definite.

More generally, when we have two symmetric matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times n}$, we will write

$$\boldsymbol{A} \preceq \boldsymbol{B} \text{ if and only if for all } \boldsymbol{x} \in \mathbb{R}^n \text{ we have } \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{x} \tag{4.4}$$

This is a partial order, because it satisfies the three requirements of

1. Reflexivity: $\boldsymbol{A} \preceq \boldsymbol{A}$.
2. Anti-symmetry: $\boldsymbol{A} \preceq \boldsymbol{B}$ and $\boldsymbol{B} \preceq \boldsymbol{A}$ implies $\boldsymbol{A} = \boldsymbol{B}$
3. Transitivity: $\boldsymbol{A} \preceq \boldsymbol{B}$ and $\boldsymbol{B} \preceq \boldsymbol{C}$ implies $\boldsymbol{A} \preceq \boldsymbol{C}$

Check for yourself that these properties hold!

The PSD order has other very useful properties: $\boldsymbol{A} \preceq \boldsymbol{B}$ implies $\boldsymbol{A} + \boldsymbol{C} \preceq \boldsymbol{B} + \boldsymbol{C}$ for any symmetric matrix $\boldsymbol{C}$. Convince yourself of this too!

And, combining this observation with transitivity, we can see that $\boldsymbol{A} \preceq \boldsymbol{B}$ and $\boldsymbol{C} \preceq \boldsymbol{D}$ implies $\boldsymbol{A} + \boldsymbol{C} \preceq \boldsymbol{B} + \boldsymbol{D}$.

Here is another useful property: If $\mathbf{0} \preceq \boldsymbol{A}$ then for all $\alpha \geq 1$

$$\frac{1}{\alpha}\boldsymbol{A} \preceq \boldsymbol{A} \preceq \alpha \boldsymbol{A}.$$

Here is another one:

Claim 4.4.1. *If $\boldsymbol{A} \preceq \boldsymbol{B}$, then for all i*

$$\lambda_i(\boldsymbol{A}) \leq \lambda_i(\boldsymbol{B}).$$

Proof. We can prove this Claim by applying the subspace version of the Courant-Fischer theorem.

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

□

Note that the converse of Clam 4.4.1 is very much false, for example the matrices $\boldsymbol{A} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ have equal eigenvalues, but both $\boldsymbol{A} \not\preceq \boldsymbol{B}$ and $\boldsymbol{B} \not\preceq \boldsymbol{A}$.

Remark 4.4.2. It's useful to get used to and remember some of the properties of the Loewner order, but all the things we have established so far are almost immediate from the basic characterization in Equation (4.4). So, ideally, don't memorize all these facts, instead, try to see that they are simple consequences of the definition.

Upper bounding $\lambda_n(L_G)$ Using Degrees. In an earlier chapter, we observed that for any graph $G = (V, E, \boldsymbol{w})$, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \succeq \mathbf{0}$. We can see this from $\boldsymbol{x}^\top(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{x} = \sum_{(u,v)\in E} \boldsymbol{w}(u,v)(\boldsymbol{x}(u) - \boldsymbol{x}(v))^2 \geq 0$. Similarly $\boldsymbol{D} + \boldsymbol{A} \succeq \mathbf{0}$. because $\boldsymbol{x}^\top(\boldsymbol{D} + \boldsymbol{A})\boldsymbol{x} = \sum_{(u,v)\in E} \boldsymbol{w}(u,v)(\boldsymbol{x}(u) + \boldsymbol{x}(v))^2 \geq 0$. But this means that $-\boldsymbol{A} \preceq \boldsymbol{D}$ and hence $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \preceq 2\boldsymbol{D}$.

So, for the path graph P_n, we have $\boldsymbol{L}_{P_n} \preceq \boldsymbol{D} - \boldsymbol{A} \preceq 2\boldsymbol{D} \preceq 4\boldsymbol{I}$. So by Claim 4.4.1

$$\lambda_n(\boldsymbol{L}_{P_n}) \leq 4. \tag{4.5}$$

We can see that our test vector-based lower bound on $\lambda_n(\boldsymbol{L}_{P_n})$ from Equation (4.3) is tight up to a factor 4.

Since this type of argument works for any unit weight graph, it proves the following claim.

Claim 4.4.3. *For any unit weight graph G, $\lambda_n(\boldsymbol{L}_G) \leq 2\max_{v\in V} \boldsymbol{d}(v)$.*

This is tight on a graph consisting of a single edge.

4.4.4 Lower bounding $\lambda_2(P_n)$ via the Loewner Order

We will now conclude our analysis of the eigenvalues λ_2 and λ_n of the path graph P_n by finally deriving a lower bound on $\lambda_2(P_n)$. Again, this requires us to use new tools. As in the last section, we first introduce the new tool for general graphs and then exploit the tool to derive the lower bound.

The Loewner Order for Graphs. It's sometimes convenient to overload the PSD order to also apply to graphs. We will write

$$G \preceq H$$

if $\boldsymbol{L}_G \preceq \boldsymbol{L}_H$.

For example, given two unit weight graphs $G = (V, E)$ and $H = (V, F)$, if H is a subgraph of G, then

$$\boldsymbol{L}_H \preceq \boldsymbol{L}_G.$$

We can see this from the Laplacian quadratic form:

$$\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} = \sum_{(u,v)\in E} \boldsymbol{w}(u,v)(\boldsymbol{x}(u) - \boldsymbol{x}(v))^2.$$

Dropping edges will only decrease the value of the quadratic form. The same is for decreasing the weights of edges. The graph order notation is especially useful when we allow for scaling a graph by a constant, say $c > 0$,

$$c \cdot H \preceq G$$

What is $c \cdot H$? It is the same graph as H, but the weight of every edge is multiplied by c. Now we can make statements like $\frac{1}{2}H \preceq G \preceq 2H$, which turn out to be useful notion of the two graphs approximating each other.

The Path Inequality. Now, we'll see a general tool for comparing two graphs G and H to prove inequalities like $cH \preceq G$ for some constant c. Our tools won't necessarily work well for all cases, but we'll see some examples where they do.

In the rest of the chapter, we will often need to compare two graphs defined on the same vertex set $V = \{1, \ldots, n\} = [n]$.

We use $G_{i,j}$ to denote the unit weight graph on vertex set $[n]$ consisting of a single edge between vertices i and j.

Lemma 4.4.4 (The Path Inequality)**.**

$$(n-1) \cdot P_n \succeq G_{1,n},$$

Proof. We want to show that for every $\boldsymbol{x} \in \mathbb{R}^n$,

$$(n-1) \cdot \sum_{i=1}^{n-1} (\boldsymbol{x}(i+1) - \boldsymbol{x}(i))^2 \geq (\boldsymbol{x}(n) - \boldsymbol{x}(1))^2.$$

For $i \in [n-1]$, set

$$\boldsymbol{\Delta}(i) = \boldsymbol{x}(i+1) - \boldsymbol{x}(i).$$

The inequality we want to prove then becomes

$$(n-1) \sum_{i=1}^{n-1} (\boldsymbol{\Delta}(i))^2 \geq \left(\sum_{i=1}^{n-1} \boldsymbol{\Delta}(i) \right)^2.$$

But, this is immediate from the Cauchy-Schwarz inequality $\boldsymbol{a}^\top \boldsymbol{b} \leq \|\boldsymbol{a}\|_2 \|\boldsymbol{b}\|_2$:

$$\begin{aligned}
(n-1) \sum_{i=1}^{n-1} (\boldsymbol{\Delta}(i))^2 &= \|\mathbf{1}_{n-1}\|^2 \cdot \|\boldsymbol{\Delta}\|^2 \\
&= (\|\mathbf{1}_{n-1}\| \cdot \|\boldsymbol{\Delta}\|)^2 \\
&\geq (\mathbf{1}_{n-1}^\top \boldsymbol{\Delta})^2 \\
&= (\sum_{i=1}^{n-1} \boldsymbol{\Delta}(i))^2
\end{aligned}$$

□

Lower Bounding λ_2 of a Path Graph. We will now use Lemma 4.4.4 to prove a lower bound on $\lambda_2(\boldsymbol{L}_{P_n})$. Our strategy will be to prove that the path P_n is at least some multiple of the complete graph K_n, measured by the Loewner order, i.e. $K_n \preceq f(n) P_n$ for some function $f : \mathbb{N} \to \mathbb{R}$. We can combine this with our observation earlier that $\lambda_2(\boldsymbol{L}_{K_n}) = n$ to show that

$$f(n) \lambda_2(\boldsymbol{L}_{P_n}) \geq \lambda_2(\boldsymbol{L}_{K_n}) = n, \tag{4.6}$$

and this will give our lower bound on $\lambda_2(\boldsymbol{L}_{P_n})$. When establishing the inequality between P_n and K_n, we can treat each edge of the complete graph separately, by first noting that

$$\boldsymbol{L}_{K_n} = \sum_{i<j} \boldsymbol{L}_{G_{i,j}}$$

For every edge (i, j) in the complete graph, we apply the Path Inequality, Lemma 4.4.4:

$$\begin{aligned}
G_{i,j} &\preceq (j-i) \sum_{k=i}^{j-1} G_{k,k+1} \\
&\preceq (j-i) P_n
\end{aligned}$$

This inequality says that $G_{i,j}$ is at most $(j-i)$ times the part of the path connecting i to j, and that this part of the path is less than the whole.

Summing inequality (4.3) over all edges $(i,j) \in K_n$ gives

$$K_n = \sum_{i<j} G_{i,j} \preceq \sum_{i<j} (j-i) P_n.$$

To finish the proof, we compute

$$\sum_{i<j} (j-i) \leq \sum_{i<j} n \leq n^3$$

So

$$L_{K_n} \preceq n^3 \cdot L_{P_n}.$$

Plugging this into Equation (4.6) we obtain

$$\frac{1}{n^2} \leq \lambda_2(P_n).$$

This only differs from our test vector-based upper bound in Equation (4.3) by a factor 12.

We could make this considerably tighter by being more careful about the sums.

4.5 The Complete Binary Tree

To consolidate our new expertise, let us finally consider the complete binary tree T_n with unit weight edges on n vertices. Note that in order to have T_n be a complete binary tree, we require that n is such that there is an integer d for which $n = 2^{d+1} - 1$.

T_n is the balanced binary tree on this many vertices, i.e. it consists of a root node, which has two children, each of those children have two children and so on until we reach a depth of d from the root, at which point the child vertices have no more children. A simple induction shows that indeed $n = 2^{d+1} - 1$.

We can also describe the edge set by saying that each node i has edges to its children $2i$ and $2i+1$ whenever the node labels do not exceed n. We emphasize that we still think of the graph as undirected.

4.5.1 Deriving Bounds on $\lambda_n(T_n)$

We'll start by above bounding $\lambda_n(\boldsymbol{L}_{T_n})$ using a test vector.

We let $\boldsymbol{x}(i) = 0$ for all nodes that have a child node, and $\boldsymbol{x}(i) = -1$ for even-numbered leaf nodes and $\boldsymbol{x}(i) = +1$ for odd-numbered leaf nodes. Note that there are $(n+1)/2$ leaf nodes, and every leaf node has a single edge, connecting it to a parent with value 0. Thus

$$\lambda_n(\boldsymbol{L}) = \max_{\boldsymbol{v} \neq \boldsymbol{0}} \frac{\boldsymbol{v}^\top \boldsymbol{L} \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} \geq \frac{\boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \frac{(n+1)/2}{(n+1)/2} = 1. \tag{4.7}$$

Meanwhile, every vertex has degree at most 3, so by Claim 4.4.3, $\lambda_n(\boldsymbol{L}) \leq 6$. So we can bound the largest eigenvalue above and below by a constant.

4.5.2 Upper Bounding λ_2 via Test Vectors

Let us give an upper bound on λ_2 of the tree using a test vector. Let $\boldsymbol{x} \in \mathbb{R}^v$ have $\boldsymbol{x}(1) = 0$ and $\boldsymbol{x}(i) = -1$ for i in the left subtree and $\boldsymbol{x}(i) = +1$ in the right subtree. Then

$$\lambda_2(\boldsymbol{L}_{T_n}) = \min_{\substack{\boldsymbol{v} \neq \boldsymbol{0} \\ \boldsymbol{v}^\top \boldsymbol{1} = 0}} \frac{\boldsymbol{v}^\top \boldsymbol{L} \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} \leq \frac{\boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \frac{2}{n-1}.$$

So, we have shown $\frac{1}{2n \log_2(n)} \leq \lambda_2(\boldsymbol{L}_{T_n}) \leq \frac{2}{n-1}$, and unlike the previous examples, the gap is more than a constant.

4.5.3 Lower Bounding λ_2

The following lemma gives a simple lower bound on λ_2 for any graph.

Lemma 4.5.1. *For any unit weight graph G with diameter D,*

$$\lambda_2(\boldsymbol{L}_G) \geq \frac{1}{nD}.$$

Proof. We will again prove a lower bound comparing G to the complete graph. For each edge $(i, j) \in K_n$, let $G^{i,j}$ denote a shortest path in G from i to j. This path will have length at most D. So, we have

$$\begin{aligned} K_n &= \sum_{i<j} G_{i,j} \\ &\preceq \sum_{i<j} DG^{i,j} \\ &\preceq \sum_{i<j} DG \\ &\preceq n^2 DG. \end{aligned}$$

So, we obtain the bound

$$n^2 D \lambda_2(G) \geq n,$$

which implies our desired statement. □

Since the complete binary tree T_n has diameter $2d \leq 2\log_2(n)$, by Lemma 4.5.1, $\lambda_2(\boldsymbol{L}_{T_n}) \geq \frac{1}{2n\log_2(n)}$.

In the exercises for Week 3, I will ask you to improve the lower bound to $1/(cn)$ for some constant c.

Chapter 5

Conductance, Expanders and Cheeger's Inequality

A common algorithmic problem that arises is the problem of partitioning the vertex set V of a graph G into clusters $X_1, X_2, \ldots, X_k$ such that

- for each i, the *induced* graph $G[X_i] = (X_i, E \cap (X_i \times X_i))$ is "well-connected", and
- only an ϵ-fraction of edges e are not contained in any induced graph $G[X_i]$ (where ϵ is a very small constant).

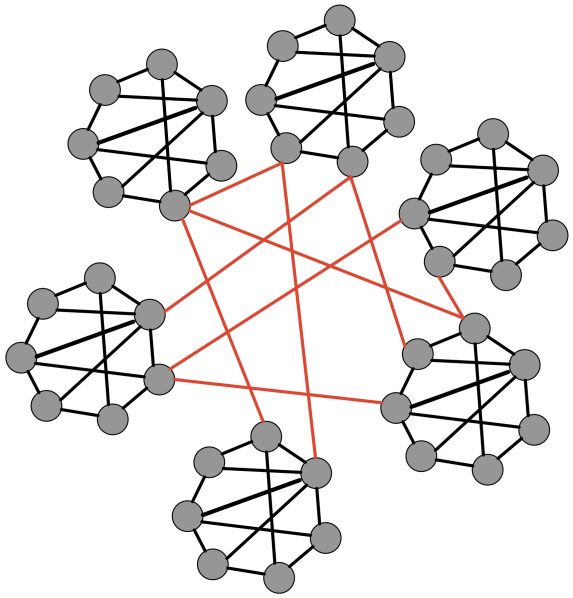


Figure 5.1: After removing the red edges (of which there are few in relation to the total number of edges), each connected component in G is "well-connected".

In this lecture, we make precise what "well-connected" means by introducing the notions of *conductance* and *expanders*.

Building on the chapter, we show that the second eigenvalue of the Laplacian L associated with graph G can be used to certify that a graph is "well-connected" (more precisely the second eigenvalue of a normalized version of the Laplacian). This result, called Cheeger's inequality, is one of the key tools in Spectral Graph Theory. Moreover, it can be turned into an algorithm that computes the partition efficiently!

5.1 Conductance and Expanders

Graph Definitions. In this lecture, we let $G = (V, E)$ be unweighted[1] and always be *connected*, and let $\boldsymbol{d}(v)$ be the degree of a vertex v in G. We define the *volume* $\mathrm{vol}(S)$ for any vertex subset $S \subseteq V$, to be the sum of degrees, i.e. $\mathrm{vol}(S) = \sum_{v \in S} \boldsymbol{d}(v)$.

For any $A, B \subseteq V$, we define $E(A, B)$ to be the set of edges in $E \cap (A \times B)$, i.e. with one endpoint in A and one endpoint in B. We let $G[A]$ be the *induced* graph G by $A \subseteq V$, which is the graph G restricted to the vertices A, i.e. an edge e in G is in $G[A]$ iff both endpoints are in A.

Conductance. Given set $\emptyset \subset S \subset V$, then we define the conductance $\phi(S)$ of S by

$$\phi(S) = \frac{|E(S, V \setminus S)|}{\min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}}.$$

It can be seen that $\phi(\cdot)$ is symmetric in the sense that $\phi(S) = \phi(V \setminus S)$. We define the conductance of the graph G denoted $\phi(G)$ by

$$\phi(G) = \min_{\emptyset \subset S \subset V} \phi(S).$$

We note that finding the conductance of a graph G is NP-hard. However, good approximations can be found as we will see today (and in a later lecture).

Expander and Expander Decomposition. For any $\phi \in (0, 1]$, we say that a graph G is a *ϕ-expander* if $\phi(G) \geq \phi$. We say that the partition $X_1, X_2, \ldots, X_k$ of the vertex set V is a *ϕ-expander decomposition* of quality q if

- each induced graph $G[X_i]$ is a ϕ-expander, and
- the number of edges not contained in any $G[X_i]$ is at most $q \cdot \phi \cdot m$.

[1] Everything we present here also works for weighted graphs, however, we focus on unweighted graphs for simplicity.

Today, we obtain a ϕ-expander decomposition of quality $q = O(\phi^{-1/2} \cdot \log n)$. In a few lectures, we revisit the problem and obtain quality $q = O(\log^c n)$ for some small constant c. In practice, we mostly care about values $\phi \approx 1$.

An Algorithm to Compute Conductance and Expander Decomposition. In this lecture, the main focus is *not* to obtain an algorithm to compute conductance but rather only to show that the conductance of a graph can be approximated using the eigenvalues of the "normalized" Laplacian.

However, this proof gives then rise to an algorithm CertifyOrCut(G, ϕ) that given a graph G and a parameter ϕ either:

- *Certifies* that G is a ϕ-expander, or
- Presents a *cut* S such that $\phi(S) \leq \sqrt{2\phi}$.

Careful but rather straight-forward recursive application of this procedure then yields an algorithm to compute a ϕ-expander decomposition.

To implement this algorithm above, we are required to find the second eigenvalue of the 'normalized' Laplacian $\boldsymbol{N}$ that we introduce in this lecture. We can find the eigenvalues using the power method[2] which multiplies a random vector with the matrix multiple times to find the largest eigenvalue. In our case, we want to use the inverse of $\boldsymbol{N}$ and pick a random vector orthogonal to the trivial first eigenvector which yields the 2nd eigenvalue of $\boldsymbol{N}$. The inverse-matrix vector multiplications can be done extremely fast given a solver for Laplacian systems $\boldsymbol{Lx} = \boldsymbol{d}$ which we learn about in the next chapters.

5.2 A Lower Bound for Conductance via Eigenvalues

An Alternative Characterization of Conductance. Let us now take a closer look at the definition of conductance and observe that if a set S has $\text{vol}(S) \leq \text{vol}(V)/2$ then

$$\phi(S) = \frac{|E(S, V \setminus S)|}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}} = \frac{|E(S, V \setminus S)|}{\text{vol}(S)} = \frac{\mathbf{1}_S^\top \boldsymbol{L} \mathbf{1}_S}{\mathbf{1}_S^\top \boldsymbol{D} \mathbf{1}_S}.$$

To see this, observe that we can rewrite the numerator above using the Laplacian of G as

$$|E(S, V \setminus S)| = \sum_{(u,v) \in E} (\mathbf{1}_S(u) - \mathbf{1}_S(v))^2 = \mathbf{1}_S^\top \boldsymbol{L} \mathbf{1}_S$$

where $\mathbf{1}_S$ is the characteristic vector of S. Further, we can rewrite the denominator as

$$\text{vol}(S) = \mathbf{1}_S^\top \boldsymbol{d} = \mathbf{1}_S^\top \boldsymbol{D} \mathbf{1}_S$$

[2]See `https://en.wikipedia.org/wiki/Power_iteration`.

where $\boldsymbol{D} = \text{diag}(\boldsymbol{d})$ is the degree-matrix. We can now alternatively define the graph conductance of G by

$$\phi(G) = \min_{\substack{\emptyset \subset S \subset V, \\ \text{vol}(S) \leq \text{vol}(V)/2}} \frac{\mathbf{1}_S^\top \boldsymbol{L} \mathbf{1}_S}{\mathbf{1}_S^\top \boldsymbol{D} \mathbf{1}_S} \tag{5.1}$$

where we use that $\phi(S) = \phi(V \setminus S)$ such that the objective value is unchanged as long as for each set $\emptyset \subset S \subset V$ either S or $V \setminus S$ is in the set that we minimize over.

The Normalized Laplacian. Let us next define the *normalized* Laplacian

$$\boldsymbol{N} = \boldsymbol{D}^{-1/2} \boldsymbol{L} \boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2} \boldsymbol{A} \boldsymbol{D}^{-1/2}.$$

To learn a bit about this new matrix, let us first look at the first eigenvalue where we use the test vector $y = \boldsymbol{D}^{1/2}\mathbf{1}$, to get by Courant-Fischer (see Theorem 3.1.4) that

$$\lambda_1(\boldsymbol{N}) = \min_{\boldsymbol{x} \neq \mathbf{0}} \frac{\boldsymbol{x}^\top \boldsymbol{N} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} \leq \frac{\boldsymbol{y}^\top \boldsymbol{D}^{-1/2} \boldsymbol{L} \boldsymbol{D}^{-1/2} \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{y}} = \frac{\mathbf{1}^\top \boldsymbol{L} \mathbf{1}}{\boldsymbol{y}^\top \boldsymbol{y}} = 0 \tag{5.2}$$

because $\boldsymbol{D}^{-1/2}\boldsymbol{D}^{1/2} = I$ and $\boldsymbol{L}\mathbf{1} = 0$ (for the former we use the assumption that G is connected). Since $\boldsymbol{N}$ is PSD (as you will show in the exercises), we also know $\lambda_1(\boldsymbol{N}) \geq 0$, so $\lambda_1(\boldsymbol{N}) = 0$.

Let us use Courant-Fischer again to reason a bit about the second eigenvalue of $\boldsymbol{N}$:

$$\lambda_2(\boldsymbol{N}) = \min_{\substack{\boldsymbol{x} \perp \boldsymbol{D}^{1/2}\mathbf{1} \\ \boldsymbol{x} \neq \mathbf{0}}} \frac{\boldsymbol{x}^\top \boldsymbol{N} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \min_{\substack{\boldsymbol{z} \perp \boldsymbol{d} \\ \boldsymbol{z} \neq \mathbf{0}}} \frac{\boldsymbol{z}^\top \boldsymbol{D}^{1/2} \boldsymbol{N} \boldsymbol{D}^{1/2} \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}^{1/2} \boldsymbol{D}^{1/2} \boldsymbol{z}} = \min_{\substack{\boldsymbol{z} \perp \boldsymbol{d} \\ \boldsymbol{z} \neq \mathbf{0}}} \frac{\boldsymbol{z}^\top \boldsymbol{L} \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D} \boldsymbol{z}}. \tag{5.3}$$

Relating Conductance to the Normalized Laplacian. At this point, it might become clearer why $\boldsymbol{N}$ is a natural matrix to consider when arguing about conductance: if we could argue that for every $\emptyset \subset S \subset V, \text{vol}(S) \leq \text{vol}(V)/2$, we have $\mathbf{1}_S \perp \boldsymbol{d}$, then it would be easy to see that taking the second eigenvalue of $\boldsymbol{N}$ in equation 5.3 is a relaxation of the minimization problem 5.1 defining $\phi(G)$.

While this is clearly not true, we can still argue along these lines.

Theorem 5.2.1 (Cheeger's Inequality, Lower Bound)**.** *We have* $\frac{\lambda_2(\boldsymbol{N})}{2} \leq \phi(G)$.

Proof. Instead of using $\mathbf{1}_S$ directly, we shift $\mathbf{1}_S$ by $\mathbf{1}$ such that it is orthogonal to $\boldsymbol{d}$: we define $\boldsymbol{z}_S = \mathbf{1}_S - \alpha\mathbf{1}$ where α is the scalar that solves

$$\begin{aligned}
& 0 = \boldsymbol{d}^\top \boldsymbol{z}_S \\
\iff & 0 = \boldsymbol{d}^\top (\mathbf{1}_S - \alpha \mathbf{1}) \\
\iff & 0 = \boldsymbol{d}^\top \mathbf{1}_S - \alpha \boldsymbol{d}^\top \mathbf{1} \\
\iff & \alpha = \frac{\boldsymbol{d}^\top \mathbf{1}_S}{\boldsymbol{d}^\top \mathbf{1}} = \frac{\text{vol}(S)}{\text{vol}(V)}.
\end{aligned}$$

To conclude the proof, it remains to argue that $\frac{\mathbf{1}_S^\top \boldsymbol{L}\mathbf{1}_S}{\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S} \geq \frac{1}{2} \cdot \frac{\boldsymbol{z}_S^\top \boldsymbol{L}\boldsymbol{z}_S}{\boldsymbol{z}_S^\top \boldsymbol{D}\boldsymbol{z}_S}$:

- Numerator: since $\mathbf{1}^\top \boldsymbol{L}\mathbf{1} = 0$, we have that $\mathbf{1}_S^\top \boldsymbol{L}\mathbf{1}_S = \boldsymbol{z}_S^\top \boldsymbol{L}\boldsymbol{z}_S$.
- Denominator: observe by straight-forward calculations that

$$\begin{aligned}
\boldsymbol{z}_S^\top \boldsymbol{D}\boldsymbol{z}_S &= \mathrm{vol}(S) \cdot (1-\alpha)^2 + \mathrm{vol}(V \setminus S) \cdot (-\alpha)^2 \\
&= \mathrm{vol}(S) - 2\,\mathrm{vol}(S) \cdot \alpha + \mathrm{vol}(V) \cdot \alpha^2 \\
&= \mathrm{vol}(S) - \frac{\mathrm{vol}(S)^2}{\mathrm{vol}(V)} \\
&= \mathrm{vol}(S) - \mathrm{vol}(S) \cdot \frac{\mathrm{vol}(S)}{\mathrm{vol}(V)} \\
&\geq \frac{1}{2}\,\mathrm{vol}(S) = \frac{1}{2}\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S
\end{aligned}$$

  where we use the assumption that $\mathrm{vol}(S) \leq \mathrm{vol}(V)/2$.

□

5.3 An Upper Bound for Conductance via Eigenvalues

Slightly more surprisingly, we can also show that the second eigenvalue $\lambda_2(\boldsymbol{N})$ can be used to upper bound the conductance.

Theorem 5.3.1 (Cheeger's Inequality, Upper Bound)**.** *We have* $\phi(G) \leq \sqrt{2 \cdot \lambda_2(\boldsymbol{N})}$.

Proof. To prove the theorem, we want to show that for *any* $\boldsymbol{z} \perp \boldsymbol{d}$, we can find a set $\emptyset \subset S \subset V$ with $\mathrm{vol}(S) \leq \mathrm{vol}(V)/2$, such that

$$\frac{\mathbf{1}_S^\top \boldsymbol{L}\mathbf{1}_S}{\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S} \leq \sqrt{2 \cdot \frac{\boldsymbol{z}^\top \boldsymbol{L}\boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}. \tag{5.4}$$

As a first step, we would like to change $\boldsymbol{z}$ slightly to make it more convenient to work with:

- we *renumber* the vertices in V such that we have

$$\boldsymbol{z}(1) \leq \boldsymbol{z}(2) \leq \cdots \leq \boldsymbol{z}(n).$$

- we *center* $\boldsymbol{z}$, that is we let $\boldsymbol{z}_c = \boldsymbol{z} - \alpha\mathbf{1}$ where α is chosen such that

$$\sum_{\boldsymbol{z}_c(i)<0} \boldsymbol{d}(i) < \mathrm{vol}(V)/2 \text{ and } \sum_{\boldsymbol{z}_c(i)\leq 0} \boldsymbol{d}(i) \geq \mathrm{vol}(V)/2$$

  i.e. $\sum_{\boldsymbol{z}_c(i)>0} \boldsymbol{d}(i) \leq \mathrm{vol}(V)/2$.

- we *scale*, let $\boldsymbol{z}_{sc} = \beta \boldsymbol{z}_c$ for some scalar β such that $\boldsymbol{z}_{sc}(1)^2 + \boldsymbol{z}_{sc}(n)^2 = 1$.

In the exercises, you will show that changing $\boldsymbol{z}$ to $\boldsymbol{z}_{sc}$ can only make the ratio we are interested in smaller, i.e. that $\frac{\boldsymbol{z}^\top \boldsymbol{L}\boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}} \geq \frac{\boldsymbol{z}_{sc}^\top \boldsymbol{L}\boldsymbol{z}_{sc}}{\boldsymbol{z}_{sc}^\top \boldsymbol{D}\boldsymbol{z}_{sc}}$. Thus, if we can show that equation 5.4 holds for $\boldsymbol{z}_{sc}$ in place of $\boldsymbol{z}$, then it also follows for $\boldsymbol{z}$ itself.

We now arrive at the main idea of the proof: we define the set $S_\tau = \{i \in V \mid \boldsymbol{z}_{sc}(i) < \tau\}$ for some random variable τ with probability density function

$$p(t) = \begin{cases} 2 \cdot |t| & t \in [\boldsymbol{z}_{sc}(1), \boldsymbol{z}_{sc}(n)], \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

So, we have probability $\mathbb{P}[\alpha < \tau < \beta] = \int_{t=\alpha}^{\beta} p(t)\, dt$.

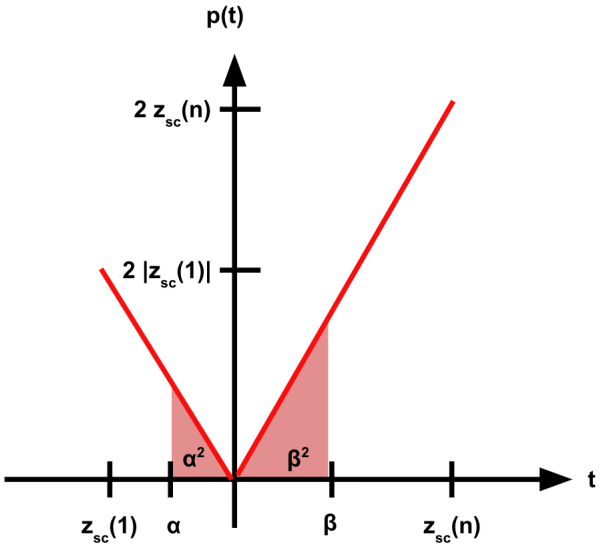


Figure 5.2: The probability density function $p(t)$. The area between α and β for $\alpha < 0 < \beta$ is $\alpha^2 + \beta^2$. This is the probability of the event $\mathbb{P}[\alpha < \tau < \beta]$. Note in particular that the total area is $\boldsymbol{z}_{sc}(1)^2 + \boldsymbol{z}_{sc}(n)^2$ which is equal to 1 by assumption and thus $p(t)$ is a valid pdf.

Since the volume incident to S_τ might be quite large, let us define S for convenience by

$$S = \begin{cases} S_\tau & \text{vol}(S_\tau) < \text{vol}(V)/2, \\ V \setminus S_\tau & \text{otherwise.} \end{cases}$$

Claim 5.3.2. *We have* $\frac{\mathbb{E}_\tau\left[\mathbf{1}_S^\top \boldsymbol{L}\mathbf{1}_S\right]}{\mathbb{E}_\tau\left[\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S\right]} \leq \sqrt{2 \cdot \frac{\boldsymbol{z}_{sc}^\top \boldsymbol{L}\boldsymbol{z}_{sc}}{\boldsymbol{z}_{sc}^\top \boldsymbol{D}\boldsymbol{z}_{sc}}}$.

Proof. Recall $\mathbf{1}_S^\top \boldsymbol{L}\mathbf{1}_S = E(S_\tau, V \setminus S_\tau)$, and by choice of τ, we have for any edge $e = \{i, j\} \in E$

where $\boldsymbol{z}_{sc}(i) \leq \boldsymbol{z}_{sc}(j)$,

$$\begin{aligned}\mathbb{P}_\tau[e \in E(S_\tau, V \setminus S_\tau)] &= \mathbb{P}_\tau[\boldsymbol{z}_{sc}(i) < \tau \leq \boldsymbol{z}_{sc}(j)] \\ &= \int_{t=i}^{j} 2|t| \, dt \\ &= \begin{cases} \boldsymbol{z}_{sc}(i)^2 + \boldsymbol{z}_{sc}(j)^2 & \boldsymbol{z}(i) < 0 < \boldsymbol{z}(j), \\ |\boldsymbol{z}_{sc}(i)^2 - \boldsymbol{z}_{sc}(j)^2| & \text{otherwise.} \end{cases}\end{aligned}$$

The last equality can be seen from Figure 5.2. We can further upper bound either case by $|\boldsymbol{z}_{sc}(i) - \boldsymbol{z}_{sc}(j)| \cdot (|\boldsymbol{z}_{sc}(i)| + |\boldsymbol{z}_{sc}(j)|)$ (we leave this as an exercise).

Using our new upper bound, we can sum over all edges $e \in E$ to conclude that

$$\begin{aligned}\mathbb{E}_\tau[|E(S_\tau, V \setminus S_\tau)|] &\leq \sum_{i \sim j} |\boldsymbol{z}_{sc}(i) - \boldsymbol{z}_{sc}(j)| \cdot (|\boldsymbol{z}_{sc}(i)| + |\boldsymbol{z}_{sc}(j)|) \\ &\leq \sqrt{\sum_{i \sim j} (\boldsymbol{z}_{sc}(i) - \boldsymbol{z}_{sc}(j))^2 \cdot \sum_{i \sim j} (|\boldsymbol{z}_{sc}(i)| + |\boldsymbol{z}_{sc}(j)|)^2}\end{aligned}$$

where the last line follows from $\langle \boldsymbol{x}, \boldsymbol{y} \rangle^2 \leq \langle \boldsymbol{x}, \boldsymbol{x} \rangle \cdot \langle \boldsymbol{y}, \boldsymbol{y} \rangle$ (i.e. Cauchy-Schwarz).

The first sum should look familiar by now: it is simply the Quadratic Laplacian Form $\sum_{i \sim j} (\boldsymbol{z}_{sc}(i) - \boldsymbol{z}_{sc}(j))^2 = \boldsymbol{z}_{sc}^\top \boldsymbol{L} \boldsymbol{z}_{sc}$.

It is not hard to reason about the second term either

$$\sum_{i \sim j} (|\boldsymbol{z}_{sc}(i)| + |\boldsymbol{z}_{sc}(j)|)^2 \leq 2 \sum_{i \sim j} \boldsymbol{z}_{sc}(i)^2 + \boldsymbol{z}_{sc}(j)^2 = 2 \sum_{i \in V} \boldsymbol{d}(i) \boldsymbol{z}_{sc}(i)^2 = 2 \boldsymbol{z}_{sc}^\top \boldsymbol{D} \boldsymbol{z}_{sc}.$$

Putting everything together, we obtain

$$\mathbb{E}_\tau[|E(S_\tau, V \setminus S_\tau)|] \leq \sqrt{\boldsymbol{z}_{sc}^\top \boldsymbol{L} \boldsymbol{z}_{sc} \cdot 2 \boldsymbol{z}_{sc}^\top \boldsymbol{D} \boldsymbol{z}_{sc}} = \sqrt{2 \cdot \frac{\boldsymbol{z}_{sc}^\top \boldsymbol{L} \boldsymbol{z}_{sc}}{\boldsymbol{z}_{sc}^\top \boldsymbol{D} \boldsymbol{z}_{sc}}} \cdot \boldsymbol{z}_{sc}^\top \boldsymbol{D} \boldsymbol{z}_{sc} \tag{5.6}$$

While this almost looks like what we want, we still have to argue that $\boldsymbol{z}_{sc}^\top \boldsymbol{D} \boldsymbol{z}_{sc} = \mathbb{E}_\tau[\mathbf{1}_S^\top \boldsymbol{D} \mathbf{1}_S]$ to finish the proof.

To this end, when unrolling the expectation, we use a simple trick that splits by cases:

$$\begin{aligned}\mathbb{E}_\tau[\mathbf{1}_S^\top \boldsymbol{D} \mathbf{1}_S] &= \sum_{i \in V} \boldsymbol{d}(i) \cdot \mathbb{P}[i \in S] \\ &= \sum_{i \in V, \boldsymbol{z}_{sc}(i) < 0} \boldsymbol{d}(i) \cdot \mathbb{P}[i \in S \wedge S = S_\tau] + \sum_{i \in V, \boldsymbol{z}_{sc}(i) \geq 0} \boldsymbol{d}(i) \cdot \mathbb{P}[i \in S \wedge S \neq S_\tau] \\ &= \sum_{i \in V, \boldsymbol{z}_{sc}(i) < 0} \boldsymbol{d}(i) \cdot \mathbb{P}[\boldsymbol{z}_{sc}(i) < \tau \wedge \tau < 0] + \sum_{i \in V, \boldsymbol{z}_{sc}(i) \geq 0} \boldsymbol{d}(i) \cdot \mathbb{P}[\boldsymbol{z}_{sc}(i) \geq \tau \wedge \tau \geq 0]\end{aligned}$$

where we use the centering of $\boldsymbol{z}_{sc}$ the definition of S and that the event $\{i \in S \wedge S = S_\tau\}$ can be rewritten as the event $\{i < \tau \wedge \tau < 0\}$ (the other case is analogous).

Let i be a vertex with $\boldsymbol{z}_{sc}(i) < 0$, then the probability $\mathbb{P}[i \in S \wedge S = S_\tau]$ is exactly $\boldsymbol{z}_{sc}(i)^2$ by choice of the density function of τ (again the case for i with $\boldsymbol{z}_{sc}(i)$ non-negative is analgous). Thus, summing over all vertices, we obtain

$$\begin{aligned}\mathbb{E}_\tau[\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S] &= \sum_{i \in V, \boldsymbol{z}_{sc}(i)<0} \boldsymbol{d}(i) \cdot \mathbb{P}[\boldsymbol{z}_{sc}(i) < \tau \wedge \tau < 0] + \sum_{i \in V, \boldsymbol{z}_{sc}(i)\geq 0} \mathbb{P}[\boldsymbol{z}_{sc}(i) \geq \tau \wedge \tau \geq 0] \\ &= \sum_{i \in V} \boldsymbol{d}(i) \cdot \boldsymbol{z}_{sc}(i)^2 = \boldsymbol{z}_{sc}^\top \boldsymbol{D}\boldsymbol{z}_{sc}.\end{aligned}$$

Therefore, we can plug in our result directly into Equation 5.6 and the proof is completed by dividing both sides by $\mathbb{E}_\tau[\mathbf{1}_S^\top \boldsymbol{D}\mathbf{1}_S]$. □

While Theorem 5.3.2 only ensures our claim in expectation, this is already sufficient to conclude that there exists some set S that satisfies the same guarantees deterministically, as you will prove in Problem Set 4. This is often called the *probabilistic method of expectation* and can be seen from the definition of expectation. We have thus proven the upper bound of Cheeger's inequality. □

5.4 Conclusion

Today, we have introduced the concepts of conductance and formalized expanders and expander decompositions. These are crucial concepts that you will encounter often in literature and also again in this course. They are a key tool in many recent breakthroughs in Theoretical Computer Science.

In the second part of the lecture (the main part), we discussed Cheeger's inequality which allows to relate the second eigenvalue of the normalized Laplacian to a graphs conductance. We summarize the full statement here.

Theorem 5.4.1 (Cheeger's Inequality)**.** *We have* $\frac{\lambda_2(\boldsymbol{N})}{2} \leq \phi(G) \leq \sqrt{2 \cdot \lambda_2(\boldsymbol{N})}$.

We point out that this Theorem is tight as you will show in the exercises. The proof for Cheeger's inequality is probably the most advanced proof, we have seen so far in the course. The many tricks that make the proof work might sometimes seem a bit magical but it is important to remember that they are a result of many people polishing this proof over and over. The proof techniques used are extremely useful and can be re-used in various contexts. We therefore strongly encourage you to really understand the proof yourself!

Chapter 6

Pseudo-inverses and Effective Resistance

6.1 What is a (Moore-Penrose) Pseudoinverse?

Recall that for a connected graph G with Laplacian $\boldsymbol{L}$, we have $\ker(\boldsymbol{L}) = \text{span}\{\mathbf{1}\}$, which means $\boldsymbol{L}$ is not invertible. However, we still want some matrix which behaves like a real inverse. To be more specific, given a Laplacian $\boldsymbol{L} \in \mathbb{R}^{V \times V}$, we want some matrix $\boldsymbol{L}^+ \in \mathbb{R}^{V \times V}$ s.t.

1) $(\boldsymbol{L}^+)^\top = \boldsymbol{L}^+$ (symmetric)

2) $\boldsymbol{L}^+\mathbf{1} = \mathbf{0}$, or more generally, $\boldsymbol{L}^+\boldsymbol{v} = \mathbf{0}$ for $\boldsymbol{v} \in \ker(\boldsymbol{L})$

3) $\boldsymbol{L}^+\boldsymbol{L}\boldsymbol{v} = \boldsymbol{L}\boldsymbol{L}^+\boldsymbol{v} = \boldsymbol{v}$ for $\boldsymbol{v} \perp \mathbf{1}$, or more generally, for $\boldsymbol{v} \in \ker(\boldsymbol{L})^\perp$

Under the above conditions, $\boldsymbol{L}^+$ is uniquely defined and we call it the pseudoinverse of $\boldsymbol{L}$. Note that there are many other equivalent definitions of the pseudoinverse of some matrix $\boldsymbol{A}$, and we can also generalize the concept to matrices that aren't symmetric or even square.

Let $\lambda_i, \boldsymbol{v}_i$ be the i-th pair of eigenvalue and eigenvector of $\boldsymbol{L}$, with $\{\boldsymbol{v}_i\}_{i=1}^n$ forming a orthogonal basis. Then by the spectral theorem,

$$\boldsymbol{L} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top = \sum_i \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top,$$

where $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_n \end{bmatrix}$ and $\boldsymbol{\Lambda} = \text{diag}\{\lambda_1, ..., \lambda_n\}$. And we can show that its pseudoinverse is exactly

$$\boldsymbol{L}^+ = \sum_{i, \lambda_i \neq 0} \lambda_i^{-1} \boldsymbol{v}_i \boldsymbol{v}_i^\top.$$

Checking conditions 1), 2), 3) is immediate. We can also prove uniqueness, but this takes slightly more work.

6.2 Electrical Flows Again

Recall the incidence matrix $\boldsymbol{B} \in \mathbb{R}^{V \times E}$ of a graph $G = (V, E)$.

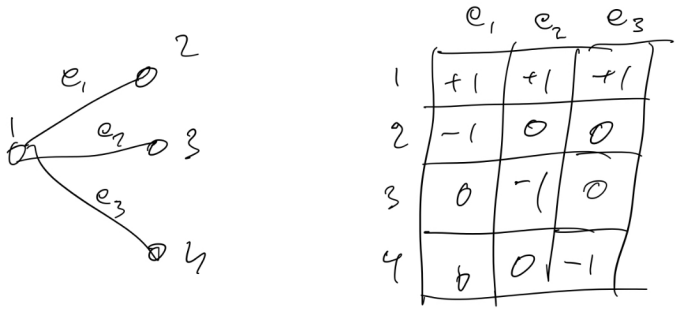


Figure 6.1: An example of a graph and its incidence matrix B.

In Chapter 1, we introduced the electrical flow routing demand $\boldsymbol{d} \in \mathbb{R}^V$. Let's call the electrical flow $\tilde{\boldsymbol{f}} \in \mathbb{R}^E$. The net flow constraint requires $\boldsymbol{B}\tilde{\boldsymbol{f}} = \boldsymbol{d}$. By Ohm's Law, $\tilde{\boldsymbol{f}} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x}$ for some voltage $x \in \mathbb{R}^V$ where $\boldsymbol{R} = \mathrm{diag}(\boldsymbol{r})$ and $\boldsymbol{r}(e) =$ resistance of edge e. We showed (in the exercises) that when $\boldsymbol{d} \perp \mathbf{1}$, there exists an voltage $\tilde{\boldsymbol{x}} \perp \mathbf{1}$ s.t. $\tilde{\boldsymbol{f}} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \tilde{\boldsymbol{x}}$ and $\boldsymbol{B}\tilde{\boldsymbol{f}} = \boldsymbol{d}$. This $\tilde{\boldsymbol{x}}$ solves $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$ where $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top$.

And we also made the following claim.

Claim 6.2.1.

$$\tilde{\boldsymbol{f}} = \operatorname*{arg\,min}_{\boldsymbol{B}\boldsymbol{f}=\boldsymbol{d}} \boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} \text{ where } \boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} = \sum_e \boldsymbol{r}(e)\boldsymbol{f}(e)^2, \tag{6.1}$$

You proved this in the exercises for Week 1. Let's recap the proof briefly, just to get back into thinking about electrical flows.

Proof. Consider any $\boldsymbol{f} \in \mathbb{R}^E$ s.t. $\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}$. For any $\boldsymbol{x} \in \mathbb{R}^V$, we have

$$\begin{aligned}
\frac{1}{2}\boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} &= \frac{1}{2}\boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} - \boldsymbol{x}^\top (\underbrace{\boldsymbol{B}\boldsymbol{f} - \boldsymbol{d}}_{\mathbf{0}}) \\
&\geq \min_{\boldsymbol{f} \in \mathbb{R}^E} \underbrace{\frac{1}{2}\boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} - \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{f} + \boldsymbol{d}^\top \boldsymbol{x}}_{g(\boldsymbol{f})} \\
&= \boldsymbol{d}^\top \boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}
\end{aligned}$$

since $\nabla_{\boldsymbol{f}} g(\boldsymbol{f}) = \mathbf{0}$ gives us $\boldsymbol{f} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{x}$. Thus, for all $\boldsymbol{f} \in \mathbb{R}^E$ s.t. $\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}$ and all $\boldsymbol{x} \in \mathbb{R}^V$,

$$\frac{1}{2}\boldsymbol{f}^\top \boldsymbol{R}\boldsymbol{f} \geq \boldsymbol{d}^\top \boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}. \tag{6.2}$$

But for the electrical flow $\tilde{\boldsymbol{f}}$ and electrical voltage $\tilde{\boldsymbol{x}}$, we have $\tilde{\boldsymbol{f}} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top\tilde{\boldsymbol{x}}$ and $\boldsymbol{L}\tilde{\boldsymbol{x}} = \boldsymbol{d}$. So

$$\tilde{\boldsymbol{f}}^\top\boldsymbol{R}\tilde{\boldsymbol{f}} = \left(\boldsymbol{R}^{-1}\boldsymbol{B}^\top\tilde{\boldsymbol{x}}\right)^\top\boldsymbol{R}\left(\boldsymbol{R}^{-1}\boldsymbol{B}^\top\tilde{\boldsymbol{x}}\right) = \tilde{\boldsymbol{x}}^\top\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^\top\boldsymbol{L}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^\top\boldsymbol{d}.$$

Therefore,

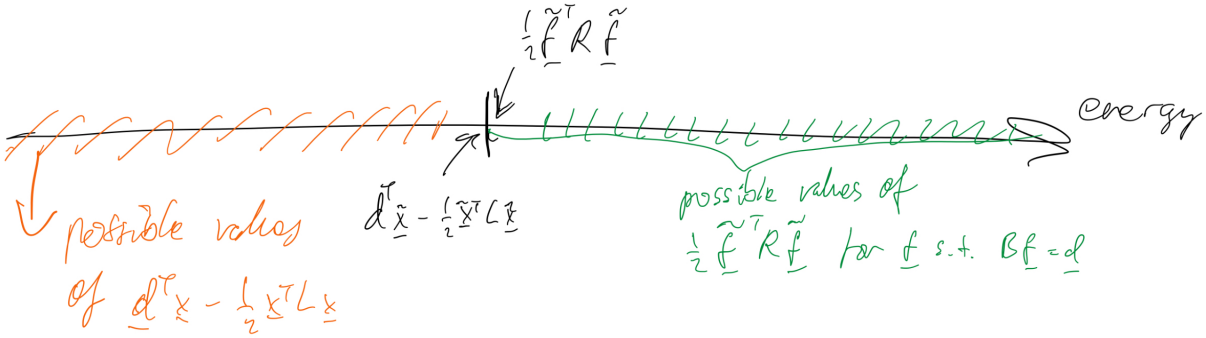
$$\frac{1}{2}\tilde{\boldsymbol{f}}^\top\boldsymbol{R}\tilde{\boldsymbol{f}} = \boldsymbol{d}^\top\tilde{\boldsymbol{x}} - \frac{1}{2}\tilde{\boldsymbol{x}}^\top\boldsymbol{L}\tilde{\boldsymbol{x}}. \tag{6.3}$$

By combining Equation (6.2) and Equation (6.3), we see that for all $\boldsymbol{f}$ s.t. $\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}$,

$$\frac{1}{2}\boldsymbol{f}^\top\boldsymbol{R}\boldsymbol{f} \geq \boldsymbol{d}^\top\tilde{\boldsymbol{x}} - \frac{1}{2}\tilde{\boldsymbol{x}}^\top\boldsymbol{L}\tilde{\boldsymbol{x}} = \frac{1}{2}\tilde{\boldsymbol{f}}^\top\boldsymbol{R}\tilde{\boldsymbol{f}}.$$

Thus $\tilde{\boldsymbol{f}}$ is the minimum electrical energy flow among all flows that route demand $\boldsymbol{d}$, proving Equation (6.1) holds.

The drawing below shows how the quantities line up:



□

6.3 Effective Resistance

Given a graph $G = (V, E)$, for any pair of vertices $(a, b) \in V$, we want to compute the cost (or energy) of routing 1 unit of current from a to b. We call such cost the effective resistance between a and b, denoted by $R_{\text{eff}}(a, b)$. Recall for a single resistor $r(a, b)$,

$$\text{energy} = r(a, b)f^2(a, b) = r(a, b).$$

So when we have a graph consisting of just one edge (a, b), the effective resistance is just $R_{\text{eff}}(a, b) = r(a, b)$.

In a general graph, we can also consider the energy required to route one unit of current between two vertices. For any pair $a, b \in V$, we have

$$R_{\text{eff}}(a, b) = \min_{\boldsymbol{B}\boldsymbol{f}=\boldsymbol{e}_b-\boldsymbol{e}_a} \boldsymbol{f}^\top\boldsymbol{R}\boldsymbol{f},$$

where $\boldsymbol{e}_v \in \mathbb{R}^V$ is the indicator vector of v. Note that the cost of routing F units of flow from a to b will be $R_{\text{eff}}(a,b) \cdot F^2$.

Since $(\boldsymbol{e}_b - \boldsymbol{e}_a)^\top \mathbf{1} = 0$, we know from the previous section that $R_{\text{eff}}(a,b) = \tilde{\boldsymbol{f}}^\top \boldsymbol{R}\tilde{\boldsymbol{f}}$ where $\tilde{\boldsymbol{f}}$ is the electrical flow. Now we can write $\boldsymbol{L}\tilde{\boldsymbol{x}} = \boldsymbol{e}_b - \boldsymbol{e}_a$ and $\tilde{\boldsymbol{x}} = \boldsymbol{L}^+(\boldsymbol{e}_b - \boldsymbol{e}_a)$ for the electrical voltages routing 1 unit of current from a to b. Now the energy of routing 1 unit of current from a to b is

$$R_{\text{eff}}(a,b) = \tilde{\boldsymbol{f}}^\top \boldsymbol{R}\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{x}}^\top \boldsymbol{L}\tilde{\boldsymbol{x}} = (\boldsymbol{e}_b - \boldsymbol{e}_a)^\top \boldsymbol{L}^+\boldsymbol{L}\boldsymbol{L}^+(\boldsymbol{e}_b - \boldsymbol{e}_a) = (\boldsymbol{e}_b - \boldsymbol{e}_a)^\top \boldsymbol{L}^+(\boldsymbol{e}_b - \boldsymbol{e}_a),$$

where the last equality is due to $\boldsymbol{L}^+\boldsymbol{L}\boldsymbol{L}^+ = \boldsymbol{L}^+$.

Remark 6.3.1. We have now seen several different expressions that all take on the same value: the energy of the electrical flow. It's useful to remind yourself what these are. Consider an electrical flow $\tilde{\boldsymbol{f}}$ routes demand $\boldsymbol{d}$, and associated electrical voltages $\tilde{\boldsymbol{x}}$. We know that $\boldsymbol{B}\tilde{\boldsymbol{f}} = \boldsymbol{d}$, and $\boldsymbol{f} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top\tilde{\boldsymbol{x}}$, and $\boldsymbol{L}\tilde{\boldsymbol{x}} = \boldsymbol{d}$, where $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top$. And we have seen how to express the electrical energy using many different quantities:

$$\tilde{\boldsymbol{f}}^\top \boldsymbol{R}\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{x}}^\top \boldsymbol{L}\tilde{\boldsymbol{x}} = \boldsymbol{d}^\top \boldsymbol{L}^+\boldsymbol{d} = \boldsymbol{d}^\top\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{f}}^\top \boldsymbol{B}^\top\tilde{\boldsymbol{x}}$$

Claim 6.3.2. *Any PSD matrix $\boldsymbol{A}$ has a PSD square root $\boldsymbol{A}^{1/2}$ s.t. $\boldsymbol{A}^{1/2}\boldsymbol{A}^{1/2} = \boldsymbol{A}$.*

Proof. By the spectral theorem, $\boldsymbol{A} = \sum_i \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^\top$ where $\{\boldsymbol{v}_i\}$ are orthonormal. Let $\boldsymbol{A}^{1/2} = \sum_i \lambda_i^{1/2}\boldsymbol{v}_i\boldsymbol{v}_i^\top$. Then

$$\begin{aligned}
\boldsymbol{A}^{1/2}\boldsymbol{A}^{1/2} &= \left(\sum_i \lambda_i^{1/2}\boldsymbol{v}_i\boldsymbol{v}_i^\top\right)^2 \\
&= \sum_i \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^\top\boldsymbol{v}_i\boldsymbol{v}_i^\top + \sum_{i\neq j} \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^\top\boldsymbol{v}_j\boldsymbol{v}_j^\top \\
&= \sum_i \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^\top
\end{aligned}$$

where the last equality is due to $\boldsymbol{v}_i^\top\boldsymbol{v}_j = \delta_{ij}$. It's easy to see that $\boldsymbol{A}^{1/2}$ is also PSD. □

Let $\boldsymbol{L}^{+/2}$ be the square root of $\boldsymbol{L}^+$. So

$$R_{\text{eff}}(a,b) = (\boldsymbol{e}_b - \boldsymbol{e}_a)^\top \boldsymbol{L}^+(\boldsymbol{e}_b - \boldsymbol{e}_a) = \|\boldsymbol{L}^{+/2}(\boldsymbol{e}_b - \boldsymbol{e}_a)\|^2.$$

Example: Effective resistance in a path. Consider a path graph on vertices $V = \{1, 2, 3, \ldots, k+1\}$, with with resistances $\boldsymbol{r}(1), \boldsymbol{r}(2), \ldots, \boldsymbol{r}(k)$ on the edges of the path.

Figure 6.2: A path graph with k edges.

The effective resistance between the endpoints is

$$R_{\text{eff}}(1, k+1) = \sum_{i=1}^{k} \boldsymbol{r}(i)$$

To see this, observe that to have 1 unit of flow going from vertex 1 to vertex $k+1$, we must have one unit flowing across each edge i. Let $\boldsymbol{\Delta}(i)$ be the voltage difference across edge i, and $\boldsymbol{f}(i)$ the flow on the edge. Then $1 = \boldsymbol{f}(i) = \frac{\boldsymbol{\Delta}(i)}{\boldsymbol{r}(i)}$, so that $\boldsymbol{\Delta}(i) = \boldsymbol{r}(i)$. The electrical voltages are then $\tilde{\boldsymbol{x}} \in \mathbb{R}^V$ where $\tilde{\boldsymbol{x}}(i) = \tilde{\boldsymbol{x}}(1) + \sum_{j<i} \boldsymbol{\Delta}(j)$. Hence the effective resistance is

$$R_{\text{eff}}(1, k+1) = \boldsymbol{d}^\top \tilde{\boldsymbol{x}} = (\boldsymbol{e}_{k+1} - \boldsymbol{e}_1)^\top \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}(k+1) - \tilde{\boldsymbol{x}}(1) = \sum_{i=1}^{k} \boldsymbol{r}(i).$$

This behavior is sometimes known as the fact that the resistance of resistors adds up when they are connected in series.

Example: Effective resistance of parallel edges. So far, we have only considered graphs with at most one edge between any two vertices. But that math also works if we allow a pair of vertices to have multiple distinct edges connecting them. We refer to this as *multi-edges*. Suppose we have a graph on just two vertices, $V = \{1, 2\}$, and these are connected by k parallel multi-edges with resistances $\boldsymbol{r}(1), \boldsymbol{r}(2), \ldots, \boldsymbol{r}(k)$.

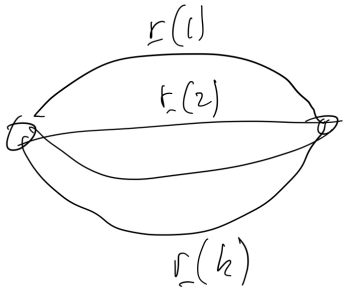


Figure 6.3: A graph on just two vertices with k parallel multiedges.

The effective resistance between the endpoints is

$$R_{\text{eff}}(1, 2) = \frac{1}{\sum_{i=1}^{k} 1/\boldsymbol{r}(i)}.$$

Let's see why. Our electrical voltages $\tilde{\boldsymbol{x}} \in \mathbb{R}^V$ can be described by just the voltage difference $\Delta \in \mathbb{R}$ between vertex 1 and vertex 2, i.e. $\tilde{\boldsymbol{x}}(2) - \tilde{\boldsymbol{x}}(1) = \Delta$. which creates a flow on edge i of $\tilde{\boldsymbol{f}}(i) = \Delta/\boldsymbol{r}(i)$. Thus the total flow from vertex 1 to vertex 2 is $1 = \sum_i \Delta/\boldsymbol{r}(i)$, so that $\Delta = \frac{1}{\sum_{i=1}^k 1/\boldsymbol{r}(i)}$. Meanwhile, the effective resistance is also

$$R_{\text{eff}}(1,2) = (\boldsymbol{e}_2 - \boldsymbol{e}_1)^\top \tilde{\boldsymbol{x}} = \Delta = \frac{1}{\sum_{i=1}^k 1/\boldsymbol{r}(i)}$$

6.3.1 Effective Resistance is a Distance

Definition 6.3.3. Consider a weighted undirected graph G with vertex set V. We say function $d : V \times V \to \mathbb{R}$, which takes a pair of vertices and returns a real number, is a *distance* if it satisfies

1. $d(a,a) = 0$ for all $a \in V$
2. $d(a,b) \geq 0$ for all $a, b \in V$.
3. $d(a,b) = d(b,a)$ for all $a, b \in V$.
4. $d(a,b) \leq d(a,c) + d(c,b)$ for all $a, b, c \in V$.

Lemma 6.3.4. *R_{eff} is a distance.*

Before proving this lemma, let's see a claim that will help us finish the proof.

Claim 6.3.5. *Let $\boldsymbol{L}\tilde{\boldsymbol{x}} = \boldsymbol{e}_b - \boldsymbol{e}_a$. Then for all $\boldsymbol{c} \in V$, we have $\tilde{\boldsymbol{x}}(b) \geq \tilde{\boldsymbol{x}}(c) \geq \tilde{\boldsymbol{x}}(a)$.*

We only sketch a proof of this claim:

Proof sketch. Consider any $c \in V$, where $c \neq a, b$. Now $(\boldsymbol{L}\tilde{\boldsymbol{x}})(c) = 0$, i.e.

$$\left(\sum_{(u,c)} \boldsymbol{w}(u,c)\right) \tilde{\boldsymbol{x}}(c) - \left(\sum_{(u,c)} \boldsymbol{w}(u,c)\tilde{\boldsymbol{x}}(u)\right) = 0$$

Rearranging $\tilde{\boldsymbol{x}}(c) = \frac{\sum_{(u,c)} \boldsymbol{w}(u,c)\tilde{\boldsymbol{x}}(u)}{\sum_{(u,c)} \boldsymbol{w}(u,c)}$. This tells us that $\tilde{\boldsymbol{x}}(c)$ is a weighted average of the voltages of its neighbors. From this, we can show that $\tilde{\boldsymbol{x}}(a)$ and $\tilde{\boldsymbol{x}}(b)$ are the extreme values. □

Proof. It is easy to check that conditions 1, 2, and 3 of Definition 6.3.3 are satisfied by R_{eff}. Let us confirm condition 4.

For any u, v, let $\tilde{\boldsymbol{x}}_{u,v} = \boldsymbol{L}^+(-\boldsymbol{e}_u + \boldsymbol{e}_v)$. Then

$$\tilde{\boldsymbol{x}}_{a,b} = \boldsymbol{L}^+(-\boldsymbol{e}_a + \boldsymbol{e}_b) = \boldsymbol{L}^+(-\boldsymbol{e}_a + \boldsymbol{e}_c - \boldsymbol{e}_c + \boldsymbol{e}_b) = \tilde{\boldsymbol{x}}_{a,c} + \tilde{\boldsymbol{x}}_{c,b}.$$

Thus,

$$\begin{aligned} R_{\text{eff}}(a,b) = (-\boldsymbol{e}_a + \boldsymbol{e}_b)^\top \tilde{\boldsymbol{x}}_{a,b} &= (-\boldsymbol{e}_a + \boldsymbol{e}_b)^\top (\tilde{\boldsymbol{x}}_{a,c} + \tilde{\boldsymbol{x}}_{c,b}) \\ &= -\tilde{\boldsymbol{x}}_{a,c}(a) + \tilde{\boldsymbol{x}}_{a,c}(b) - \tilde{\boldsymbol{x}}_{c,b}(a) + \tilde{\boldsymbol{x}}_{c,b}(b) \\ &\leq -\tilde{\boldsymbol{x}}_{a,c}(a) + \tilde{\boldsymbol{x}}_{a,c}(c) - \tilde{\boldsymbol{x}}_{c,b}(c) + \tilde{\boldsymbol{x}}_{c,b}(b). \end{aligned}$$

where in the last line we applied Claim 6.3.5 to show that $\tilde{\boldsymbol{x}}_{a,c}(b) \leq \tilde{\boldsymbol{x}}_{a,c}(c)$ and $-\tilde{\boldsymbol{x}}_{c,b}(a) \leq -\tilde{\boldsymbol{x}}_{c,b}(c)$. □

Chapter 7

Different Perspectives on Gaussian Elimination

7.1 An Optimization View of Gaussian Elimination for Laplacians

In this section, we will explore how to exactly minimize a Laplacian quadratic form by minimizing over one variable at a time. It turns out that this is in fact Gaussian Elimination in disguise – or, more precisely, the variant of Gaussian elimination that we tend to use on symmetric matrices, which is called Cholesky factorization.

Consider a Laplacian $\boldsymbol{L}$ of a connected graph $G = (V, E, \boldsymbol{w})$, where $\boldsymbol{w} \in \mathbb{R}^E$ is a vector of positive edge weights. Let $\boldsymbol{W} \in \mathbb{R}^{E \times E}$ be the diagonal matrix with the edge weights on the diagonal, i.e. $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w})$ and $\boldsymbol{L} = \boldsymbol{B}\,\boldsymbol{W}\,\boldsymbol{B}^\top$. Let $\boldsymbol{d} \in \mathbb{R}^V$ be a demand vector s.t. $\boldsymbol{d} \perp \mathbf{1}$.

Let us define an energy

$$\mathcal{E}(\boldsymbol{x}) = -\boldsymbol{d}^\top \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x}$$

Note that this function is convex and is minimized at $\boldsymbol{x}$ s.t. $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$.

We will now explore an approach to solving the minimization problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^V} \mathcal{E}(\boldsymbol{x})$$

Let $\boldsymbol{x} = \begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix}$ where $y \in \mathbb{R}$ and $\boldsymbol{z} \in \mathbb{R}^{V \setminus \{1\}}$.

We will now explore how to minimize over y, given any $\boldsymbol{z}$. Once we find an expression for y in terms of $\boldsymbol{z}$, we will be able to reduce it to a new quadratic minimization problem in $\boldsymbol{z}$,

$$\mathcal{E}'(\boldsymbol{z}) = -\boldsymbol{d}'^\top \boldsymbol{z} + \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{L}'\boldsymbol{z}$$

where $\boldsymbol{d}'$ is a demand vector on the remaining vertices, with $\boldsymbol{d} \perp \mathbf{1}$ and $\boldsymbol{L}'$ is a Laplacian of a graph on the remaining vertices $V' = V \setminus \{1\}$. We can then repeat the procedure to eliminate another variable and so on. Eventually, we can then find all the solution to our original minimization problem.

To help us understand how to minimize over the first variable, we introduce some notation for the first row and column of the Laplacian:

$$\boldsymbol{L} = \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-1} \end{pmatrix}. \tag{7.1}$$

Note that W is the weighted degree of vertex 1, and that

$$\begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} \tag{7.2}$$

is the Laplacian of the subgraph of G containing only the edges incident on vertex 1, while $\boldsymbol{L}_{-1}$ is the Laplacian of the subgraph of G containing all edges *not* incident on vertex 1.

Let us also write $\boldsymbol{d} = \begin{pmatrix} b \\ \boldsymbol{c} \end{pmatrix}$ where $b \in \mathbb{R}$ and $\boldsymbol{c} \in \mathbb{R}^{V \setminus \{1\}}$.

Now,

$$\begin{aligned} \mathcal{E}(\boldsymbol{x}) &= -\boldsymbol{d}^\top \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} = -\begin{pmatrix} b \\ \boldsymbol{c} \end{pmatrix}^\top \begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix} + \frac{1}{2}\begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix}^\top \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-1} \end{pmatrix} \begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix} \\ &= -by - \boldsymbol{c}^\top \boldsymbol{z} + \frac{1}{2}\left(y^2 W - 2y\boldsymbol{a}^\top \boldsymbol{z} + \boldsymbol{z}^\top \operatorname{diag}(\boldsymbol{a})\boldsymbol{z} + \boldsymbol{z}^\top \boldsymbol{L}_{-1}\boldsymbol{z}\right). \end{aligned}$$

Now, to minimize over y, we set $\frac{\partial}{\partial y}\mathcal{E}(\boldsymbol{x}) = 0$ and get

$$-b + yW - \boldsymbol{a}^\top \boldsymbol{z} = 0.$$

Solving for y, we get that the minimizing y is

$$y = \frac{1}{W}(b + \boldsymbol{a}^\top \boldsymbol{z}). \tag{7.3}$$

Observe that

$$\begin{aligned} \mathcal{E}(\boldsymbol{x}) &= -by - \boldsymbol{c}^\top \boldsymbol{z} + \frac{1}{2}\left(y^2 W - 2y\boldsymbol{a}^\top \boldsymbol{z} + \boldsymbol{z}^\top \operatorname{diag}(\boldsymbol{a})\boldsymbol{z} + \boldsymbol{z}^\top \boldsymbol{L}_{-1}\boldsymbol{z}\right) \\ &= -by - \boldsymbol{c}^\top \boldsymbol{z} + \frac{1}{2}\left(\frac{1}{W}(yW - \boldsymbol{a}^\top \boldsymbol{z})^2 \underbrace{-\frac{1}{W}\boldsymbol{z}^\top \boldsymbol{a}\boldsymbol{a}^\top \boldsymbol{z} + \boldsymbol{z}^\top \operatorname{diag}(\boldsymbol{a})\boldsymbol{z} + \boldsymbol{z}^\top \boldsymbol{L}_{-1}\boldsymbol{z}}_{\text{Let } \boldsymbol{S} = \operatorname{diag}(\boldsymbol{a}) - \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top + \boldsymbol{L}_{-1}}\right) \\ &= -by - \boldsymbol{c}^\top \boldsymbol{z} + \frac{1}{2}\left(\frac{1}{W}(yW - \boldsymbol{a}^\top \boldsymbol{z})^2 + \boldsymbol{z}^\top \boldsymbol{S}\boldsymbol{z}\right), \end{aligned}$$

where we simplified the expression by defining $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{a}) - \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top + \boldsymbol{L}_{-1}$. Plugging in $y = \frac{1}{W}(b + \boldsymbol{a}^\top \boldsymbol{z})$, we get

$$\min_y \mathcal{E}\begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix} = -\left(\boldsymbol{c} + b\frac{1}{W}\boldsymbol{a}\right)^\top \boldsymbol{z} - \frac{b^2}{2W} + \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{S}\boldsymbol{z}.$$

Now, we define $\boldsymbol{d}' = \boldsymbol{c} + b\frac{1}{W}\boldsymbol{a}$ and $\mathcal{E}'(\boldsymbol{z}) = -\boldsymbol{d}'^\top \boldsymbol{z} + \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{S}\boldsymbol{z}$. And, we can see that

$$\arg\min_{\boldsymbol{z}} \min_y \mathcal{E}\begin{pmatrix} y \\ \boldsymbol{z} \end{pmatrix} = \arg\min_{\boldsymbol{z}} \mathcal{E}'(\boldsymbol{z}),$$

since dropping the constant term $-\frac{b^2}{2W}$ does not change what the minimizing $\boldsymbol{z}$ values are.

Claim 7.1.1.

1. $\boldsymbol{d}' \perp \mathbf{1}$
2. $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{a}) - \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top + \boldsymbol{L}_{-1}$ *is a Laplacian of a graph on the vertex set* $V \setminus \{1\}$.

We will prove Claim 7.1.1 in a moment. From the Claim, we see that the problem of finding $\arg\min_{\boldsymbol{z}} \mathcal{E}'(\boldsymbol{z})$, is exactly of the same form as finding $\arg\min_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x})$, but with one fewer variables.

We can get a minimizing $\boldsymbol{x}$ that solves $\arg\min_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x})$ by repeating the variable elimination procedure until we get down to a single variable and finding its value. We then have to work back up to getting a solution for $\boldsymbol{z}$, and then substitute that into Equation (7.3) to get the value for y.

Remark 7.1.2. In fact, this perspective on Gaussian elimination also makes sense for any positive definite matrix. In this setting, minimizing over one variable will leave us with another positive definite quadratic minimization problem.

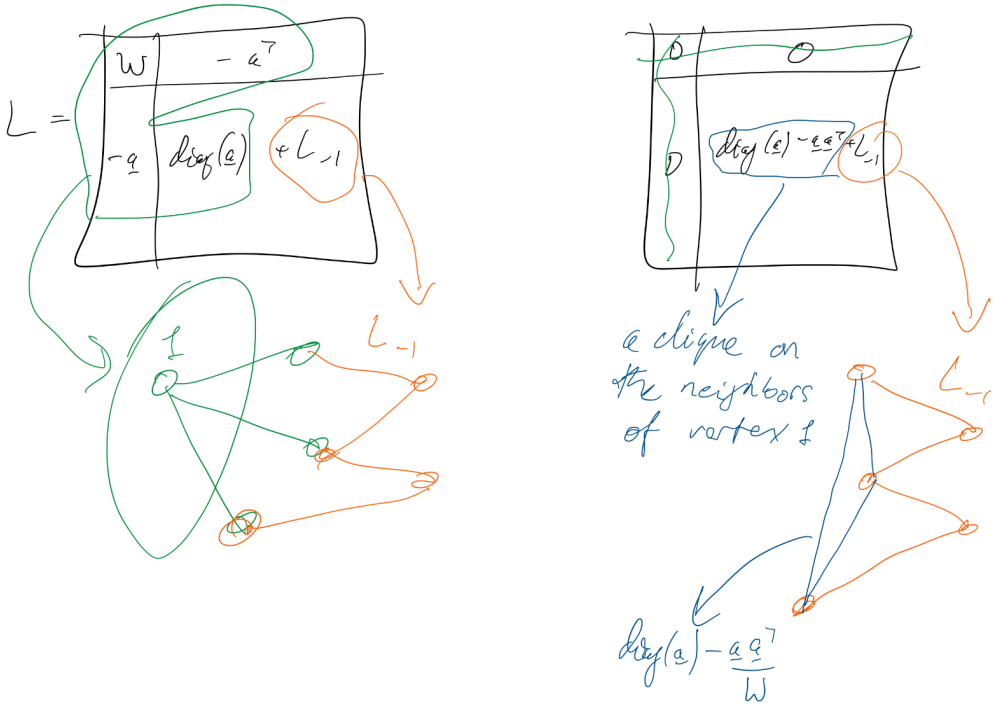
Proof of Claim 7.1.1. To establish the first part, we note that $\mathbf{1}^\top \boldsymbol{d}' = \mathbf{1}^\top \boldsymbol{c} + b\frac{\mathbf{1}^\top \boldsymbol{a}}{W} = \mathbf{1}^\top \boldsymbol{c} + b = \mathbf{1}^\top \boldsymbol{d} = 0$. To establish the second part, we notice that $\boldsymbol{L}_{-1}$ is a graph Laplacian by definition. Since the sum of two graph Laplacians is another graph Laplacian, it now suffices to show that $\boldsymbol{S}$ is a graph Laplacian.

Claim 7.1.3. *A matrix* $\boldsymbol{M}$ *is a graph Laplacian if and only it satisfies the following conditions:*

- $\boldsymbol{M}^\top = \boldsymbol{M}$.
- *The diagonal entries of* $\boldsymbol{M}$ *are non-negative, and the off-diagonal entries of* $\boldsymbol{M}$ *are non-positive.*
- $\boldsymbol{M}\mathbf{1} = \mathbf{0}$.

Let's see that Claim 7.1.3 is true. Firstly, when the conditions hold we can write $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$ where $\boldsymbol{D}$ is diagonal and non-negative, and $\boldsymbol{A}$ is non-negative, symmetric, and zero on the diagonal, and from the last condition $\boldsymbol{D}(i,i) = \sum_{j \neq i} \boldsymbol{A}(i,j)$. Thus we can view $\boldsymbol{A}$ as a graph adjacency matrix and $\boldsymbol{D}$ as the corresponding diagonal matrix of weighted degrees. Secondly, it is easy to check that the conditions hold for any graph Laplacian, so the conditions indeed hold if and only if. Now we have to check that the claim applies to $\boldsymbol{S}$. We leave this as an exercise for the reader.

Finally, we want to argue that the graph corresponding to $\boldsymbol{S}$ is connected. Consider any $i, j \in V \setminus \{1\}$. Since G, the graph of $\boldsymbol{L}$, is connected, there exists a simple path in G connecting i and j. If this path does not use vertex 1, it is a path in the graph of $\boldsymbol{L}_{-1}$ and hence in the graph of $\boldsymbol{S}$. If the path does use vertex 1, it must do so by reaching the vertex on some edge $(v, 1)$ and leaving on a different edge $(1, u)$. Replace this pair of edges with edge (u, v), which appears in the graph of $\boldsymbol{S}$ because $\boldsymbol{S}(u, v) < 0$. Now we have a path in the graph of $\boldsymbol{S}$. □



7.2 An Additive View of Gaussian Elimination

Cholesky decomposition basics. Again we consider a graph Laplacian $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ of a connected graph $G = (V, E, \boldsymbol{w})$, where as usual $|V| = n$ and $|E| = m$.

In this Section, we'll study how to decompose a graph Laplacian as $\boldsymbol{L} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$, where $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix, i.e. $\boldsymbol{\mathcal{L}}(i,j) = 0$ for $i < j$. Such a factorization is called a Cholesky decomposition. It is essentially the result of Gaussian elimination with a slight twist to ensure the matrices maintained at intermediate steps of the algorithm remain symmetric.

We use $\mathrm{nnz}(\boldsymbol{A})$ to denote the number of non-zero entries of matrix $\boldsymbol{A}$.

Lemma 7.2.1. *Given an* invertible *square lower triangular matrix $\boldsymbol{\mathcal{L}}$, we can solve the linear equation $\boldsymbol{\mathcal{L}}\boldsymbol{y} = \boldsymbol{b}$ in time $O(\mathrm{nnz}(\boldsymbol{\mathcal{L}}))$. Similarly, given an upper triangular matrix $\boldsymbol{\mathcal{U}}$, we can solve linear equations $\boldsymbol{\mathcal{U}}\boldsymbol{z} = \boldsymbol{c}$ in time $O(\mathrm{nnz}(\boldsymbol{\mathcal{U}}))$.*

We omit the proof, which is a straight-forward exercise. The algorithms for solving linear equations in upper and lower triangular matrices are known as forward and back substitution respectively.

Remark 7.2.2. Strictly speaking, the lemma requires us to have access an adjacency list representation of $\boldsymbol{\mathcal{L}}$ so that we can quickly tell where the non-zero entries are.

Using forward and back substitution, if we have a decomposition of an invertible matrix $\boldsymbol{M}$ into $\boldsymbol{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$, we can now solve linear equations in $\boldsymbol{M}$ in time $O(\mathrm{nnz}(\boldsymbol{\mathcal{L}}))$.

Remark 7.2.3. We have learned about decompositions using a lower triangular matrix, and later we will see an algorithm for computing these. In fact, we can have more flexibility than that. From an algorithmic perspective, it is sufficient that there exists a permutation matrix $\boldsymbol{P}$ s.t. $\boldsymbol{P}\boldsymbol{\mathcal{L}}\boldsymbol{P}^\top$ is lower triangular. If we know the ordering under which the matrix becomes lower triangular, we can perform substitution according to that order to solve linear equations in the matrix without having to explicitly apply a permutation to the matrix.

Dealing with pseudoinverses. But how can we solve a linear equation in $\boldsymbol{L} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$, where $\boldsymbol{L}$ is not invertible? For graph Laplacians we have a simple characterization of the kernel, and because of this, dealing with the lack of invertibility turns out to be fairly easy.

We can use the following lemma which you will prove in an exercise next week.

Lemma 7.2.4. *Consider a real symmetric matrix $\boldsymbol{M} = \boldsymbol{X}\boldsymbol{Y}\boldsymbol{X}^\top$, where $\boldsymbol{X}$ is real and invertible and $\boldsymbol{Y}$ is real symmetric. Let $\boldsymbol{\Pi}_{\boldsymbol{M}}$ denote the orthogonal projection to the image of $\boldsymbol{M}$. Then $\boldsymbol{M}^+ = \boldsymbol{\Pi}_{\boldsymbol{M}}(\boldsymbol{X}^\top)^{-1}\boldsymbol{Y}^+\boldsymbol{X}^{-1}\boldsymbol{\Pi}_{\boldsymbol{M}}$.*

The factorizations $\boldsymbol{L} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$ that we produce will have the property that all diagonal entries of $\boldsymbol{\mathcal{L}}$ are strictly non-zero, except that $\boldsymbol{\mathcal{L}}(n,n) = 0$. Let $\widehat{\boldsymbol{\mathcal{L}}}$ be the matrix whose entries agree with $\boldsymbol{\mathcal{L}}$, except that $\widehat{\boldsymbol{\mathcal{L}}}(n,n) = 1$. Let $\boldsymbol{\mathcal{D}}$ be the diagonal matrix with $\boldsymbol{\mathcal{D}}(i,i) = 1$ for $i < n$ and $\boldsymbol{\mathcal{D}}(n,n) = 0$. Then $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top = \widehat{\boldsymbol{\mathcal{L}}}\boldsymbol{\mathcal{D}}\widehat{\boldsymbol{\mathcal{L}}}^\top$, and $\widehat{\boldsymbol{\mathcal{L}}}$ is invertible, and $\boldsymbol{\mathcal{D}}^+ = \boldsymbol{\mathcal{D}}$. Finally, $\boldsymbol{\Pi}_{\boldsymbol{L}} = \boldsymbol{I} - \frac{1}{n}\boldsymbol{1}\boldsymbol{1}^\top$, because this matrix is acts like identity on vectors orthogonal to $\boldsymbol{1}$ and ensures $\boldsymbol{\Pi}_{\boldsymbol{L}}\boldsymbol{1} = \boldsymbol{0}$, and this matrix can be applied to a vector in $O(n)$ time. Thus $\boldsymbol{L}^+ = \boldsymbol{\Pi}_{\boldsymbol{L}}(\widehat{\boldsymbol{\mathcal{L}}}^\top)^{-1}\boldsymbol{\mathcal{D}}\widehat{\boldsymbol{\mathcal{L}}}^{-1}\boldsymbol{\Pi}_{\boldsymbol{L}}$, and this matrix can be applied in time $O(\mathrm{nnz}(\boldsymbol{\mathcal{L}}))$.

An additive view of Gaussian Elimination. The following theorem describes Gaussian Elimination / Cholesky decompostion of a graph Laplacian.

Theorem 7.2.5 (Cholesky Decomposition on graph Laplacians)**.** *Let $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ be a graph Laplacian of a connected graph $G = (V, E, \boldsymbol{w})$, where $|V| = n$. Using Gaussian Elimination, we can compute in $O(n^3)$ time a factorization $\boldsymbol{L} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$ where $\boldsymbol{\mathcal{L}}$ is lower triangular, and has positive diagonal entries except $\boldsymbol{\mathcal{L}}(n, n) = 0$.*

Proof. Let $\boldsymbol{L}^{(0)} = \boldsymbol{L}$. We will use $\boldsymbol{A}(:, i)$ to denote the the ith column of a matrix $\boldsymbol{A}$. Now, for $i = 1$ to $i = n - 1$ we define

$$\boldsymbol{l}_i = \frac{1}{\sqrt{\boldsymbol{L}^{(i-1)}(i,i)}}\boldsymbol{L}^{(i-1)}(:, i) \text{ and } \boldsymbol{L}^{(i)} = \boldsymbol{L}^{(i-1)} - \boldsymbol{l}_i\boldsymbol{l}_i^\top$$

Finally, we let $\boldsymbol{l}_n = \boldsymbol{0}_{n\times 1}$. We will show later that

$$\boldsymbol{L}^{(n-1)} = \boldsymbol{0}_{n\times n}. \tag{7.4}$$

It follows that $\boldsymbol{L} = \sum_i \boldsymbol{l}_i\boldsymbol{l}_i^\top$, provided this procedure is well-defined, i.e. $\boldsymbol{L}^{(i-1)}(i, i) \neq 0$ for all $i < n$. We will sketch a proof of this later, while also establishing several other properties of the procedure.

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $U \subseteq [n]$, we will use $\boldsymbol{A}(U, U)$ to denote the principal submatrix of $\boldsymbol{A}$ obtained by restricting to the rows and columns with index in U, i.e. all entries $\boldsymbol{A}(i, j)$ where $i, j \in U$.

Claim 7.2.6. *Fix some $i < n$. Let $U = \{i + 1, \ldots, n\}$. Then $\boldsymbol{L}^{(i)}(i, j) = 0$ if $i \notin U$ or $j \notin U$. And $\boldsymbol{L}^{(i)}(U, U)$ is a graph Laplacian of a connected graph on the vertex set U.*

From this claim, it follows that $\boldsymbol{L}^{(i-1)}(i, i) \neq 0$ for $i < n$, since a connected graph Laplacian on a graph with $|U| > 1$ vertices cannot have a zero on the diagonal, and it follows that $\boldsymbol{L}^{(n-1)}(i, i) = 0$, because the only graph we allow on one vertex is the empty graph. This shows Equation (7.4) holds. □

Sketch of proof of Claim 7.2.6. We will focus on the first elimination, as the remaining are similar. Adopting the same notation as in Equation (7.1), we write

$$\boldsymbol{L}^{(0)} = \boldsymbol{L} = \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-1} \end{pmatrix}$$

and, noting that

$$\boldsymbol{l}_1\boldsymbol{l}_1^\top = \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top \end{pmatrix}$$

we see that

$$\boldsymbol{L}^{(1)} = \boldsymbol{L}^{(0)} - \boldsymbol{l}_1\boldsymbol{l}_1^\top = \begin{pmatrix} 0 & \boldsymbol{0} \\ \boldsymbol{0} & \operatorname{diag}(\boldsymbol{a}) - \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top + \boldsymbol{L}_{-1} \end{pmatrix}.$$

Thus the first row and column of $\boldsymbol{L}^{(1)}$ are zero claimed. It also follows by Claim 7.1.1 that $\boldsymbol{L}^{(1)}(\{2,\ldots,n\},\{2,\ldots,n\})$ is the Laplacian of a connected graph. This proves Claim 7.2.6 for the case $i = 1$. An induction following the same pattern can be used to prove the claim for all $i < n$. □

Chapter 8

Random Matrix Concentration and Spectral Graph Sparsification

8.1 Matrix Sampling and Approximation

We want to begin understanding how sums of random matrices behave, in particular, whether they exhibit a tendency to concentrate in the same way that sum of scalar random variables do under various conditions.

First, let's recall a scalar Chernoff bound, which shows that a sum of bounded, non-negative random variables tend to concentrate around their mean.

Theorem 8.1.1 (A Chernoff Concentration Bound)**.** *Suppose $X_1, \ldots, X_k \in \mathbb{R}$ are independent, non-negative, random variables with $X_i \leq R$ always. Let $X = \sum_i X_i$, and $\mu = \mathbb{E}[X]$, then for $0 < \epsilon \leq 1$*

$$\Pr[X \geq (1+\epsilon)\mu] \leq \exp\left(\frac{-\epsilon^2\mu}{4R}\right) \text{ and } \Pr[X \leq (1-\epsilon)\mu] \leq \exp\left(\frac{-\epsilon^2\mu}{4R}\right).$$

The Chernoff bound should be familiar to most of you, but you may not have seen the following very similar bound. The Bernstein bound, which we will state in terms of zero-mean variables, is much like the Chernoff bound. It also requires bounded variables. But, when the variables have small variance, the Bernstein bound is sometimes stronger.

Theorem 8.1.2 (A Bernstein Concentration Bound)**.** *Suppose $X_1, \ldots, X_k \in \mathbb{R}$ are independent, zero-mean, random variables with $|X_i| \leq R$ always. Let $X = \sum_i X_i$, and $\sigma^2 = Var[X] = \sum_i \mathbb{E}[X_i^2]$, then for $t > 0$*

$$\Pr[|X| \geq t] \leq 2\exp\left(\frac{-t^2}{2Rt + 4\sigma^2}\right).$$

We will now prove the Bernstein concentration bound for scalar random variables, as a warm-up to the next section, where we will prove a version of it for matrix-valued random variables.

To help us prove Bernstein's bound, first let's recall Markov's inequality. This is a very weak concentration inequality, but also very versatile, because it requires few assumptions.

Lemma 8.1.3 (Markov's Inequality)**.** *Suppose $\boldsymbol{X} \in \mathbb{R}$ is a non-negative random variable, with a finite expectation. Then for any $t > 0$,*

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Proof.

$$\begin{aligned}\mathbb{E}[X] &= \Pr[X \geq t]\,\mathbb{E}[X \mid X \geq t] + \Pr[X < t]\,\mathbb{E}[X \mid X < t] \\ &\geq \Pr[X \geq t]\,\mathbb{E}[X \mid X \geq t] \\ &\geq Pr[X \geq t] \cdot t.\end{aligned}$$

We can rearrange this to get the desired statement. □

Now, we are ready to prove Bernstein's bound.

Proof of Theorem 8.1.2. We will focus on bounding the probability that $\Pr[X \geq t]$. The proof that $\Pr[-X \geq t]$ is small proceeds in the same way.

First we observe that

$$\begin{aligned}\Pr[X \geq t] &= \Pr[\exp(\theta X) \geq \exp(\theta t)] \\ &\qquad \text{for any } \theta > 0, \text{ because } x \mapsto \exp(\theta x) \text{ is strictly increasing.} \\ &\leq \exp(-\theta t)\,\mathbb{E}[\exp(\theta X)] \qquad \text{by Lemma 8.1.3 (Markov's Inequality).}\end{aligned}$$

Now, let's require that $\theta \leq 1/R$ This will allow us to use the following bound: For all $|z| \leq 1$,

$$\exp(z) \leq 1 + z + z^2. \tag{8.1}$$

We omit a proof of this, but the plots in Figure 8.1 suggest that this upper bound holds. The reader should consider how to prove this. With this in mind, we see that

$$\begin{aligned}\mathbb{E}[\exp(\theta X)] &= \mathbb{E}\left[\exp\left(\theta \sum_i X_i\right)\right] \\ &= \mathbb{E}[\Pi_i \exp(\theta X_i)] \\ &= \Pi_i\,\mathbb{E}[\exp(\theta X_i)] \quad \text{because } \mathbb{E}[YZ] = \mathbb{E}[Y]\,\mathbb{E}[Z] \text{ for independent } Y \text{ and } Z. \\ &\leq \Pi_i\,\mathbb{E}\left[1 + \theta X_i + (\theta X_i)^2\right] \\ &= \Pi_i(1 + \theta^2\,\mathbb{E}\left[X_i^2\right]) \qquad \text{because } X_i \text{ are zero-mean.} \\ &\leq \Pi_i \exp(\theta^2\,\mathbb{E}\left[X_i^2\right]) \qquad \text{because } 1 + z \leq \exp(z) \text{ for all } z \in R. \\ &= \exp\left(\sum_i \theta^2\,\mathbb{E}\left[X_i^2\right]\right) = \exp(\theta^2\sigma^2).\end{aligned}$$

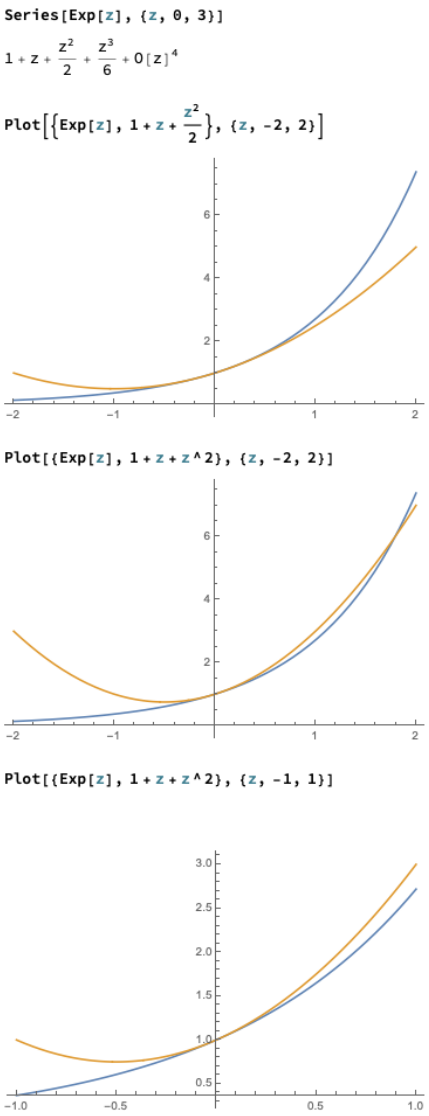


Figure 8.1: Plotting $\exp(z)$ compared to $1 + z + z^2$.

Thus $\Pr[X \geq t] \leq \exp(-\theta t)\,\mathbb{E}\left[\exp(\theta X)\right] \leq \exp(-\theta t + \theta^2\sigma^2)$. Now, to get the best possible bound, we'd like to minimize $-\theta t + \theta^2\sigma^2$ subject to the constraint $0 < \theta \leq 1/R$. Setting

$$\frac{\partial}{\partial\theta}\left(-\theta t + \theta^2\sigma^2\right) = -t + 2\theta\sigma^2.$$

Setting this derivative to zero gives $\theta = \frac{t}{2\sigma^2}$, and plugging that in gives

$$-\theta t + \theta^2\sigma^2 = -\frac{t^2}{4\sigma^2}$$

This choice only satisfies our constraints on θ if $\frac{t}{2\sigma^2} \leq 1/R$. Otherwise, we let $\theta = 1/R$ and note that in this case

$$-\theta t + \theta^2\sigma^2 = -\frac{t}{R} + \frac{\sigma^2}{R^2} \leq -\frac{t}{R} + \frac{t}{2R} = -\frac{t}{2R}$$

where we got the inequality from $t > 2\sigma^2/R$. Altogether, we can conclude that there always is a choice of θ s.t.

$$-\theta t + \theta^2\sigma^2 \leq -\min\left(\frac{t}{2R}, \frac{t^2}{4\sigma^2}\right) \leq -\frac{t^2}{2Rt + 4\sigma^2}.$$

In fact, with the benefit of hindsight, and a little algebra, we arrive at the same conclusion in another way: One can check that the following choice of θ is always valid and achives the same bound: $\theta = \frac{1}{2\sigma^2}\left(t - \frac{\sqrt{R}\cdot t^{3/2}}{\sqrt{2\sigma^2 + Rt}}\right)$. □

We use $\|\cdot\|$ to denote the spectral norm on matrices. Let's take a look at a version of Bernstein's bound that applies to sums of random matrices.

Theorem 8.1.4 (A Bernstein Matrix Concentration Bound (Tropp 2011))**.** *Suppose $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k \in \mathbb{R}^{n\times n}$ are independent, symmetric matrix-valued random variables. Assume each $\boldsymbol{X}_i$ is zero-mean, i.e. $\mathbb{E}[\boldsymbol{X}_i] = \mathbf{0}_{n\times n}$, and that $\|\boldsymbol{X}_i\| \leq R$ always. Let $\boldsymbol{X} = \sum_i \boldsymbol{X}_i$, and $\sigma^2 = Var[\boldsymbol{X}] = \sum_i \mathbb{E}\left[\boldsymbol{X}_i^2\right]$, then for $t > 0$*

$$\Pr[\|\boldsymbol{X}\| \geq t] \leq 2n \exp\left(\frac{-t^2}{2Rt + 4\sigma^2}\right).$$

This basically says that probability of $\boldsymbol{X}$ being large in spectral norm behaves like the scalar case, except the bound is larger by a factor n, where the matrices are $n \times n$. We can get a feeling for why this might be a reasonable bound by considering the case of random diagonal matrices. Now $\|\boldsymbol{X}\| = \max_j |\boldsymbol{X}(j,j)| = \max_j |\sum_i \boldsymbol{X}_i(j,j)|$. In this case, we need to bound the largest of the n diagonal entries: We can do this by a union bound over n instances of the scalar problem – and this also turns out to be essentially tight in some cases, meaning we can't expect a better bound in general.

8.2 Matrix Concentration

In this section we will prove the Bernstein matrix concentration bound (Tropp 2011) that we saw in the previous section (Theorem 8.1.4).

Theorem 8.2.1. *Suppose $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k \in \mathbb{R}^{n\times n}$ are independent, symmetric matrix-valued random variables. Assume each $\boldsymbol{X}_i$ is zero-mean, i.e. $\mathbb{E}[\boldsymbol{X}_i] = \mathbf{0}_{n\times n}$, and that $\|\boldsymbol{X}_i\| \leq R$ always. Let $\boldsymbol{X} = \sum_i \boldsymbol{X}_i$, and $\sigma^2 = \| Var[\boldsymbol{X}] \| = \| \sum_i \mathbb{E}\left[\boldsymbol{X}_i^2\right] \|$, then for $t > 0$*

$$\Pr[\|\boldsymbol{X}\| \geq t] \leq 2n \exp\left(\frac{-t^2}{2Rt + 4\sigma^2}\right).$$

But let's collect some useful tools for the proof first.

Definition 8.2.2 (trace)**.** The trace of a square matrix $\boldsymbol{A}$ is defined as

$$\operatorname{Tr}(\boldsymbol{A}) := \sum_i \boldsymbol{A}(i,i)$$

Claim 8.2.3 (cyclic property of trace)**.** $\operatorname{Tr}(\boldsymbol{A}\boldsymbol{B}) = \operatorname{Tr}(\boldsymbol{B}\boldsymbol{A})$

Let S^n denote the set of all $n \times n$ real symmetric matrices, S^n_+ the set of all $n \times n$ positive semidefinite matrices, and S^n_{++} the set of all $n \times n$ positive definite matrices. Their relation is clear, $S^n_{++} \subset S^n_+ \subset S^n$. For any $\boldsymbol{A} \in S^n$ with eigenvalues $\lambda_1(\boldsymbol{A}) \leq \cdots \leq \lambda_n(\boldsymbol{A})$, by spectral decomposition theorem, $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$ where $\boldsymbol{\Lambda} = \operatorname{diag}_i\{\lambda_i(\boldsymbol{A})\}$ and $\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^\top = \boldsymbol{I}$, we'll use this property without specifying in the sequel.

Claim 8.2.4. *Given a symmetric and real matrix* $\boldsymbol{A}$, $\operatorname{Tr}(\boldsymbol{A}) = \sum_i \lambda_i$, *where* $\{\lambda_i\}$ *are eigenvalues of* A.

Proof.

$$\operatorname{Tr}(\boldsymbol{A}) = \operatorname{Tr}\left(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top\right) = \operatorname{Tr}\left(\boldsymbol{\Lambda}\underbrace{\boldsymbol{V}^\top\boldsymbol{V}}_{\boldsymbol{I}}\right) = \operatorname{Tr}(\boldsymbol{\Lambda}) = \sum_i \lambda_i.$$

□

8.2.1 Matrix Functions

Definition 8.2.5 (Matrix function)**.** Given a real-valued function $f : \mathbb{R} \to \mathbb{R}$, we extend it to a matrix function $f : S^n \to S^n$. For $\boldsymbol{A} \in S^n$ with spectral decomposition $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$, let

$$f(\boldsymbol{A}) = \boldsymbol{V} \operatorname*{diag}_i \{f(\lambda_i)\}\, \boldsymbol{V}^\top.$$

Example. Recall that every PSD matrix $\boldsymbol{A}$ has a square root $\boldsymbol{A}^{1/2}$. If $f(x) = x^{1/2}$ for $x \in \mathbb{R}_+$, then $f(\boldsymbol{A}) = \boldsymbol{A}^{1/2}$ for $\boldsymbol{A} \in S^n_+$.

Example. If $f(x) = \exp(x)$ for $x \in \mathbb{R}$, then $f(\boldsymbol{A}) = \exp(\boldsymbol{A}) = \boldsymbol{V}\exp(\boldsymbol{\Lambda})\boldsymbol{V}^\top$ for $\boldsymbol{A} \in S^n$. Note that $\exp(\boldsymbol{A})$ is positive definite for any $\boldsymbol{A} \in S^n$.

8.2.2 Monotonicity and Operator Monotonicity

Cosider a function $f : \mathcal{D} \to \mathcal{C}$. If we have a partial order $\leq_{\mathcal{D}}$ defined on $\mathcal{D}$ and a partial order $\leq_{\mathcal{C}}$ defined on $\mathcal{C}$, then we say that the function is monotone increasing (resp. decreasing) w.r.t. this pair of orderings if for all $d_1, d_2 \in \mathcal{D}$ s.t. $d_1 \leq_{\mathcal{D}} d_2$ we have $f(d_1) \leq_{\mathcal{C}} f(d_2)$ (resp. decreasing if $f(d_2) \leq_{\mathcal{C}} f(d_1)$).

Let's introduce some terminonology for important special cases of this idea. We say that a function $f : \mathcal{S} \to \mathbb{R}$, where $\mathcal{S} \subseteq S^n$, is monotone increasing if $\boldsymbol{A} \preceq \boldsymbol{B}$ implies $f(\boldsymbol{A}) \leq f(\boldsymbol{B})$.

Meanwhile, a function $f : \mathcal{S} \to \mathcal{T}$ where $\mathcal{S}, \mathcal{T} \subseteq S^n$ is said to be operator monotone increasing if $\boldsymbol{A} \preceq \boldsymbol{B}$ implies $f(\boldsymbol{A}) \preceq f(\boldsymbol{B})$.

Lemma 8.2.6. *Let $T \subseteq \mathbb{R}$. If the scalar function $f : T \to \mathbb{R}$ is monotone increasing, the matrix function $\boldsymbol{X} \mapsto \operatorname{Tr}(f(\boldsymbol{X}))$ is monotone increasing.*

Proof. From previous chapters, we know if $\boldsymbol{A} \preceq \boldsymbol{B}$ then $\lambda_i(\boldsymbol{A}) \leq \lambda_i(\boldsymbol{B})$ for all i. As $x \mapsto f(x)$ is monotone, then $\lambda_i(f(\boldsymbol{A})) \leq \lambda_i(f(\boldsymbol{B}))$ for all i. By Claim 8.2.4, $\operatorname{Tr}(f(\boldsymbol{A})) \leq \operatorname{Tr}(f(\boldsymbol{B}))$. □

From this, and the fact that $x \mapsto \exp(x)$ is a monotone function on the reals, we get the following corollary.

Corollary 8.2.7. *If $\boldsymbol{A} \preceq \boldsymbol{B}$, then $\operatorname{Tr}(\exp(\boldsymbol{A})) \leq \operatorname{Tr}(\exp(\boldsymbol{B}))$, i.e. $\boldsymbol{X} \mapsto \operatorname{Tr}(\exp(\boldsymbol{X}))$ is monotone increasing.*

Lemma 8.2.8. *If $\boldsymbol{0} \prec \boldsymbol{A} \preceq \boldsymbol{B}$, then $\boldsymbol{B}^{-1} \preceq \boldsymbol{A}^{-1}$, i.e. $\boldsymbol{X} \mapsto \boldsymbol{X}^{-1}$ is operator monotone decreasing on S^n_{++}.*

You will prove the above lemma in this week's exercises.

Lemma 8.2.9. *If $0 \prec \boldsymbol{A} \preceq \boldsymbol{B}$, then $\log(\boldsymbol{A}) \preceq \log(\boldsymbol{B})$.*

To prove this lemma, we first recall an integral representation of the logarithm.

Lemma 8.2.10.

$$\log a = \int_0^\infty \left(\frac{1}{1+t} - \frac{1}{a+t} \right) \mathrm{d}t$$

Proof.

$$\begin{aligned}
\int_0^\infty \left(\frac{1}{1+t} - \frac{1}{a+t} \right) \mathrm{d}t &= \lim_{T \to \infty} \int_0^T \left(\frac{1}{1+t} - \frac{1}{a+t} \right) \mathrm{d}t \\
&= \lim_{T \to \infty} \left[\log(1+t) - \log(a+t) \right]_0^T \\
&= \log(a) + \lim_{T \to \infty} \log \left(\frac{1+T}{a+T} \right) \\
&= \log(a)
\end{aligned}$$

□

Proof sketch of Lemma 8.2.9. Because all the matrices involved are diagonalized by the same orthogonal transformation, we can conclude from Lemma 8.2.10 that for a matrix $\boldsymbol{A} \succ \boldsymbol{0}$,

$$\log(\boldsymbol{A}) = \int_0^\infty \left(\frac{1}{1+t} \boldsymbol{I} - (t\boldsymbol{I} + \boldsymbol{A})^{-1} \right) \mathrm{d}t$$

This integration can be expressesd as the limit of a sum with positive coefficients, and from this we can show that is the integrand (the term inside the integration symbol) is operator monotone increasing in $\boldsymbol{A}$ by Lemma 8.2.8, the result of the integral, i.e. $\log(\boldsymbol{A})$ must also be operator monotone increasing. □

The following is a more general version of Lemma 1.6.

Lemma 8.2.11. *Let $T \subset \mathbb{R}$. If the scalar function $f : T \to \mathbb{R}$ is monotone, the matrix function $\boldsymbol{X} \mapsto \operatorname{Tr}(f(\boldsymbol{X}))$ is monotone.*

Remark 8.2.12. It is not always true that when $f : \mathbb{R} \to \mathbb{R}$ is monotone, $f : S^n \to S^n$ is operator monotone. For example, $\boldsymbol{X} \mapsto \boldsymbol{X}^2$ and $\boldsymbol{X} \mapsto \exp(\boldsymbol{X})$ are *not* operator monotone.

8.2.3 Some Useful Facts

Lemma 8.2.13. $\exp(\boldsymbol{A}) \preceq \boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2$ *for* $\|\boldsymbol{A}\| \leq 1$.

Proof.

$$\begin{aligned}\boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 - \exp(\boldsymbol{A}) &= \boldsymbol{V}\boldsymbol{I}\boldsymbol{V}^\top + \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top + \boldsymbol{V}\boldsymbol{\Lambda}^2\boldsymbol{V}^\top - \boldsymbol{V}\exp(\boldsymbol{\Lambda})\boldsymbol{V}^\top \\ &= \boldsymbol{V}\left(\boldsymbol{I} + \boldsymbol{\Lambda} + \boldsymbol{\Lambda}^2 - \exp(\boldsymbol{\Lambda})\right)\boldsymbol{V}^\top \\ &= \boldsymbol{V}\operatorname*{diag}_i\{1 + \lambda_i + \lambda_i^2 - \exp(\lambda_i)\}\boldsymbol{V}^\top\end{aligned}$$

Recall $\exp(x) \leq 1 + x + x^2$ for all $|x| \leq 1$. Since $\|A\| \leq 1$ i.e. $|\lambda_i| \leq 1$ for all i, thus $1 + \lambda_i + \lambda_i^2 - \exp(\lambda_i) \geq 0$ for all i, meaning $\boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 - \exp(\boldsymbol{A}) \succeq 0$. □

Lemma 8.2.14. $\log(\boldsymbol{I} + \boldsymbol{A}) \preceq \boldsymbol{A}$ *for* $A \succ -\boldsymbol{I}$.

Proof.

$$\begin{aligned}\boldsymbol{A} - \log(\boldsymbol{I} + \boldsymbol{A}) &= \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top - \boldsymbol{V}\log(\boldsymbol{\Lambda} + \boldsymbol{I})\boldsymbol{V}^\top \\ &= \boldsymbol{V}\left(\boldsymbol{\Lambda} - \log(\boldsymbol{\Lambda} + \boldsymbol{I})\right)\boldsymbol{V}^\top \\ &= \boldsymbol{V}\operatorname*{diag}_i\{\lambda_i - \log(1 + \lambda_i)\}\boldsymbol{V}^\top\end{aligned}$$

Recall $x \geq \log(1 + x)$ for all $x > -1$. Since $\|A\| \succ -\boldsymbol{I}$ i.e. $\lambda_i > -1$ for all i, thus $\lambda_i - \log(1 + \lambda_i) \geq 0$ for all i, meaning $\boldsymbol{A} - \log(\boldsymbol{I} + \boldsymbol{A}) \succeq 0$. □

Theorem 8.2.15 (Lieb)**.** *Let $f : S^n_{++} \to \mathbb{R}$ be a matrix function given by*

$$f(\boldsymbol{A}) = \operatorname{Tr}\left(\exp\left(\boldsymbol{H} + \log(\boldsymbol{A})\right)\right)$$

for some $\boldsymbol{H} \in S^n$. Then $-f$ is convex (i.e. f is concave).

The Lieb's theorem will be crucial in our proof of Theorem 8.1.4, but it is also highly non-trivial and we will omit its proof here. The interested reader can find a proof in Chapter 8 of [T+15].

Lemma 8.2.16 (Jensen's inequality)**.** $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$ *when f is convex;* $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$ *when f is concave.*

8.2.4 Proof of Matrix Bernstein Concentration Bound

Now, we are ready to prove the Bernstein matrix concentration bound.

Proof of Theorem 8.1.4. For any $\boldsymbol{A} \in S^n$, its spectral norm $\|\boldsymbol{A}\| = \max\{|\lambda_n(\boldsymbol{A})|, |\lambda_1(\boldsymbol{A})|\} = \max\{\lambda_n(\boldsymbol{A}), -\lambda_1(\boldsymbol{A})\}$. Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of $\boldsymbol{X}$. Then,

$$\Pr[\|\boldsymbol{X}\| \geq t] = \Pr\left[(\lambda_n \geq t) \bigvee (-\lambda_1 \geq t)\right] \leq \Pr[\lambda_n \geq t] + \Pr[-\lambda_1 \geq t].$$

Let $\boldsymbol{Y} := \sum_i -\boldsymbol{X}_i$. It's easy to see that $-\lambda_n \leq \cdots \leq -\lambda_1$ are eigenvalues of $\boldsymbol{Y}$, implying $\lambda_n(\boldsymbol{Y}) = -\lambda_1(\boldsymbol{X})$. Since $\mathbb{E}[-\boldsymbol{X}_i] = \mathbb{E}[\boldsymbol{X}_i] = 0$ and $\|-\boldsymbol{X}_i\| = \|\boldsymbol{X}_i\| \leq R$ for all i, if we can bound $\Pr[\lambda_n(\boldsymbol{X}) \geq t]$, then applying to $\boldsymbol{Y}$, we can bound $\Pr[\lambda_n(\boldsymbol{Y}) \geq t]$. As

$$\Pr[-\lambda_1(\boldsymbol{X}) \geq t] = \Pr[\lambda_n(\boldsymbol{Y}) \geq t],$$

it suffices to bound $\Pr[\lambda_n \geq t]$.

For any $\theta > 0$, $\lambda_n \geq t \iff \exp(\theta\lambda_n) \geq \exp(\theta t)$ and $\operatorname{Tr}(\exp(\theta\boldsymbol{X})) = \sum_i \exp(\theta\lambda_i)$ by Claim 8.2.4, thus $\lambda_n \geq t \Rightarrow \operatorname{Tr}(\exp(\theta\boldsymbol{X})) \geq \exp(\theta t)$. Then, using Markov's inequality,

$$\begin{aligned}\Pr[\lambda_n \geq t] &\leq \Pr[\operatorname{Tr}(\exp(\theta\boldsymbol{X})) \geq \exp(\theta t)] \\ &\leq \exp(-\theta t)\,\mathbb{E}[\operatorname{Tr}(\exp(\theta\boldsymbol{X}))]\end{aligned}$$

For two independent random variables $\boldsymbol{U}$ and $\boldsymbol{V}$, we have

$$\mathop{\mathbb{E}}_{\boldsymbol{U},\boldsymbol{V}} f(\boldsymbol{U}, \boldsymbol{V}) = \mathop{\mathbb{E}}_{\boldsymbol{U}} \mathop{\mathbb{E}}_{\boldsymbol{V}} [f(\boldsymbol{U}, \boldsymbol{V}) | \boldsymbol{U}] = \mathop{\mathbb{E}}_{\boldsymbol{U}} \mathop{\mathbb{E}}_{\boldsymbol{V}} [f(\boldsymbol{U}, \boldsymbol{V})].$$

Define $\boldsymbol{X}_{<i} = \sum_{j<i} \boldsymbol{X}_j$. Let $0 < \theta \le 1/R$,

$$
\begin{aligned}
\mathbb{E}\operatorname{Tr}\left(\exp(\theta \boldsymbol{X})\right) &= \mathop{\mathbb{E}}_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_{k-1}} \mathop{\mathbb{E}}_{\boldsymbol{X}_k} \operatorname{Tr}\exp\left(\underbrace{\theta \boldsymbol{X}_{<k}}_{\boldsymbol{H}} + \underbrace{\theta \boldsymbol{X}_k}_{=\log\exp(\theta \boldsymbol{X}_k)}\right), \quad \{\boldsymbol{X}_i\} \text{ are independent} \\
&\le \mathop{\mathbb{E}}_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_{k-1}} \operatorname{Tr}\exp\left(\theta \boldsymbol{X}_{<k} + \log \mathbb{E}\exp(\theta \boldsymbol{X}_k)\right), \quad \text{by 8.2.15 and 8.2.16} \\
&\le \mathop{\mathbb{E}}_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_{k-1}} \operatorname{Tr}\exp\left(\theta \boldsymbol{X}_{<k} + \log \mathbb{E}\left[\boldsymbol{I} + \theta \boldsymbol{X}_k + \theta^2 \boldsymbol{X}_k^2\right]\right), \quad \text{by 8.2.13, 8.2.7, and 8.2.9} \\
&\le \mathop{\mathbb{E}}_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_{k-1}} \operatorname{Tr}\exp\left(\theta \boldsymbol{X}_{<k} + \theta^2 \mathbb{E}\,\boldsymbol{X}_k^2\right), \quad \text{by 8.2.14 and 8.2.7} \\
&= \mathop{\mathbb{E}}_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_{k-2}} \mathop{\mathbb{E}}_{\boldsymbol{X}_{k-1}} \operatorname{Tr}\exp\left(\underbrace{\theta^2 \mathbb{E}\,\boldsymbol{X}_k^2 + \theta \boldsymbol{X}_{<k-1}}_{\boldsymbol{H}} + \theta \boldsymbol{X}_{k-1}\right), \\
&\;\;\vdots \\
&\le \operatorname{Tr}\exp\left(\theta^2 \sum_i \mathbb{E}\left[\boldsymbol{X}_i^2\right]\right), \\
&\le \operatorname{Tr}\exp\left(\theta^2\sigma^2 \boldsymbol{I}\right), \quad \text{by 8.2.7 and } \sum_i \mathbb{E}\left[\boldsymbol{X}_i^2\right] \preceq \sigma^2 \boldsymbol{I} \\
&= n \cdot \exp(\theta^2\sigma^2).
\end{aligned}
$$

Then,

$$\Pr[\lambda_n \ge t] \le n \cdot \exp(-\theta t + \theta^2\sigma^2),$$

and

$$\Pr[\|\boldsymbol{X}\| \ge t] \le 2n \cdot \exp(-\theta t + \theta^2\sigma^2).$$

Similar to the proof of Bernstein concentration bound for one-dimension random variable, minimize the RHS over $0 < \theta \le 1/R$ yields

$$\Pr[\|\boldsymbol{X}\| \ge t] \le 2n \cdot \exp\left(\frac{-t^2}{2Rt + 4\sigma^2}\right).$$

□

8.3 Spectral Graph Sparsification

In this section, we will see that for any dense graph, we can find another sparser graph whose graph Laplacian is approximately the same as measured by their quadratic forms. This turns out to be a very useful tool for designing algorithms.

Definition 8.3.1. Given $\boldsymbol{A}, \boldsymbol{B} \in S_+^n$ and $\epsilon > 0$, we say

$$\boldsymbol{A} \approx_\epsilon \boldsymbol{B} \text{ if and only if } \frac{1}{1+\epsilon}\boldsymbol{A} \preceq \boldsymbol{B} \preceq (1+\epsilon)\boldsymbol{A}.$$

Suppose we start with a connected graph $G = (V, E, \boldsymbol{w})$, where as usual we say that $|V| = n$ and $|E| = m$. We want to produce another graph $\tilde{G} = (V, \tilde{E}, \tilde{\boldsymbol{w}})$ s.t $\left|\tilde{E}\right| \ll |E|$ and at the same time $\boldsymbol{L}_G \approx_\epsilon \boldsymbol{L}_{\tilde{G}}$. We call $\tilde{G}$ a *spectral sparsifier* of G. Our construction will also ensure that $\tilde{E} \subseteq E$, although this is not important in most applications. Figure 8.2 shows an example of a graph G and spectral sparsifier $\tilde{G}$.

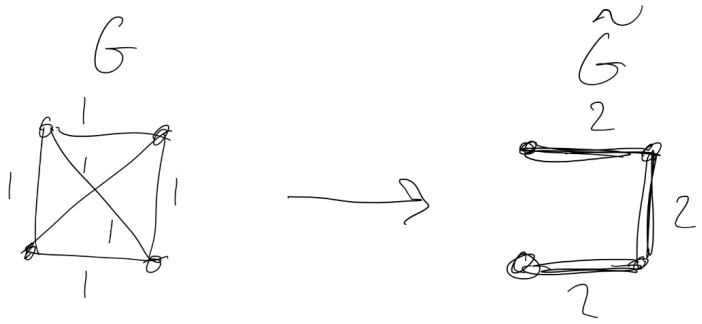


Figure 8.2: A graph G and a spectral sparsifier $\tilde{G}$, satisfisying $\boldsymbol{L}_G \approx_\epsilon \boldsymbol{L}_{\tilde{G}}$ for $\epsilon = 2.42$.

We are going to construct $\tilde{G}$ by sampling some of the edges of G according to a suitable probability distribution and scaling up their weight to make up for the fact that we pick fewer of them.

To get a better understanding for the notion of approximation given in 8.3.1 means, let's observe a simple consequence of it.

Given a vertex subset $T \subseteq V$, we say that $(T, V \setminus T)$ is a *cut* in G and that the value of the cut is

$$c_G(T) = \sum_{e \in E \cap (T \times V \setminus T)} \boldsymbol{w}(e).$$

Figure 8.3 shows the $c_G(T)$ in a graph G.

Theorem 8.3.2. *If $\boldsymbol{L}_G \approx_\epsilon \boldsymbol{L}_{\tilde{G}}$, then for all $T \subseteq V$,*

$$\frac{1}{1+\epsilon} c_G(T) \leq c_{\tilde{G}}(T) \leq (1+\epsilon) c_G(T).$$

Proof. Let $\mathbf{1}_T \in \mathbb{R}^V$ be the indicator of the set T, i.e. $\mathbf{1}_T(u) = 1$ for $u \in V$ and $\mathbf{1}_T(u) = 0$ otherwise. We can see that $\mathbf{1}_T^\top \boldsymbol{L}_G \mathbf{1}_T = c_G(T)$, and hence the theorem follows by comparing the quadratic forms. □

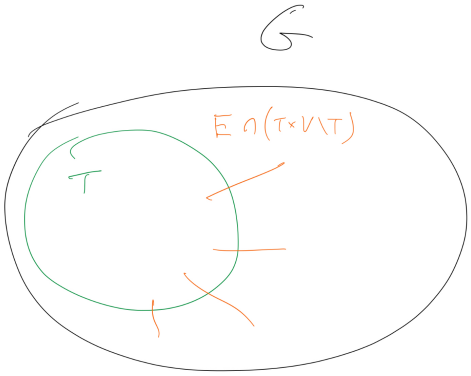


Figure 8.3: The cut $c_G(T)$ in G.

But how well can we spectrally approximate a graph with a sparse graph? The next theorem gives us a nearly optimal answer to this question.

Theorem 8.3.3 (Spectral Graph Approximation by Sampling, (Spielman-Srivastava 2008))**.** *Consider a connected graph $G = (V, E, \boldsymbol{w})$, 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 $\tilde{\boldsymbol{w}}(e) = \frac{1}{p_e}\boldsymbol{w}(e)$, then with probability at least $1 - \delta$ the graph $\tilde{G} = (V, \tilde{E}, \tilde{\boldsymbol{w}})$ satisfies*

$$\boldsymbol{L}_G \approx_\epsilon \boldsymbol{L}_{\tilde{G}} \text{ and } \left|\tilde{E}\right| \leq O(n\epsilon^{-2}\log(n/\delta)).$$

The original proof can be found in [SS11].

Remark 8.3.4. For convenience, we will abbreviate $\boldsymbol{L}_G$ as $\boldsymbol{L}$ and $\boldsymbol{L}_{\tilde{G}}$ as $\tilde{\boldsymbol{L}}$ in the rest of this section.

We are going to analyze a sampling procedure by turning our goal into a problem of matrix concentration. Recall that

Fact 8.3.5. *$\boldsymbol{A} \preceq \boldsymbol{B}$ implies $\boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^\top \preceq \boldsymbol{C}\boldsymbol{B}\boldsymbol{C}^\top$ for any $\boldsymbol{C} \in \mathbb{R}^{n\times n}$.*

By letting $\boldsymbol{C} = \boldsymbol{L}^{+/2}$, we can see that

$$\boldsymbol{L} \approx_\epsilon \tilde{\boldsymbol{L}} \text{ implies } \boldsymbol{\Pi}_{\boldsymbol{L}} \approx_\epsilon \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}, \tag{8.2}$$

where $\boldsymbol{\Pi}_{\boldsymbol{L}} = \boldsymbol{L}^{+/2}\boldsymbol{L}\boldsymbol{L}^{+/2}$ is the orthogonal projection to the complement of the kernel of $\boldsymbol{L}$.

Definition 8.3.6. Given a matrix $\boldsymbol{A}$, we define $\boldsymbol{\Pi}_{\boldsymbol{A}}$ to be the orthogonal projection to the complement of the kernel of $\boldsymbol{A}$, i.e. $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{v} = \boldsymbol{0}$ for $\boldsymbol{v} \in \ker(\boldsymbol{A})$ and $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{v} = \boldsymbol{v}$ for $\boldsymbol{v} \in \ker(\boldsymbol{A})^\perp$. Recall that $\ker(\boldsymbol{A})^\perp = \mathrm{im}(\boldsymbol{A}^\top)$.

Claim 8.3.7. *For a matrix $\boldsymbol{A} \in S^n$ with spectral decomposition $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top = \sum_i \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^\top$ s.t. $\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}$, we have $\boldsymbol{\Pi}_{\boldsymbol{A}} = \sum_{i:\lambda_i\neq 0}\boldsymbol{v}_i\boldsymbol{v}_i^\top$, and $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{A}^{+/2}\boldsymbol{A}\boldsymbol{A}^{+/2} = \boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+\boldsymbol{A}$.*

From the definition, we can see that $\boldsymbol{\Pi_L} = \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$.

Now that we understand the projection $\boldsymbol{\Pi_L}$, it is not hard to show the following claim.

Claim 8.3.8.

1. $\boldsymbol{\Pi_L} \approx_\epsilon \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}$ *implies* $\boldsymbol{L} \approx_\epsilon \tilde{\boldsymbol{L}}$.
2. *For* $\epsilon \leq 1$, *we have that* $\left\|\boldsymbol{\Pi_L} - \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\right\| \leq \epsilon/2$ *implies* $\boldsymbol{\Pi_L} \approx_\epsilon \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}$.

Really, the only idea needed here is that when comparing quadratic forms in matrices with the same kernel, we necessarily can't have the quadratic forms disagree on vectors in the kernel. Simple! But we are going to write it out carefully, since we're still getting used to these types of calculations.

Proof of Claim 8.3.8. To prove Part 1, we assume $\boldsymbol{\Pi_L} \approx_\epsilon \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}$. Recall that G is a connected graph, so $\ker(\boldsymbol{L}) = \operatorname{span}\{\mathbf{1}\}$, while $\tilde{\boldsymbol{L}}$ is the Laplacian of a graph which may or may not be connected, so $\ker(\tilde{\boldsymbol{L}}) \supseteq \ker(\boldsymbol{L})$, and equivalently $\operatorname{im}(\tilde{\boldsymbol{L}}) \subseteq \operatorname{im}(\boldsymbol{L})$. Now, for any $\boldsymbol{v} \in \ker(\boldsymbol{L})$ we have $\boldsymbol{v}^\top\tilde{\boldsymbol{L}}\boldsymbol{v} = 0 = \boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v}$. For any $\boldsymbol{v} \in \ker(\boldsymbol{L})^\perp$ we have $\boldsymbol{v} = \boldsymbol{L}^{+/2}\boldsymbol{z}$ for some $\boldsymbol{z}$, as $\ker(\boldsymbol{L})^\perp = \operatorname{im}(\boldsymbol{L}) = \operatorname{im}(\boldsymbol{L}^{+/2})$. Hence

$$\boldsymbol{v}^\top\tilde{\boldsymbol{L}}\boldsymbol{v} = \boldsymbol{z}^\top\boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\boldsymbol{z} \geq \frac{1}{1+\epsilon}\boldsymbol{z}^\top\boldsymbol{L}^{+/2}\boldsymbol{L}\boldsymbol{L}^{+/2}\boldsymbol{z} = \frac{1}{1+\epsilon}\boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v}$$

and similarly

$$\boldsymbol{v}^\top\tilde{\boldsymbol{L}}\boldsymbol{v} = \boldsymbol{z}^\top\boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\boldsymbol{z} \leq (1+\epsilon)\boldsymbol{z}^\top\boldsymbol{L}^{+/2}\boldsymbol{L}\boldsymbol{L}^{+/2}\boldsymbol{z} = (1+\epsilon)\boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v}.$$

Thus we have established $\boldsymbol{L} \approx_\epsilon \tilde{\boldsymbol{L}}$.

To prove Part 2, we assume $\left\|\boldsymbol{\Pi_L} - \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\right\| \leq \epsilon/2$. This is equivalent to

$$-\frac{\epsilon}{2}\boldsymbol{I} \preceq \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2} - \boldsymbol{\Pi_L} \preceq \frac{\epsilon}{2}\boldsymbol{I}$$

But since

$$\mathbf{1}^\top(\boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2} - \boldsymbol{\Pi_L})\mathbf{1} = 0,$$

we can in fact sharpen this to

$$-\frac{\epsilon}{2}\boldsymbol{\Pi_L} \preceq \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2} - \boldsymbol{\Pi_L} \preceq \frac{\epsilon}{2}\boldsymbol{\Pi_L}.$$

Rearranging, we then conclude

$$(1-\frac{\epsilon}{2})\boldsymbol{\Pi_L} \preceq \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2} \preceq (1+\frac{\epsilon}{2})\boldsymbol{\Pi_L}.$$

Finally, we note that $1/(1+\epsilon) \leq (1-\frac{\epsilon}{2})$ to reach our conclusion, $\boldsymbol{\Pi_L} \approx_\epsilon \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}$. □

We now have most of the tools to prove Theorem 8.3.3, but to help us, we are going to establish one small piece of helpful notation: We define a matrix function $\Phi : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ by

$$\Phi(\boldsymbol{A}) = \boldsymbol{L}^{+/2}\boldsymbol{A}\boldsymbol{L}^{+/2}.$$

We sometimes call this a "normalizing map", because it transforms a matrix to the space where spectral norm bounds can be translated into relative error guarantees compare to the $\boldsymbol{L}$ quadratic form.

Proof of Theorem 8.3.3. By Claim 8.3.8, it suffices to show

$$\left\|\boldsymbol{\Pi}_{\boldsymbol{L}} - \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\right\| \leq \epsilon/2. \tag{8.3}$$

We introduce a set of independent random variables, one for each edge e, with a probability p_e associated with the edge which we will fix later. We then let

$$\boldsymbol{Y}_e = \begin{cases} \frac{\boldsymbol{w}(e)}{p_e}\boldsymbol{b}_e\boldsymbol{b}_e^\top & \text{with probability } p_e \\ \boldsymbol{0} & \text{otherwise.} \end{cases}$$

This way, $\tilde{\boldsymbol{L}} = \sum_e \boldsymbol{Y}_e$. Note that $\mathbb{E}[\boldsymbol{Y}_e] = p_e\frac{\boldsymbol{w}(e)}{p_e}\boldsymbol{b}_e\boldsymbol{b}_e^\top = \boldsymbol{w}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$, and so

$$\mathbb{E}\left[\tilde{\boldsymbol{L}}\right] = \sum_e \mathbb{E}[\boldsymbol{Y}_e] = \boldsymbol{L}.$$

By linearity of Φ,

$$\mathbb{E}\left[\Phi(\tilde{\boldsymbol{L}})\right] = \Phi(\mathbb{E}\left[\tilde{\boldsymbol{L}}\right]) = \boldsymbol{\Pi}_{\boldsymbol{L}}.$$

Let us also define

$$\boldsymbol{X}_e = \Phi(\boldsymbol{Y}_e) - \mathbb{E}[\Phi(\boldsymbol{Y}_e)] \text{ and } \boldsymbol{X} = \sum_e \boldsymbol{X}_e$$

Note that this ensures $\mathbb{E}[\boldsymbol{X}_e] = \boldsymbol{0}$. We are now going to fix the edge sampling probabilities, in a way that depends on some overall scaling parameter $\alpha > 0$. We let

$$p_e = \min\left(\alpha\left\|\Phi\left(\boldsymbol{w}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top\right)\right\|, 1\right)$$

then we see from the definition of $\boldsymbol{Y}_e$ that whenever $p_e < 1$

$$\|\Phi(\boldsymbol{Y}_e)\| \leq \frac{1}{\alpha}$$

from this, we can conclude, with a bit of work, that for all e

$$\|\boldsymbol{X}_e\| \leq \frac{1}{\alpha}. \tag{8.4}$$

We can also show that

$$\left\|\sum_e \mathbb{E}\left[\boldsymbol{X}_e^2\right]\right\| \leq \frac{1}{\alpha}. \tag{8.5}$$

In the exercises for this chapter, we will ask you to show that Equations (8.4) and (8.5) hold. This means that we can apply Theorem 8.1.4 to our $\boldsymbol{X} = \sum_e \boldsymbol{X}_e$, with $R = \frac{1}{\alpha}$ and $\sigma^2 = \frac{1}{\alpha}$, to get

$$\Pr\left[\left\|\boldsymbol{\Pi}_{\boldsymbol{L}} - \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\right\| \geq \epsilon/2\right] \leq 2n \exp\left(\frac{-0.25\epsilon^2}{(\epsilon+4)/\alpha}\right)$$

Since $0 < \epsilon < 1$, this means that if $\alpha = 40\epsilon^{-2}\log(n/\delta)$, then

$$Pr\left[\left\|\boldsymbol{\Pi}_{\boldsymbol{L}} - \boldsymbol{L}^{+/2}\tilde{\boldsymbol{L}}\boldsymbol{L}^{+/2}\right\| \geq \epsilon/2\right] \leq \frac{2n\delta^2}{n^2} \leq \delta/2.$$

In the last step, we assumed $n \geq 4$.

Lastly, we'd like to know that the graph $\tilde{G}$ is sparse. The number of edges in $\tilde{G}$ is equal to the number of $\boldsymbol{Y}_e$ that come out nonzero. Thus, the expected value of $\left|\tilde{E}\right|$ is

$$\mathbb{E}\left[\left|\tilde{E}\right|\right] = \sum_e p_e \leq \alpha \sum_e \boldsymbol{w}(e)\left\|\boldsymbol{L}^{+/2}\boldsymbol{b}_e\boldsymbol{b}_e^\top\boldsymbol{L}^{+/2}\right\|$$

We can bound the sum of the norms with a neat trick relating it to the trace of $\boldsymbol{\Pi}_{\boldsymbol{L}}$. Note that in general for a vector $\boldsymbol{a} \in \mathbb{R}^n$, we have $\left\|\boldsymbol{a}\boldsymbol{a}^\top\right\| = \boldsymbol{a}^\top\boldsymbol{a} = \mathrm{Tr}\left(\boldsymbol{a}\boldsymbol{a}^\top\right)$. And hence

$$\begin{aligned}
\sum_e \boldsymbol{w}(e)\left\|\boldsymbol{L}^{+/2}\boldsymbol{b}_e\boldsymbol{b}_e^\top\boldsymbol{L}^{+/2}\right\| &= \sum_e \boldsymbol{w}(e)\mathrm{Tr}\left(\boldsymbol{L}^{+/2}\boldsymbol{b}_e\boldsymbol{b}_e^\top\boldsymbol{L}^{+/2}\right) \\
&= \mathrm{Tr}\left(\boldsymbol{L}^{+/2}\left(\sum_e \boldsymbol{w}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top\right)\boldsymbol{L}^{+/2}\right) \\
&= \mathrm{Tr}\left(\boldsymbol{\Pi}_{\boldsymbol{L}}\right) = n-1.
\end{aligned}$$

Thus with our choice of α,

$$\mathbb{E}\left[\left|\tilde{E}\right|\right] \leq 40\epsilon^{-2}\log(n/\delta)n.$$

With a scalar Chernoff bound, can show that $\left|\tilde{E}\right| \leq O(\epsilon^{-2}\log(n/\delta)n)$ with probability at least $1-\delta/2$. Thus by a union bound, the this condition and Equation (8.3) are both satisfied with probability at least $1-\delta$. □

Remark 8.3.9. Note that

$$\left\|\Phi\left(\boldsymbol{w}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top\right)\right\| = \boldsymbol{w}(e)\left\|\boldsymbol{L}^{+/2}\boldsymbol{b}_e\boldsymbol{b}_e^\top\boldsymbol{L}^{+/2}\right\| \leq \boldsymbol{w}(e)\left\|\boldsymbol{L}^{+/2}\boldsymbol{b}_e\right\|_2^2.$$

Recall that in Chapter 6, we saw that the effective resistance between vertex v and vertex u is given by $\left\|\boldsymbol{L}^{+/2}(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|_2^2$, and for an edge e connecting vertex u and v, we have $\boldsymbol{b}_e = \boldsymbol{e}_u - \boldsymbol{e}_v$. That means the norm of the "baby Laplacian" $\boldsymbol{w}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$ of a single edge with weight $\boldsymbol{w}(e)$ is exactly $\boldsymbol{w}(e)$ times the effective resistance between the two endpoints of the edge.

We haven't shown how to compute the sampling probabilities efficiently, so right now, it isn't clear whether we can efficiently find $\tilde{G}$. It turns out that if we have access to a fast algorithm for solving Laplacian linear equations, then we can find sufficiently good approximations to the effective resistances quickly, and use these to compute $\tilde{G}$. An algorithm for this is described in [SS11].

Chapter 9

Solving Laplacian Linear Equations

9.1 Solving Linear Equations Approximately

Given a Laplacian $\boldsymbol{L}$ of a connected graph and a demand vector $\boldsymbol{d} \perp \mathbf{1}$, we want to find $\boldsymbol{x}^*$ solving the linear equation $\boldsymbol{L}\boldsymbol{x}^* = \boldsymbol{d}$. We are going to focus on fast algorithms for finding approximate (but highly accurate) solutions.

This means we need a notion of an approximate solution. Since our definition is not special to Laplacians, we state it more generally for positive semi-definite matrices.

Definition 9.1.1. Given PSD matrix $\boldsymbol{M}$ and $\boldsymbol{d} \in \ker(\boldsymbol{M})^\perp$, let $\boldsymbol{M}\boldsymbol{x}^* = \boldsymbol{d}$. We say that $\tilde{\boldsymbol{x}}$ is an ϵ-approximate solution to the linear equation $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{d}$ if

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_{\boldsymbol{M}}^2 \leq \epsilon \|\boldsymbol{x}^*\|_{\boldsymbol{M}}^2 .$$

Remark 9.1.2. The requirement $\boldsymbol{d} \in \ker(\boldsymbol{M})^\perp$ can be removed, but this is not important for us.

Theorem 9.1.3 (Spielman and Teng (2004) [ST04])**.** *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 $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$, using an algorithm that takes time $O(m \log^c n \log(1/\epsilon))$ for some fixed constant c and succeeds with probability $1 - 1/n^{10}$.*

In the original algorithm of Spielman and Teng, the exponent on the log in the running time was $c \approx 70$.

Today, we are going to see a simpler algorithm. But first, we'll look at one of the key tools behind all algorithms for solving Laplacian linear equations quickly.

9.2 Preconditioning and Approximate Gaussian Elimination

Recall our definition of two positive semi-definite matrices being approximately equal.

Definition 9.2.1 (Spectral approximation)**.** Given $\boldsymbol{A}, \boldsymbol{B} \in S^n_+$, we say that

$$\boldsymbol{A} \approx_K \boldsymbol{B} \text{ if and only if } \frac{1}{1+K}\boldsymbol{A} \preceq \boldsymbol{B} \preceq (1+K)\boldsymbol{A}.$$

Suppose we have a positive definite matrix $\boldsymbol{M} \in S^n_{++}$ and want to solve a linear equation $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{d}$. We can do this using gradient descent or accelerated gradient descent, as we covered in Graded Homework 1. But if we have access to an easy-to-invert matrix that happens to also be a good spectral approximation of $\boldsymbol{M}$, then we can use this to speed up the (accelerated) gradient descent algorithm. An example of this would be that we have a factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_K \boldsymbol{M}$, where $\boldsymbol{\mathcal{L}}$ is lower triangular and sparse, which means we can invert it quickly.

The following lemma, which you will prove in Problem Set 6, makes this preconditioning precise.

Lemma 9.2.2. *Given a matrix $\boldsymbol{M} \in S^n_{++}$, a vector $\boldsymbol{d}$ and a decomposition $\boldsymbol{M} \approx_K \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$, we can find $\tilde{\boldsymbol{x}}$ that ϵ-approximately solves $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{d}$, using $O((1+K)\log(K/\epsilon)(T_{matvec}+T_{sol}+n))$ time.*

- *T_{matvec} denotes the time required to compute $\boldsymbol{M}\boldsymbol{z}$ given a vector $\boldsymbol{z}$, i.e. a "matrix-vector multiplication".*
- *T_{sol} denotes the time required to compute $\boldsymbol{\mathcal{L}}^{-1}\boldsymbol{z}$ or $(\boldsymbol{\mathcal{L}}^\top)^{-1}\boldsymbol{z}$ given a vector $\boldsymbol{z}$.*

Dealing with pseudo-inverses. When our matrices have a null space, preconditioning becomes slightly more complicated, but as long as it is easy to project to the complement of the null space, there's no real issue. The following describes precisely what we need (but you can ignore the null-space issue when first reading these notes without losing anything significant).

Lemma 9.2.3. *Given a matrix $\boldsymbol{M} \in S^n_+$, a vector $\boldsymbol{d} \in \ker(\boldsymbol{M})^\perp$ and a decomposition $\boldsymbol{M} \approx_K \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{L}}^\top$, where $\boldsymbol{\mathcal{L}}$ is invertible, we can find $\tilde{\boldsymbol{x}}$ that ϵ-approximately solves $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{d}$, using $O((1+K)\log(K/\epsilon)(T_{matvec}+T_{sol}+T_{proj}+n))$ time.*

- *T_{matvec} denotes the time required to compute $\boldsymbol{M}\boldsymbol{z}$ given a vector $\boldsymbol{z}$, i.e. a "matrix-vector multiplication".*
- *T_{sol} denotes the time required to compute $\boldsymbol{\mathcal{L}}^{-1}\boldsymbol{z}$ and $(\boldsymbol{\mathcal{L}}^\top)^{-1}\boldsymbol{z}$ and $\boldsymbol{\mathcal{D}}^+\boldsymbol{z}$ given a vector $\boldsymbol{z}$.*

- T_{proj} *denotes the time required to compute* $\boldsymbol{\Pi}_{\boldsymbol{M}}\boldsymbol{z}$ *given a vector* $\boldsymbol{z}$.

Theorem 9.2.4 (Kyng and Sachdeva (2015) [KS16])**.** *Given a Laplacian* $\boldsymbol{L}$ *of a weighted undirected graph* $G = (V, E, \boldsymbol{w})$ *with* $|E| = M$ *and* $|V| = n$*, we can find a decomposition* $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_{0.5} \boldsymbol{L}$*, such that* $\boldsymbol{\mathcal{L}}$ *has number of non-zeroes* $\mathrm{nnz}(\boldsymbol{\mathcal{L}}) = O(m \log^3 n)$*, with probability at least* $1 - 3/n^5$*. in time* $O(m \log^3 n)$*.*

We can combine Theorem 9.2.4 with Lemma 9.2.3 to get a fast algorithm for solving Laplacian linear equations.

Corollary 9.2.5. *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* $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{d}$*, using an algorithm that takes time* $O(m \log^3 n \log(1/\epsilon))$ *and succeeds with probability* $1 - 1/n^{10}$*.*

Proof sketch. First we need to get a factorization that confirms to Lemma 9.2.3. The decomposition $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$ provided by Theorem 9.2.4 can be rewritten as $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top = \widetilde{\boldsymbol{\mathcal{L}}}\boldsymbol{\mathcal{D}}(\widetilde{\boldsymbol{\mathcal{L}}})^\top$ where $\widetilde{\boldsymbol{\mathcal{L}}}$ is equal to $\boldsymbol{\mathcal{L}}$ except $\boldsymbol{\mathcal{L}}(n, n) = 1$ and we let $\boldsymbol{\mathcal{D}}$ be the identity matrix, except $\boldsymbol{\mathcal{D}}(n, n) = 0$. This ensures $\boldsymbol{\mathcal{D}}^+ = \boldsymbol{\mathcal{D}}$ and that $\widetilde{\boldsymbol{\mathcal{L}}}$ is invertible and lower triagular with $O(m \log^3 n)$ non-zeros. We note that the inverse of an invertible lower or upper triangular matrix with N non-zeros can be applied in time $O(N)$ given an adjacency list representation of the matrix. Finally, as $\ker(\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top) = \mathrm{span}\{\mathbf{1}\}$, we have $\boldsymbol{\Pi}_{\widetilde{\boldsymbol{\mathcal{L}}}\boldsymbol{\mathcal{D}}(\widetilde{\boldsymbol{\mathcal{L}}})^\top} = \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, and this projection matrix can be applied in $O(n)$ time. Altogether, this means that $T_{\text{matvec}} + T_{\text{sol}} + T_{\text{proj}} = O(n)$, which suffices to complete the proof. □

9.3 Approximate Gaussian Elimination Algorithm

Recall *Gaussian Elimination / Cholesky decomposition* of a graph Laplacian $\boldsymbol{L}$. We will use $\boldsymbol{A}(:, i)$ to denote the the ith column of a matrix $\boldsymbol{A}$. We can write the algorithm as

Algorithm 1: Gaussian Elimination / Cholesky Decomposition

Input: Graph Laplacian $\boldsymbol{L}$
Output: Lower triangular $\boldsymbol{\mathcal{L}}$ s.t. $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top = \boldsymbol{L}$

Let $\boldsymbol{S}_0 = \boldsymbol{L}$;
for $i = 1$ *to* $i = n - 1$ **do**
  $\boldsymbol{l}_i = \frac{1}{\sqrt{\boldsymbol{S}_{i-1}(i,i)}}\boldsymbol{S}_{i-1}(:, i)$;
  $\boldsymbol{S}_i = \boldsymbol{S}_{i-1} - \boldsymbol{l}_i\boldsymbol{l}_i^\top$.
$\boldsymbol{l}_n = \mathbf{0}_{n\times 1}$;
return $\boldsymbol{\mathcal{L}} = \begin{bmatrix}\boldsymbol{l}_1 \cdots \boldsymbol{l}_n\end{bmatrix}$;

We want to introduce some notation that will help us describe and analyze a faster version of Gaussian elimination – one that uses sampling to create a sparse approximation of the decomposition.

Consider a Laplacian $\boldsymbol{S}$ of a graph H and a vertex v of H. We define $\textsc{Star}(v, \boldsymbol{S})$ to be the Laplacian of the subgraph of H consisting of edges incident on v. We define

$$\textsc{Clique}(v, \boldsymbol{S}) = \textsc{Star}(v, \boldsymbol{S}) - \frac{1}{\boldsymbol{S}(v,v)} \boldsymbol{S}(:, v)\boldsymbol{S}(:, v)^\top$$

For example, suppose

$$\boldsymbol{L} = \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \mathrm{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-1} \end{pmatrix}$$

Then

$$\textsc{Star}(1, \boldsymbol{L}) = \begin{pmatrix} W & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \mathrm{diag}(\boldsymbol{a}) \end{pmatrix} \text{ and } \textsc{Clique}(1, \boldsymbol{L}) = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathrm{diag}(\boldsymbol{a}) - \frac{1}{W}\boldsymbol{a}\boldsymbol{a}^\top \end{pmatrix}$$

which is illustrated in Figure 9.1.

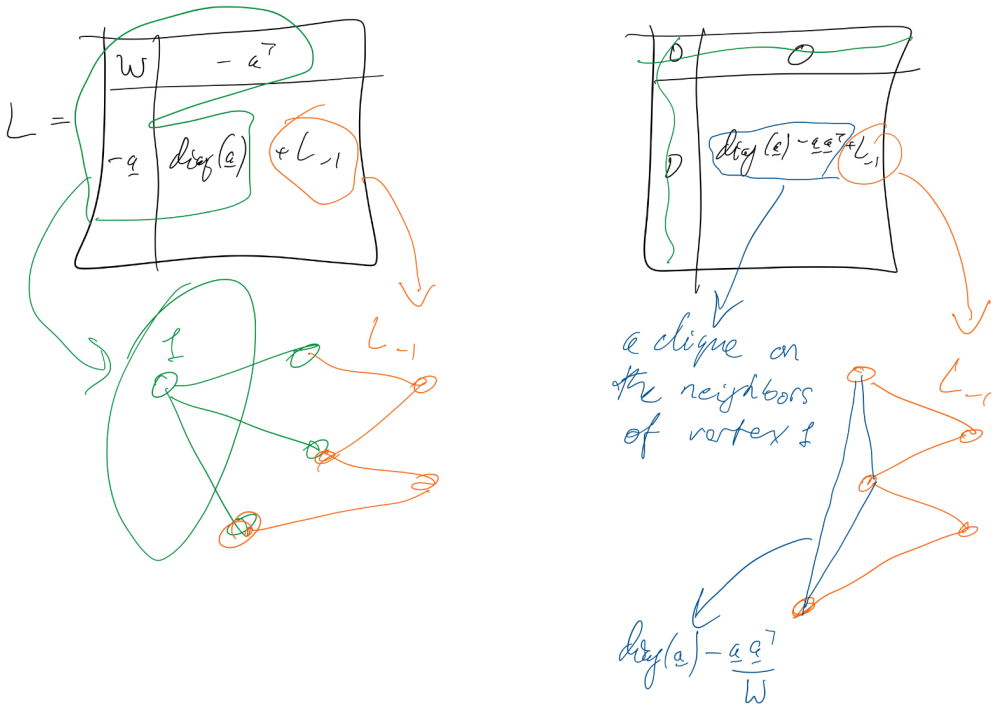


Figure 9.1: Gaussian Elimination: $\textsc{Clique}(1, \boldsymbol{L}) = \textsc{Star}(1, \boldsymbol{L}) - \frac{1}{\boldsymbol{L}(1,1)} \boldsymbol{L}(:, 1)\boldsymbol{L}(:, 1)^\top$.

In Chapter 7, we proved that $\textsc{Clique}(v, \boldsymbol{S})$ is a graph Laplacian – it follows from the proof of Claim 7.1.1 in that chapter. Thus we have that following.

Claim 9.3.1. *If $\boldsymbol{S}$ is the Laplacian of a connected graph, then $\textsc{Clique}(v, \boldsymbol{S})$ is a graph Laplacian.*

Note that in Algorithm 1, we have $\boldsymbol{l}_i \boldsymbol{l}_i^\top = \textsc{Star}(v_i, \boldsymbol{S}_{i-1}) - \textsc{Clique}(v_i, \boldsymbol{S}_{i-1})$. The update rule can be rewritten as

$$\boldsymbol{S}_i = \boldsymbol{S}_{i-1} - \textsc{Star}(v_i, \boldsymbol{S}_{i-1}) + \textsc{Clique}(v_i, \boldsymbol{S}_{i-1}),$$

This also provides way to understand why Gaussian Elimination is slow in some cases. At each step, one vertex is eliminated, but a clique is added to the subgraph on the remaining vertices, making the graph denser. And at the ith step, computing $\textsc{Star}(v_i, \boldsymbol{S}_{i-1})$ takes around $\deg(v_i)$ time, but computing $\textsc{Clique}(v_i, \boldsymbol{S}_{i-1})$ requires around $\deg(v_i)^2$ time. In order to speed up Gaussian Elimination, the algorithmic idea of [KS16] is to plug in a sparser appproximate of the intended clique instead of the entire one.

The following procedure $\textsc{CliqueSample}(v, \boldsymbol{S})$ produces a sparse approximation of $\textsc{clique}(v, \boldsymbol{S})$. Let V be the vertex set of the graph associated with $\boldsymbol{S}$ and E the edge set. We define $\boldsymbol{b}_{i,j} \in \mathbb{R}^V$ to be the vector with

$$\boldsymbol{b}_{i,j}(i) = 1 \text{ and } \boldsymbol{b}_{i,j}(j) = -1 \text{ and } \boldsymbol{b}_{i,j}(k) = 0 \text{ for } k \neq i, j.$$

Given weights $\boldsymbol{w} \in \mathbb{R}^E$ and a vertex $v \in V$, we let

$$\boldsymbol{w}_v = \sum_{(u,v)\in E} \boldsymbol{w}(u, v).$$

Algorithm 2: $\textsc{CliqueSample}(v, \boldsymbol{S})$

Input: Graph Laplacian $\boldsymbol{S} \in \mathbb{R}^{V\times V}$, of a graph with edge weights $\boldsymbol{w}$, and vertex $v \in V$
Output: $\boldsymbol{Y}_v \in \mathbb{R}^{V\times V}$ sparse approximation of $\textsc{clique}(v, \boldsymbol{S})$

$\boldsymbol{Y}_v \leftarrow \boldsymbol{0}_{n\times n}$;
foreach *Multiedge* $e = (v, i)$ *from* v *to a neighbor* i **do**
  Randomly pick a neighbor j of v with probability $\frac{\boldsymbol{w}(j,v)}{\boldsymbol{w}_v}$;
  If $i \neq j$, let $\boldsymbol{Y}_v \leftarrow \boldsymbol{Y}_v + \frac{\boldsymbol{w}(i,v)\boldsymbol{w}(j,v)}{\boldsymbol{w}(i,v)+\boldsymbol{w}(j,v)} \boldsymbol{b}_{i,j}\boldsymbol{b}_{i,j}^\top$;
return $\boldsymbol{Y}_v$;

Remark 9.3.2. We can implement each sampling of a neighbor j in $O(1)$ time using a classical algorithm known as Walker's method (also known as the Alias method or Vose's method). This algorithm requires an additional $O(\deg_{\boldsymbol{S}}(v))$ time to initialize a data structure used for sampling. Overall, this means the total time for $O(\deg_{\boldsymbol{S}}(v))$ samples is still $O(\deg_{\boldsymbol{S}}(v))$.

Lemma 9.3.3. $\mathbb{E}[\boldsymbol{Y}_v] = \textsc{Clique}(v, \boldsymbol{S})$.

Proof. Let $\boldsymbol{C} = \textsc{Clique}(v, \boldsymbol{S})$. Observe that both $\mathbb{E}[\boldsymbol{Y}_v]$ and $\boldsymbol{C}$ are Laplacians. Thus it suffices to verify $\mathbb{E}_{\boldsymbol{Y}_v(i,j)} = \boldsymbol{C}(i, j)$ for $i \neq j$.

$$\boldsymbol{C}(i, j) = -\frac{\boldsymbol{w}(i, v)\boldsymbol{w}(j, v)}{\boldsymbol{w}_v},$$

$$\mathop{\mathbb{E}}_{\boldsymbol{Y}_v(i,j)} = -\frac{\boldsymbol{w}(i, v)\boldsymbol{w}(j, v)}{\boldsymbol{w}(i, v) + \boldsymbol{w}(j, v)}\left(\frac{\boldsymbol{w}(j, v)}{\boldsymbol{w}_v} + \frac{\boldsymbol{w}(i, v)}{\boldsymbol{w}_v}\right) = -\frac{\boldsymbol{w}(i, v)\boldsymbol{w}(j, v)}{\boldsymbol{w}_v} = \boldsymbol{C}(i, j).$$

□

Remark 9.3.4. Lemma 9.3.3 shows that CliqueSample$(v, \boldsymbol{L})$ produces the original Clique$(v, \boldsymbol{L})$ in expectation.

Now, we define *Approximate Gaussian Elimination*.

Algorithm 3: Approximate Gaussian Elimination / Cholesky Decomposition

Input: Graph Laplacian $\boldsymbol{L}$

Output: Lower triangular[a] $\boldsymbol{\mathcal{L}}$ as given in Theorem 9.2.4

Let $\boldsymbol{S}_0 = \boldsymbol{L}$;

Generate a random permutation π on $[n]$;

for $i = 1$ *to* $i = n - 1$ **do**

  $\boldsymbol{l}_i = \frac{1}{\sqrt{\boldsymbol{S}_{i-1}(\pi(i),\pi(i))}} \boldsymbol{S}_{i-1}(:, \pi(i))$;

  $\boldsymbol{S}_i = \boldsymbol{S}_{i-1} - \text{Star}(\pi(i), \boldsymbol{S}_{i-1}) + \text{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1})$

$\boldsymbol{l}_n = \boldsymbol{0}_{n\times 1}$;

return $\boldsymbol{\mathcal{L}} = \begin{bmatrix} \boldsymbol{l}_1 \cdots \boldsymbol{l}_n \end{bmatrix}$ *and* π;

[a] $\boldsymbol{\mathcal{L}}$ is not actually lower triangular. However, if we let $\boldsymbol{P}_\pi$ be the permutation matrix corresponding to π, then $\boldsymbol{P}_\pi \boldsymbol{\mathcal{L}}$ is lower triangular. Knowing the ordering that achieves this is enough to let us implement forward and backward substitution for solving linear equations in $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{L}}^\top$.

Note that if we replace CliqueSample$(\pi(i), \boldsymbol{S}_{i-1})$ by Clique$(\pi(i), \boldsymbol{S}_{i-1})$ at each step, then we can recover Gaussian Elimination, but with a random elimination order.

9.4 Analyzing Approximate Gaussian Elimination

In this Section, we're going to analyze Approximate Gaussian Elimination, and see why it works.

Ultimately, the main challenge in proving Theorem 9.2.4 will be to prove for the output $\boldsymbol{\mathcal{L}}$ of Algorithm 3 that with high probability

$$0.5\boldsymbol{L} \preceq \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \preceq 1.5\boldsymbol{L}. \tag{9.1}$$

We can reduce this to proving that with high probability

$$\left\| \boldsymbol{L}^{+/2}(\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top - \boldsymbol{L})\boldsymbol{L}^{+/2} \right\| \leq 0.5 \tag{9.2}$$

Ultimately, the proof is going to have a lot in common with our proof of Matrix Bernstein in Chapter 8. Overall, the lesson there was that when we have a sum of independent, zero-mean random matrices, we can show that the sum is likely to have small spectral norm if the spectral norm of each random matrix is small, and the matrix-valued variance is also small.

Thus, to replicate the proof, we need control over

1. The *sample norms*.

2. The *sample variance*.

But, there is seemlingly another major obstacle: We are trying to analyze a process where the samples are far from independent. Each time we sample edges, we add new edges to the remaining graph, which we will the later sample again. This creates a lot of dependencies between the samples, which we have to handle.

However, it turns out that independence is more than what is needed to prove concentration. Instead, it suffices to have a sequence of random variables such that each is mean-zero in expectation, conditional on the previous ones. This is called a martingale difference sequence. We'll now learn about those.

9.4.1 Normalization, a.k.a. Isotropic Position

Since our analysis requires frequently measuring matrices after right and left-multiplication by $\boldsymbol{L}^{+/2}$, we reintroduce the "normalizing map" $\Phi : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ defined by

$$\Phi(\boldsymbol{A}) = \boldsymbol{L}^{+/2}\boldsymbol{A}\boldsymbol{L}^{+/2}.$$

We previously saw this in Chapter 8.

9.4.2 Martingales

A scalar martingale is a sequence of random variables $Z_0, \ldots, Z_k$, such that

$$\mathbb{E}\left[Z_i \mid Z_0, \ldots, Z_{i-1}\right] = Z_{i-1}. \tag{9.3}$$

That is, conditional on the outcome of all the previous random variables, the expectation of Z_i equals Z_{i-1}. If we unravel the sequence of conditional expectations, we get that *without conditioning* $\mathbb{E}\left[Z_k\right] = \mathbb{E}\left[Z_0\right]$.

Typically, we use martingales to show a statement along like "Z_k is concentrated around $\mathbb{E}\left[Z_k\right]$".

We can also think of a martingale in terms of the sequence of changes in the Z_i variables. Let $X_i = Z_i - Z_{i-1}$. The sequence of X_is is called a martingale difference sequence. We can now state the martingale condition as

$$\mathbb{E}\left[X_i \mid Z_0, \ldots, Z_{i-1}\right] = 0.$$

And because Z_0 and $X_1, \ldots, X_{i-1}$ completely determine $Z_1, \ldots, Z_{i-1}$, we could also write the martingale condition equivalently as

$$\mathbb{E}\left[X_i \mid Z_0, X_1, \ldots, X_{i-1}\right] = 0.$$

Crucially, we can write

$$Z_k = Z_0 + \sum_{i=1}^{k} Z_i - Z_{i-1} = Z_0 + \sum_{i=1}^{k} X_i$$

and when we are trying to prove concentration, the martingale difference property of the X_i's is often "as good as" independence, meaning that $\sum_{i=1}^{k} X_i$ concentrates similarly to a sum of independent random variables.

Matrix-valued martingales. We can also define matrix-valued martingales. In this case, we replace the martingalue condition of Equation (9.3), with the condition that the whole matrix stays the same in expectation. For example, we could have a sequence of random matrices $\boldsymbol{Z}_0, \ldots, \boldsymbol{Z}_k \in \mathbb{R}^{n \times n}$, such that

$$\mathbb{E}\left[\boldsymbol{Z}_i \mid \boldsymbol{Z}_0, \ldots, \boldsymbol{Z}_{i-1}\right] = \boldsymbol{Z}_{i-1}. \tag{9.4}$$

Lemma 9.4.1. *Let* $\boldsymbol{L}_i = \boldsymbol{S}_i + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top$ *for* $i = 1, ..., n$ *and* $\boldsymbol{L}_0 = \boldsymbol{S}_0 = \boldsymbol{L}$. *Then*

$$\mathbb{E}\left[\boldsymbol{L}_i | \textit{all random variables before } \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1})\right] = \boldsymbol{L}_{i-1}.$$

Proof. Let's only consider $i = 1$ here as other cases are similar.

$$\boldsymbol{L}_0 = \boldsymbol{L} = \boldsymbol{l}_1 \boldsymbol{l}_1^\top + \textsc{Clique}(v, \boldsymbol{L}) + \boldsymbol{L}_{-1}$$

$$\begin{aligned}
\boldsymbol{L}_1 &= \boldsymbol{l}_1 \boldsymbol{l}_1^\top + \textsc{CliqueSample}(v, \boldsymbol{L}) + \boldsymbol{L}_{-1} \\
\mathbb{E}\left[\boldsymbol{L}_1 | \pi(1)\right] &= \boldsymbol{l}_1 \boldsymbol{l}_1^\top + \mathbb{E}\left[\textsc{CliqueSample}(v, \boldsymbol{L}) \,|\pi(1)\right] + \boldsymbol{L}_{-1} \\
&= \boldsymbol{l}_1 \boldsymbol{l}_1^\top + \textsc{Clique}(v, \boldsymbol{L}) + \boldsymbol{L}_{-1} \\
&= \boldsymbol{L}_0
\end{aligned}$$

where we used Lemma 9.3.3 to get $\mathbb{E}\left[\textsc{CliqueSample}(v, \boldsymbol{L}) \,|\pi(1)\right] = \textsc{Clique}(v, \boldsymbol{L})$. □

Remark 9.4.2. $\sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top$ can be treated as what has already been eliminated by (Approximate) Gaussian Elimination, while $\boldsymbol{S}_i$ is what still left or going to be eliminated. In Approximate Gaussian Elimination, $\boldsymbol{L}_n = \sum_{i=1}^{n} \boldsymbol{l}_i \boldsymbol{l}_i^\top$ and our goal is to show that $\boldsymbol{L}_n \approx_K \boldsymbol{L}$. Note that $\boldsymbol{L}_i$ is always equal to the original Laplacian $\boldsymbol{L}$ for all i in Gaussian Elimination. Lemma 9.4.1 demonstrates that $\boldsymbol{L}_0, \boldsymbol{L}_1, ..., \boldsymbol{L}_n$ forms a matrix martingale.

Ultimately, our plan is to use this matrix martingale structure to show that "$\boldsymbol{L}_n$ is concentrated around $\boldsymbol{L}$" in some appropriate sense. More precisely, the spectral approximation we would like to show can be established by showing that "$\Phi(\boldsymbol{L}_n)$ is concentrated around $\Phi(\boldsymbol{L})$"

9.4.3 Martingale Difference Sequence as Edge-Samples

We start by taking a slightly different view of the observations we used to prove Lemma 9.4.1. Recall that $\boldsymbol{L}_i = \boldsymbol{S}_i + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top$, and $\boldsymbol{L}_{i-1} = \boldsymbol{S}_{i-1} + \sum_{j=1}^{i-1} \boldsymbol{l}_j \boldsymbol{l}_j^\top$ and

$$\boldsymbol{S}_i = \boldsymbol{S}_{i-1} - \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1}) + \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1})\,.$$

Putting these together, we get

$$\begin{aligned}
\boldsymbol{L}_i - \boldsymbol{L}_{i-1} &= \boldsymbol{l}_i \boldsymbol{l}_i^\top + \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1}) - \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1}) \\
&= \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1}) - \textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1}) \qquad (9.5) \\
&= \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1}) - \mathbb{E}\left[\textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1}) \mid \text{preceding samples}\right] \\
&\qquad\qquad \text{by Lemma 9.3.3.}
\end{aligned}$$

In particular, recall that by Lemma 9.3.3, conditional on the randomness before the call to $\textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1})$, we have

$$\mathbb{E}\left[\textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1}) \mid \text{preceding samples}\right] = \textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1})$$

Adopting the notation of Lemma 9.3.3 we write

$$\boldsymbol{Y}_{\pi(i)} = \textsc{CliqueSample}(\pi(i), \boldsymbol{S}_{i-1})$$

and we further introduce notation each multi-edge sample for $e \in \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})$, as $\boldsymbol{Y}_{\pi(i),e}$, denoting the random edge Laplacian sampled when the algorithm is processing multi-edge e. Thus, conditional on preceding samples, we have

$$\boldsymbol{Y}_{\pi(i)} = \sum_{e \in \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})} \boldsymbol{Y}_{\pi(i),e} \qquad (9.6)$$

Note that even the number of multi-edges in $\textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})$ depends on the preceding samples. We also want to associate zero-mean variables with each edge. Conditional on preceding samples, we also define

$$\boldsymbol{X}_{i,e} = \Phi\left(\boldsymbol{Y}_{\pi(i),e} - \mathbb{E}\left[\boldsymbol{Y}_{\pi(i),e}\right]\right) \text{ and } \boldsymbol{X}_i = \sum_{e \in \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})} \boldsymbol{X}_{i,e}$$

and combining this with Equations (9.5) and (9.6)

$$\boldsymbol{X}_i = \Phi(\boldsymbol{Y}_{\pi(i)} - \mathbb{E}\left[\boldsymbol{Y}_{\pi(i)}\right]) = \Phi(\boldsymbol{L}_i - \boldsymbol{L}_{i-1})$$

Altogether, we can write

$$\Phi\left(\boldsymbol{L}_n - \boldsymbol{L}\right) = \sum_{i=1}^{n} \Phi(\boldsymbol{L}_i - \boldsymbol{L}_{i-1}) = \sum_{i=1}^{n} \boldsymbol{X}_i = \sum_{i=1}^{n} \sum_{e \in \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})} \boldsymbol{X}_{i,e}$$

Note that the $\boldsymbol{X}_{i,e}$ variables form a martingale difference sequence, because the linearity of Φ ensures they are zero-mean conditional on preceding randomness.

9.4.4 Stopped Martingales

Unfortunately, directly analyzing the concentration properties of the $\boldsymbol{L}_i$ martingale that we just introduced turns out to be difficult. The reason is that we're trying to prove some very delicate multiplicative error guarantees. And, if we analyze $\boldsymbol{L}_i$, we find that the multiplicative error is not easy to control, *after it's already gotten big*. But that's not really what we care about anyway: We want to say it never gets big in the first place, with high probability. So we need to introduce another martingale, that lets us ignore the bad case when the error has already gotten too big. At the same time, we also need to make sure that statements about our new martingale can help us prove guarantees about $\boldsymbol{L}_i$. Fortunately, we can achieve both at once. The technique we use is related to the much broader topic of martingale *stopping times*, which we only scratch the surface of here. We're also going to be quite informal about it, in the interest of brevity. Lecture notes by Tropp [Tro19] give a more formal introduction for those who are interested.

We define the stopped martingale sequence $\tilde{\boldsymbol{L}}_i$ by

$$\tilde{\boldsymbol{L}}_i = \begin{cases} \boldsymbol{L}_i & \text{if for all } j < i \text{ we have } \boldsymbol{L}_i \preceq 1.5\boldsymbol{L} \\ \boldsymbol{L}_{j^*} & \text{for } j^* \text{ being the least } j \text{ such that } \boldsymbol{L}_j \not\preceq 1.5\boldsymbol{L} \end{cases} \tag{9.7}$$

Figure 9.2 shows the $\tilde{\boldsymbol{L}}_i$ martingale getting stuck at the first time $\boldsymbol{L}_{j^*} \not\preceq 1.5\boldsymbol{L}$.

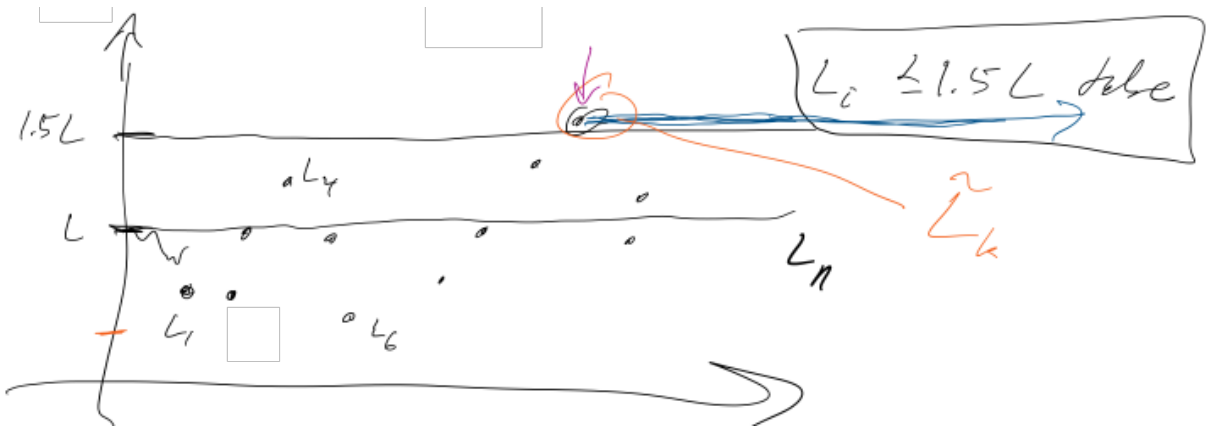


Figure 9.2: Gaussian Elimination : $\textsc{Clique}(1, \boldsymbol{L}) = \textsc{Star}(1, \boldsymbol{L}) - \frac{1}{\boldsymbol{L}(1,1)}\boldsymbol{L}(:,1)\boldsymbol{L}(:,1)^\top$.

We state the following without proof:

Claim 9.4.3.

1. *The sequence* $\left\{\tilde{\boldsymbol{L}}_i\right\}$ *for* $i = 0, \ldots, n$ *is a martingale.*
2. $\left\|\boldsymbol{L}^{+/2}(\tilde{\boldsymbol{L}}_i - \boldsymbol{L})\boldsymbol{L}^{+/2}\right\| \leq 0.5$ *implies* $\left\|\boldsymbol{L}^{+/2}(\boldsymbol{L}_i - \boldsymbol{L})\boldsymbol{L}^{+/2}\right\| \leq 0.5$

The martingale property also implies that the unconditional expectation satisfies $\mathbb{E}\left[\tilde{\boldsymbol{L}}_n\right] = \boldsymbol{L}$. The proof of the claim is easy to sketch: For Part 1, each difference is zero-mean if the condition has not been violated, and is identically zero (and hence zero-mean) if it has been violated. For Part 2, if the martingale $\left\{\tilde{\boldsymbol{L}}_i\right\}$ has stopped, then $\left\|\boldsymbol{L}^{+/2}(\tilde{\boldsymbol{L}}_i - \boldsymbol{L})\boldsymbol{L}^{+/2}\right\| \leq 0.5$ is false, and the implication is vacuosly true. If, on the other hand, the martingale has not stopped, the quantities are equal, because $\tilde{\boldsymbol{L}}_i = \boldsymbol{L}_i$, and again it's easy to see the implication holds.

Thus, ultimately, our strategy is goin to be to show that $\left\|\boldsymbol{L}^{+/2}(\tilde{\boldsymbol{L}}_i - \boldsymbol{L})\boldsymbol{L}^{+/2}\right\| \leq 0.5$ with high probability. Expressed using the normalizing map $\Phi(\cdot)$, our goal is to show that with high probability

$$\left\|\Phi(\tilde{\boldsymbol{L}}_n - \boldsymbol{L})\right\| \leq 0.5.$$

Stopped martingale difference sequence. In order to prove the spectral norm bound, we want to express the $\left\{\tilde{\boldsymbol{L}}_i\right\}$ martingale in terms of a sequence of martingale differences. To this end, we define $\tilde{\boldsymbol{X}}_i = \Phi(\tilde{\boldsymbol{L}}_i - \tilde{\boldsymbol{L}}_{i-1})$. This ensures that

$$\tilde{\boldsymbol{X}}_i = \begin{cases} \boldsymbol{X}_i & \text{if for all } j < i \text{ we have } \boldsymbol{L}_i \preceq 1.5\boldsymbol{L} \\ \boldsymbol{0} & \text{otherwise} \end{cases} \tag{9.8}$$

Whenever the modified martingale $\tilde{\boldsymbol{X}}_i$ has not yet stopped, we also introduce individual modified edge samples $\tilde{\boldsymbol{X}}_{i,e} = \boldsymbol{X}_{i,e}$. If the martingale *has* stopped, i.e. $\tilde{\boldsymbol{X}}_i = \boldsymbol{0}$, then we can take these edge samples $\tilde{\boldsymbol{X}}_{i,e}$ to be zero. We can now write

$$\Phi\left(\tilde{\boldsymbol{L}}_n - \boldsymbol{L}\right) = \sum_{i=1}^{n} \Phi(\tilde{\boldsymbol{L}}_i - \tilde{\boldsymbol{L}}_{i-1}) = \sum_{i=1}^{n} \tilde{\boldsymbol{X}}_i = \sum_{i=1}^{n} \sum_{e \in \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})} \tilde{\boldsymbol{X}}_{i,e}.$$

Thus, we can see that Equation (9.2) is implied by

$$\left\|\sum_{i=1}^{n} \tilde{\boldsymbol{X}}_i\right\| \leq 0.5. \tag{9.9}$$

9.4.5 Sample Norm Control

In this Subsection, we're going to see that the norms of each multi-edge sample is controlled throughout the algorithm.

Lemma 9.4.4. *Given two Laplacians $\boldsymbol{L}$ and $\boldsymbol{S}$ on the same vertex set.*[1] *If each multiedge e of $\textsc{Star}(v, \boldsymbol{S})$ has bounded norm in the following sense,*

$$\left\|\boldsymbol{L}^{+/2}\boldsymbol{w}_{\boldsymbol{S}}(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top\boldsymbol{L}^{+/2}\right\| \leq R,$$

[1] $\boldsymbol{L}$ can be regarded as the original Laplacian we care about, while $\boldsymbol{S}$ can be regarded as some intermediate Laplacian appearing during Approximate Gaussian Elimination.

then each possible sampled multiedge e' of CliqueSample$(v, \boldsymbol{S})$ *also satisfies*

$$\left\| \boldsymbol{L}^{+/2} \boldsymbol{w}_{\text{new}}(e') \boldsymbol{b}_{e'} \boldsymbol{b}_{e'}^\top \boldsymbol{L}^{+/2} \right\| \leq R.$$

Proof. Let $\boldsymbol{w} = \boldsymbol{w}_{\boldsymbol{S}}$ for simplicity. Consider a sampled edge between i and j with weight $\boldsymbol{w}_{\text{new}}(i,j) = \boldsymbol{w}(i,v)\boldsymbol{w}(j,v)/(\boldsymbol{w}(i,v) + \boldsymbol{w}(j,v))$.

$$\begin{aligned}
\left\| \boldsymbol{L}^{+/2} \boldsymbol{w}_{\text{new}}(i,j) \boldsymbol{b}_{ij} \boldsymbol{b}_{ij}^\top \boldsymbol{L}^{+/2} \right\| &= \boldsymbol{w}_{\text{new}}(i,j) \left\| \boldsymbol{L}^{+/2} \boldsymbol{b}_{ij} \boldsymbol{b}_{ij}^\top \boldsymbol{L}^{+/2} \right\| \\
&= \boldsymbol{w}_{\text{new}}(i,j) \left\| \boldsymbol{L}^{+/2} \boldsymbol{b}_{ij} \right\|^2 \\
&\leq \boldsymbol{w}_{\text{new}}(i,j) \left(\left\| \boldsymbol{L}^{+/2} \boldsymbol{b}_{iv} \right\|^2 + \left\| \boldsymbol{L}^{+/2} \boldsymbol{b}_{jv} \right\|^2 \right) \\
&= \frac{\boldsymbol{w}(j,v)}{\boldsymbol{w}(i,v) + \boldsymbol{w}(j,v)} \left\| \boldsymbol{L}^{+/2} \boldsymbol{w}(i,v) \boldsymbol{b}_{iv} \boldsymbol{b}_{iv}^\top \boldsymbol{L}^{+/2} \right\| + \\
&\quad \frac{\boldsymbol{w}(i,v)}{\boldsymbol{w}(i,v) + \boldsymbol{w}(j,v)} \left\| \boldsymbol{L}^{+/2} \boldsymbol{w}(j,v) \boldsymbol{b}_{jv} \boldsymbol{b}_{jv}^\top \boldsymbol{L}^{+/2} \right\| \\
&\leq \frac{\boldsymbol{w}(j,v)}{\boldsymbol{w}(i,v) + \boldsymbol{w}(j,v)} R + \frac{\boldsymbol{w}(i,v)}{\boldsymbol{w}(i,v) + \boldsymbol{w}(j,v)} R \\
&= R
\end{aligned}$$

The first inequality uses the triangle inequality of effective resistance in $\boldsymbol{L}$, in that effective resistance is a distance as we proved in Chapter 6. The second inequality just uses the conditions of this lemma. □

Remark 9.4.5. Lemma 9.4.4 only requires that each single multiedge has small norm instead of that the sum of all edges between a pair of vertices have small norm. And this lemma tells us, after sampling, each multiedge in the new graph still satisfies the bounded norm condition.

From the Lemma, we can conclude that each edge sample $\boldsymbol{Y}_{\pi(i),e}$ satisfies $\left\| \Phi(\boldsymbol{Y}_{\pi(i),e}) \right\| \leq R$ provided the assumptions of the Lemma hold. Let's record this observation as a Lemma.

Lemma 9.4.6. *If for all $e \in$ Star$(v, \boldsymbol{S}_i)$,*

$$\left\| \Phi(\boldsymbol{w}_{\boldsymbol{S}_i}(e) \boldsymbol{b}_e \boldsymbol{b}_e^\top) \right\| \leq R.$$

then all $e \in$ Star$(\pi(i), \boldsymbol{S}_i)$,

$$\left\| \Phi(\boldsymbol{Y}_{\pi(i),e}) \right\| \leq R.$$

Preprocessing by multi-edge splitting. In the original graph of Laplacian $\boldsymbol{L}$ of graph $G = (V, E, \boldsymbol{w})$, we have for each edge $\hat{e}$ that

$$\boldsymbol{w}(\hat{e}) \boldsymbol{b}_{\hat{e}} \boldsymbol{b}_{\hat{e}}^\top \preceq \sum_e \boldsymbol{w}(e) \boldsymbol{b}_e \boldsymbol{b}_e^\top = \boldsymbol{L}$$

This also implies that

$$\left\| \boldsymbol{L}^{+/2} \boldsymbol{w}(\hat{e}) \boldsymbol{b}_{\hat{e}} \boldsymbol{b}_{\hat{e}}^{\top} \boldsymbol{L}^{+/2} \right\| \leq 1.$$

Now, that means that if we split every original edge e of the graph into K multi-edges $e_1, \ldots e_K$, with a fraction $1/K$ of the weight, we get a new graph $G' = (V, E', \boldsymbol{w}')$ such that

Claim 9.4.7.

1. *G' and G have the same graph Laplacian.*
2. *$|E'| = K\,|E|$*
3. *For every multi-edge in G'*

$$\left\| \boldsymbol{L}^{+/2} \boldsymbol{w}'(e) \boldsymbol{b}_e \boldsymbol{b}_e^{\top} \boldsymbol{L}^{+/2} \right\| \leq 1/K.$$

Before we run Approximate Gaussian Elimination, we are going to do this multi-edge splitting to ensure we have control over multi-edge sample norms. Combined with Lemma 9.4.4 immediately establishes the next lemma, because we start off with all multi-edges having bounded norm and only produce multi-edges with bounded norm.

Lemma 9.4.8. *When Algorithm 3 is run on the (multi-edge) Laplacian of G', arising from splitting edges of G into K multi-edges, the every edge sample $\boldsymbol{Y}_{\pi(i),e}$ satisfies*

$$\left\| \Phi(\boldsymbol{Y}_{\pi(i),e}) \right\| \leq 1/K.$$

As we will see later $K = 200 \log^2 n$ suffices.

9.4.6 Random Matrix Concentration from Trace Exponentials

Let us recall how matrix-valued variances come into the picture when proving concentration following the strategy from Matrix Bernstein in Chapter 8.

For some matrix-valued random variable $\boldsymbol{X} \in S^n$, we'd like to show $\Pr[\|\boldsymbol{X}\| \leq 0.5]$. Using Markov's inequality, and some observations about matrix exponentials and traces, we saw that for all $\theta > 0$,

$$\Pr[\|\boldsymbol{X}\| \geq 0.5] \leq \exp(-0.5\theta) \left(\mathbb{E}\left[\mathrm{Tr}\left(\exp\left(\theta \boldsymbol{X} \right) \right) \right] + \mathbb{E}\left[\mathrm{Tr}\left(\exp\left(-\theta \boldsymbol{X} \right) \right) \right] \right). \tag{9.10}$$

We then want to bound $\mathbb{E}\left[\mathrm{Tr}\left(\exp\left(\theta \boldsymbol{X} \right) \right) \right]$ using Lieb's theorem. We can handle $\mathbb{E}\left[\mathrm{Tr}\left(\exp\left(-\theta \boldsymbol{X} \right) \right) \right]$ similarly.

Theorem 9.4.9 (Lieb)**.** *Let $f : S^n_{++} \to \mathbb{R}$ be a matrix function given by*

$$f(\boldsymbol{A}) = \mathrm{Tr}\left(\exp\left(\boldsymbol{H} + \log(\boldsymbol{A}) \right) \right)$$

for some $\boldsymbol{H} \in S^n$. Then $-f$ is convex (i.e. f is concave).

As observed by Tropp, this is useful for proving matrix concentration statements. Combined with Jensen's inequality, it gives that for a random matrix $\boldsymbol{X} \in S^n$ and a fixed $\boldsymbol{H} \in S^n$

$$\mathbb{E}\left[\operatorname{Tr}\left(\exp\left(\boldsymbol{H}+\boldsymbol{X}\right)\right)\right] \leq \operatorname{Tr}\left(\exp\left(\boldsymbol{H}+\log(\mathbb{E}\left[\exp(\boldsymbol{X})\right])\right)\right).$$

The next crucial step was to show that it suffices to obtain an upper bound on the matrix $\mathbb{E}\left[\exp(\boldsymbol{X})\right]$ w.r.t the Loewner order. Using the following three lemmas, this conclusion is an immediate corollary.

Lemma 9.4.10. *If $\boldsymbol{A} \preceq \boldsymbol{B}$, then $\operatorname{Tr}\left(\exp(\boldsymbol{A})\right) \leq \operatorname{Tr}\left(\exp(\boldsymbol{B})\right)$.*

Lemma 9.4.11. *If $0 \prec \boldsymbol{A} \preceq \boldsymbol{B}$, then $\log(\boldsymbol{A}) \preceq \log(\boldsymbol{B})$.*

Lemma 9.4.12. *$\log(\boldsymbol{I}+\boldsymbol{A}) \preceq \boldsymbol{A}$ for $\boldsymbol{A} \succ -\boldsymbol{I}$.*

Corollary 9.4.13. *For a random matrix $\boldsymbol{X} \in S^n$ and a fixed $\boldsymbol{H} \in S^n$, if $\mathbb{E}\left[\exp(\boldsymbol{X})\right] \preceq \boldsymbol{I}+\boldsymbol{U}$ where $\boldsymbol{U} \succ -\boldsymbol{I}$, then*

$$\mathbb{E}\left[\operatorname{Tr}\left(\exp\left(\boldsymbol{H}+\boldsymbol{X}\right)\right)\right] \leq \operatorname{Tr}\left(\exp\left(\boldsymbol{H}+\boldsymbol{U}\right)\right).$$

9.4.7 Mean-Exponential Bounds from Variance Bounds

To use Corollary 9.4.13, we need to construct useful upper bounds on $\mathbb{E}\left[\exp(\boldsymbol{X})\right]$. This can be done, starting from the following lemma.

Lemma 9.4.14. *$\exp(\boldsymbol{A}) \preceq \boldsymbol{I}+\boldsymbol{A}+\boldsymbol{A}^2$ for $\|\boldsymbol{A}\| \leq 1$.*

If $\boldsymbol{X}$ is zero-mean and $\|\boldsymbol{X}\| \leq 1$, this means that $\mathbb{E}\left[\exp(\boldsymbol{X})\right] \preceq \boldsymbol{I}+\mathbb{E}\left[\boldsymbol{X}^2\right]$, which is how we end up wanting to bound the matrix-valued variance $\mathbb{E}\left[\boldsymbol{X}^2\right]$. In the rest of this Subsection, we're going to see the matrix-valued variance of the stopped martingale is bounded throughout the algorithm.

Firstly, we note that for a single edge sample $\tilde{\boldsymbol{X}}_{i,e}$, by Lemma 9.4.8, we have that

$$\left\|\tilde{\boldsymbol{X}}_{i,e}\right\| \leq \left\|\Phi\left(\boldsymbol{Y}_{\pi(i),e}-\mathbb{E}\left[\boldsymbol{Y}_{\pi(i),e}\right]\right)\right\| \leq 1/K,$$

using that $\|\boldsymbol{A}-\boldsymbol{B}\| \leq \max(\|\boldsymbol{A}\|,\|\boldsymbol{B}\|)$, for $\boldsymbol{A},\boldsymbol{B} \succeq \boldsymbol{0}$, and $\|\mathbb{E}\left[\boldsymbol{A}\right]\| \leq \mathbb{E}\left[\|\boldsymbol{A}\|\right]$ by Jensen's inequality.

Thus, if $0 < \theta \leq K$, we have that

$$\begin{aligned}\mathbb{E}\left[\exp(\theta\tilde{\boldsymbol{X}}_{i,e}) \mid \text{preceding samples}\right] &\preceq \boldsymbol{I}+\mathbb{E}\left[(\theta\tilde{\boldsymbol{X}}_{i,e})^2 \mid \text{preceding samples}\right] \\ &\preceq \boldsymbol{I}+\frac{1}{K}\theta^2\cdot\mathbb{E}\left[\Phi(\boldsymbol{Y}_{\pi(i),e}) \mid \text{preceding samples}\right]\end{aligned} \tag{9.11}$$

9.4.8 The Overall Mean-Trace-Exponential Bound

We will use $\mathbb{E}_{(<i)}$ to denote expectation over variables preceding the ith elimination step. We are going to refrain from explicitly writing out conditioning in our expectations, but any *inner* expectation that appears inside another *outer* expectation should be taken as conditional on the outer expectation. We are going to use d_i to denote the multi-edge degree of vertex $\pi(i)$ in $\boldsymbol{S}_{i-1}$. This is exactly the number of edge samples in the ith elimination. Note that there is no elimination at step n (the algorithm is already finished). As a notational convenience, let's write $\hat{n} = n - 1$. With all that in mind, we bound the mean-trace-exponential for some parameter $0 < \theta \leq 0.5/\sqrt{K}$

$$\mathbb{E}\operatorname{Tr}\left(\exp(\theta\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i)\right) \tag{9.12}$$

$$= \mathop{\mathbb{E}}_{(<\hat{n})}\mathop{\mathbb{E}}_{\pi(\hat{n})}\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},1}}\cdots\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}-1}}\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}}}\operatorname{Tr}\exp\left(\underbrace{\sum_{i=1}^{\hat{n}-1}\theta\tilde{\boldsymbol{X}}_i + \sum_{e=1}^{d_{\hat{n}}-1}\theta\tilde{\boldsymbol{X}}_{\hat{n},e}}_{\boldsymbol{H}} + \theta\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}}\right)$$

$\tilde{\boldsymbol{X}}_{\hat{n},1}\ldots,\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}}$ are independent conditional on $(<\hat{n}),\pi(\hat{n})$

$$\leq \mathop{\mathbb{E}}_{(<\hat{n})}\mathop{\mathbb{E}}_{\pi(\hat{n})}\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},1}}\cdots\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}-1}}\operatorname{Tr}\exp\left(\sum_{i=1}^{\hat{n}-1}\theta\tilde{\boldsymbol{X}}_i + \sum_{e=1}^{d_{\hat{n}}-1}\theta\tilde{\boldsymbol{X}}_{\hat{n},e} + \frac{1}{K}\theta^2\cdot\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}}}\Phi(\boldsymbol{Y}_{\pi(\hat{n}),d_{\hat{n}}})\right)$$

By Equation (9.11) and Corollary 9.4.13 .

$\vdots$ Repeat for each multi-edge sample $\tilde{\boldsymbol{X}}_{\hat{n},1}\ldots,\tilde{\boldsymbol{X}}_{\hat{n},d_{\hat{n}}-1}$

$$\leq \mathop{\mathbb{E}}_{(<\hat{n})}\mathop{\mathbb{E}}_{\pi(\hat{n})}\operatorname{Tr}\exp\left(\sum_{i=1}^{\hat{n}-1}\theta\tilde{\boldsymbol{X}}_i + \sum_{e=1}^{d_{\hat{n}}}\frac{1}{K}\theta^2\cdot\mathop{\mathbb{E}}_{\tilde{\boldsymbol{X}}_{\hat{n},e}}\Phi(\boldsymbol{Y}_{\pi(\hat{n}),e})\right)$$

$$= \mathop{\mathbb{E}}_{(<\hat{n})}\mathop{\mathbb{E}}_{\pi(\hat{n})}\operatorname{Tr}\exp\left(\sum_{i=1}^{\hat{n}-1}\theta\tilde{\boldsymbol{X}}_i + \frac{1}{K}\theta^2\Phi(\textsc{Clique}(\pi(\hat{n}),\boldsymbol{S}_{\hat{n}-1}))\right)$$

To further bound the this quantity, we now need to deal with the random choice of $\pi(\hat{n})$. We'll be able to use this to bound the trace-exponential in a very strong way. From a random matrix perspective, it's the following few steps that give the analysis it's surprising strength.

We can treat $\frac{1}{K}\theta^2\Phi(\textsc{Clique}(\pi(\hat{n}),\boldsymbol{S}_{\hat{n}-1}))$ as a random matrix. It is not zero-mean, but we can still bound the trace-exponential using Corollary 9.4.13.

We can also bound the expected matrix exponential in that case, using a simple corollary of Lemma 9.4.14.

Corollary 9.4.15. $\exp(\boldsymbol{A}) \preceq \boldsymbol{I} + (1+R)\boldsymbol{A}$ *for* $\boldsymbol{0} \preceq \boldsymbol{A}$ *with* $\|\boldsymbol{A}\| \leq R \leq 1$.

Proof. The conclusion follows after observing that for $\boldsymbol{0} \preceq \boldsymbol{A}$ with $\|\boldsymbol{A}\| \leq R$, we have $\boldsymbol{A}^2 \preceq R\boldsymbol{A}$. We can see this by considering the spectral decomposition of $\boldsymbol{A}$ and dealing with each eigenvalue separately. □

Next, we need a simple structural observation about the cliques created by elimination:

Claim 9.4.16.

$$\textsc{Clique}(\pi(i), \boldsymbol{S}_i) \preceq \textsc{Star}(\pi(i), \boldsymbol{S}_i) \preceq \boldsymbol{S}_i$$

Proof. The first inequality is immediate from $\textsc{Clique}(\pi(i), \boldsymbol{S}_i) \preceq \textsc{Clique}(\pi(i), \boldsymbol{S}_i) + \boldsymbol{l}_i \boldsymbol{l}_i^\top = \textsc{Star}(\pi(i), \boldsymbol{S}_i)$. The latter inequality $\textsc{Star}(\pi(i), \boldsymbol{S}_i) \preceq \boldsymbol{S}_i$ follows from the star being a subgraph of the whole Laplacian $\boldsymbol{S}_i$. □

Next we make use of the fact that $\tilde{\boldsymbol{X}}_i$ is from the difference sequence of the *stopped* martingale. This means we can assume

$$\boldsymbol{S}_i \preceq 1.5\boldsymbol{L},$$

since otherwise $\tilde{\boldsymbol{X}}_i = \boldsymbol{0}$ and we get an even better bound on the trace-exponential. To make this formal, in Equation (9.12), we ought to do a case analysis that also includes the case $\tilde{\boldsymbol{X}}_i = \boldsymbol{0}$ when the martingale has stopped, but we omit this.

Thus we can conclude by Claim 9.4.16 that

$$\|\Phi(\textsc{Clique}(\pi(i), \boldsymbol{S}_i))\| \leq 1.5.$$

By our assumption $0 < \theta \leq 0.5/\sqrt{K}$, we have $\left\| \frac{1}{K}\theta^2 \Phi(\textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1})) \right\| \leq 1$, so that by Corollary 9.4.15,

$$\begin{aligned} \mathop{\mathbb{E}}_{\pi(i)} \exp\left(\frac{1}{K}\theta^2 \Phi(\textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1})) \right) &\preceq \boldsymbol{I} + \frac{2}{K}\theta^2 \mathop{\mathbb{E}}_{\pi(i)} \Phi(\textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1})) \qquad (9.13) \\ &\preceq \boldsymbol{I} + \frac{2}{K}\theta^2 \mathop{\mathbb{E}}_{\pi(i)} \Phi(\textsc{Star}(\pi(i), \boldsymbol{S}_{i-1})) \end{aligned}$$

by Claim 9.4.16.

Next we observe that, because every multi-edge appears in exactly two stars, and $\pi(i)$ is chosen uniformly at random among the $n+1-i$ vertices that $\boldsymbol{S}_{i-1}$ is supported on, we have

$$\mathop{\mathbb{E}}_{\pi(i)} \textsc{Star}(\pi(i), \boldsymbol{S}_{i-1}) = 2\frac{1}{n+1-i}\boldsymbol{S}_{i-1}.$$

And, since we assume $\boldsymbol{S}_i \preceq 1.5\boldsymbol{L}$, we further get

$$\mathop{\mathbb{E}}_{\pi(i)} \exp\left(\frac{1}{K}\theta^2 \Phi(\textsc{Clique}(\pi(i), \boldsymbol{S}_{i-1})) \right) \preceq \boldsymbol{I} + \frac{6\theta^2}{K(n+1-i)}\boldsymbol{I}.$$

We can combine this with Equation (9.12) and Corollary 9.4.13 to get

$$\begin{aligned} &\mathbb{E}\,\mathrm{Tr}\left(\exp(\theta \sum_{i=1}^{\hat{n}} \tilde{\boldsymbol{X}}_i) \right) \\ &\leq \mathop{\mathbb{E}}_{(<\hat{n})} \mathop{\mathbb{E}}_{\pi(\hat{n})} \mathrm{Tr}\exp\left(\sum_{i=1}^{\hat{n}-1} \theta\tilde{\boldsymbol{X}}_i + \frac{1}{K}\theta^2 \Phi(\textsc{Clique}(\pi(\hat{n}), \boldsymbol{S}_{\hat{n}-1})) \right) \\ &\leq \mathop{\mathbb{E}}_{(<\hat{n})} \mathrm{Tr}\exp\left(\sum_{i=1}^{\hat{n}-1} \theta\tilde{\boldsymbol{X}}_i + \frac{6\theta^2}{K(n+1-i)}\boldsymbol{I} \right) \end{aligned}$$

And by repeating this analysis for each term $\tilde{\boldsymbol{X}}_i$, we get

$$\begin{aligned}\mathbb{E}\,\mathrm{Tr}\left(\exp(\theta\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i)\right) &\leq \mathrm{Tr}\exp\left(\sum_{i=1}^{\hat{n}}\frac{6\theta^2}{K(n+1-i)}\boldsymbol{I}\right)\\ &\leq \mathrm{Tr}\exp\left(\frac{7\theta^2\log(n)}{K}\boldsymbol{I}\right)\\ &= n\exp\left(\frac{7\theta^2\log(n)}{K}\right)\end{aligned}$$

Then, by choosing $K = 200\log^2 n$ and $\theta = 0.5\sqrt{K}$, we get

$$\exp(-0.5\theta)\,\mathbb{E}\,\mathrm{Tr}\left(\exp(\theta\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i)\right) \leq \exp(-0.5\theta)n\exp\left(\frac{7\theta^2\log(n)}{K}\right) \leq 1/n^5.$$

$\mathbb{E}\,\mathrm{Tr}\left(\exp(-\theta\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i)\right)$ can be bounded by an identical argument, so that Equation (9.10) gives

$$\Pr\left[\left\|\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i\right\| \geq 0.5\right] \leq 2/n^5.$$

Thus we have established $\left\|\sum_{i=1}^{\hat{n}}\tilde{\boldsymbol{X}}_i\right\| \leq 0.5$ with high probability (Equation (9.9)), and this in turn implies Equation (9.2), and finally Equation (9.1):

$$0.5\boldsymbol{L} \preceq \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \preceq 1.5\boldsymbol{L}.$$

Now, all that's left to note is that the running time is linear in the multi-edge degree of the vertex being eliminated in each iteration (and this also bounds the number of non-zero entries being created in $\boldsymbol{\mathcal{L}}$). The total number of multi-edges left in the remaining graph stays constant at $Km = O(m\log^2 n)$. Thus the expected degree in the ith elimination is $Km/(n+i-1)$, because the remaining number of vertices is $n+i-1$. Hence the total running time and total number of non-zero entries created can both be bounded as

$$Km\sum_i 1/(n+i-1) = O(m\log^3 n).$$

We can further prove that the bound $O(m\log^3 n)$ on running time and number of non-zeros in $\boldsymbol{L}$ holds with high probability (e.g. $1-1/n^5$). To show this, we essentially need a scalar Chernoff bound, in except the degrees are in fact not independent, and so we need a scalar martingale concentration result, e.g. Azuma's Inequality. This way, we complete the proof of Theorem 9.2.4.

Part III

Combinatorial Graph Algorithms

Chapter 10

Classical Algorithms for Maximum Flow I

10.1 Maximum Flow

In this chapter, we will study the *Maximum Flow Problem* and discuss some classical algorithms for finding solutions to it.

Setup. Consider a directed graph $G = (V, E, \boldsymbol{c})$, where V denotes vertices, E denotes edges and $\boldsymbol{c} \in \mathbb{R}^E, \boldsymbol{c} \geq 0$ denotes edge capacities. In contrast to earlier chapters, the direction of each edge will be important, and not just as a book keeping tool (which is how we previously used it in undirected graphs). We consider an edge $(u, v) \in E$ to be *from* u and *to* v. Edge capacities are associated with directed edges, and we allow both edge (u, v) and (v, u) to exist, and they may have different capacities.

A *flow* is any vector $\boldsymbol{f} \in \mathbb{R}^E$.

We say that a flow is feasible when $\mathbf{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$. The constraint $\mathbf{0} \leq \boldsymbol{f}$ ensures that the flow respects edge directions, while the constraint $\boldsymbol{f} \leq \boldsymbol{c}$ ensures that the flow does not exceed capacity on any edge.

We still define the edge-vertex incidence matrix $\boldsymbol{B} \in \mathbb{R}^{V \times E}$ of G by

$$\boldsymbol{B}(v, e) = \begin{cases} 1 & \text{if } e = (u, v) \\ -1 & \text{if } e = (v, u) \\ 0 & \text{o.w.} \end{cases}$$

As in the undirected case, a *demand vector* is a vector $\boldsymbol{d} \in \mathbb{R}^V$. And as in the undirected case, we can express the net flow constraint that $\boldsymbol{f}$ routes the demand $\boldsymbol{d}$ by

$$\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}.$$

We will focus on the case:

$$\boldsymbol{B}\boldsymbol{f} = F(-\boldsymbol{e}_s + \boldsymbol{e}_t) = F\boldsymbol{b}_{st}.$$

Flows that satisfy the above equation for some scalar F are called s-t flows where $s, t \in V$. The vertex s is called the source, t is called the sink and $\boldsymbol{e}_s$, $\boldsymbol{e}_t$ are indicator vectors for source and sink nodes respectively. The vector $\boldsymbol{b}_{st}$ has -1 at source and 1 at sink. The maximum flow can be expressed as a linear program as follows

$$\begin{aligned} \max_{\boldsymbol{f} \in \mathbb{R}^E, F} \quad & F \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st} \\ & 0 \leq \boldsymbol{f} \leq \boldsymbol{c} \end{aligned} \tag{10.1}$$

We use $\text{val}(\boldsymbol{f})$ to denote F when $\boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st}$.

10.2 Flow Decomposition

We now look at a way of simplifying flows

Definition 10.2.1. A s-t path flow is a flow $\boldsymbol{f} \geq \boldsymbol{0}$ that can be written as

$$\boldsymbol{f} = \alpha \sum_{e \in p} \boldsymbol{e}_e$$

where p is a simple path from s to t.

Definition 10.2.2. A cycle flow is a flow $\boldsymbol{f} \geq \boldsymbol{0}$ that can be written as a cycle i.e.

$$\boldsymbol{f} = \alpha \sum_{e \in c} \boldsymbol{e}_e$$

where c is a simple cycle.

Lemma 10.2.3 (The path-cycle decomposition lemma)**.** *Any s-t flow $\boldsymbol{f}$ can be decomposed into a sum of s-t path flows and cycle flows such that the sum contains at most* $\text{nnz}(\boldsymbol{f})$ *terms. Note* $\text{nnz}(\boldsymbol{f}) \leq |E|$.

Proof. We perform induction on the number of edges with non-zero flow, which we denote by $\text{nnz}(\boldsymbol{f})$. Note that by "the support of $\boldsymbol{f}$", we mean the set $\{e \in E : \boldsymbol{f}(e) \neq 0\}$.

Base case: $\boldsymbol{f} = 0$: nothing to do.

Inductive step: Try to find a path from s to t OR a cycle in the support of $\boldsymbol{f}$.
"Path" case. If there exists such a s-t path, let α be the minimum flow value along the edges of the path, i.e.

$$\alpha = \min_{(a,b)\in p} \boldsymbol{f}(a,b)$$

$$\boldsymbol{f}' = \alpha \sum_{e\in p} \boldsymbol{e}_e$$

Update the flow $\boldsymbol{f}$ by

$$\boldsymbol{f} \leftarrow \boldsymbol{f} - \boldsymbol{f}'$$

The value of the flow will still be non-negative after this update as we subtracted the minimum entry along any positive edge on the path. The number of non-zeros, $\text{nnz}(\boldsymbol{f})$, went down by at least one. Note that the updated $\boldsymbol{f}$ must again be an s-t flow, as it is the difference of two s-t flows.

"Cycle" case. Suppose we find a cycle c in the support of $\boldsymbol{f}$. Let α be the minimum flow value along the edges of the cycle, i.e.

$$\alpha = \min_{(a,b)\in c} \boldsymbol{f}(a,b)$$

$$\boldsymbol{f}' = \alpha \sum_{e\in c} \boldsymbol{e}_e$$

Update the flow $\boldsymbol{f}$ by

$$\boldsymbol{f} \leftarrow \boldsymbol{f} - \boldsymbol{f}'$$

As in the path case, $\boldsymbol{f}$ stays non-negative, and number of non-zeros, $\text{nnz}(\boldsymbol{f})$, goes down by at least one. Note that the updated $\boldsymbol{f}$ must again be an s-t flow, as it is the difference of two s-t flows.

"No path or cycle" case. Suppose we can find neither a path nor a cycle, and $\boldsymbol{f} \neq \boldsymbol{0}$. Then there must be an edge (u, v) with non-zero flow leading into a vertex $v \neq s, t$ and with no outgoing edge from v in the support of $\boldsymbol{f}$. In that case, we must have $(\boldsymbol{B}\boldsymbol{f})(v) > 0$. But since $v \neq t$, this contradicts $\boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st}$. So this case cannot occur. □

Lemma 10.2.4. *In any s-t max flow problem instance, there is an optimal flow $\boldsymbol{f}^*$ with a path-cycle decomposition that has only paths and no cycles.*

Proof. Let $\tilde{\boldsymbol{f}}$ be an optimal flow. Let $\boldsymbol{f}^*$ be the sum of path flows in the path cycle decomposition of $\tilde{\boldsymbol{f}}$. They route the same flow (as cycles contribute to no net flow from s to t). Thus

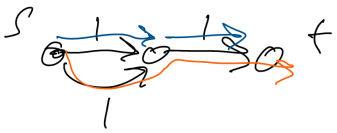
$$\boldsymbol{B}\boldsymbol{f}^* = \boldsymbol{B}\tilde{\boldsymbol{f}}$$

and hence $\text{val}(\boldsymbol{f}^*) = \text{val}(\tilde{\boldsymbol{f}})$. Furthermore

$$\boldsymbol{0} \leq \boldsymbol{f}^* \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c}$$

The first inequality follows from $\boldsymbol{f}^*$ being a sum of positive path flow. The second inequality holds as $\boldsymbol{f}^*$ is upper bounded in every single entry by $\tilde{\boldsymbol{f}}$, because we reduced it by positive entry cycles. The third inequality holds because $\tilde{\boldsymbol{f}}$ is a feasible flow, so it is upper bounded by the capacities. □

An optimal flow solving the maximum flow problem may not be unique. For example, consider the graph below with source s and sink t:



There are two optimal paths in this example. Maximum flow is a convex optimization problem but not a strongly convex problem as the solutions are not unique, and this is part of what makes it hard to solve.

10.3 Cuts and Minimum Cuts

The decomposition shown earlier provides a way to show that the maximum flow in a graph is upper bounded by constructing graph cuts.

Given a vertex subset $S \subseteq V$, we say that $(S, V \setminus S)$ is a *cut* in G and that the value of the cut is

$$c_G(S) = \sum_{e \in E \cap (S \times V \setminus S)} \boldsymbol{c}(e).$$

Note that in a directed graph, only edges crossing the cut going *from* S and *to* $V \setminus S$ count toward the cut.

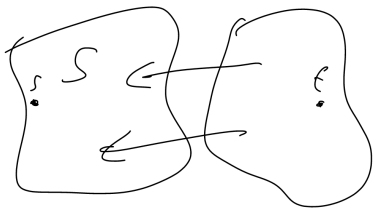


Figure 10.1: Example of a cut: No edges go from S to $(S, V \setminus S)$, and so the value of this cut is zero.

Definition. (s-t cuts). We define an s-t cut to be a subset $S \subset V$, where $s \in S$ and $t \in V \setminus S$.

A decision problem: *"Given an instance of the Maximum Flow problem, is there a flow from s to t such that $\boldsymbol{f} \neq \mathbf{0}$?"*

If YES: We can decompose this flow into s-t path flows and cycle flows.

If NO: There is no flow path from s to t. Let S be the set of vertices reachable from source s. Then $(S, V \setminus S)$ is a cut in the graph, with no edges crossing from S to $V \setminus S$. Figure 10.1 gives an example.

Upper bounding the maximum possible flow value. How can we recognize a maximum flow? Is there a way to confirm that a flow is of maximum value?

We can now introduce the *Minimum Cut* problem.

$$\begin{aligned} &\min_{S \subseteq V} c_G(S) \\ &\text{s.t. } s \in V \text{ and } t \notin V \end{aligned} \tag{10.2}$$

The Minimum Cut problem can also be phrased as a linear program, although we won't see that today.

We'd like to obtain a tight connection between flows and cuts in the graph. As a first step, we won't get that, but we can at least observe that the value of any s-t cut provides an upper bound to the maximum possible s-t flow value.

Theorem 10.3.1 (Max Flow $\leq$ Min Cut)**.** *The maximum s-t flow value in a directed graph G (Program* (10.1)*) is upper bounded by the minimum value of any s-t cut (Program* (10.2)*). I.e. if S is an s-t cut, and $\boldsymbol{f}$ a feasible s-t flow then*

$$\text{val}(\boldsymbol{f}) \leq c_G(S)$$

And in particular this holds for any minimum cut S^* and maximum flow $\boldsymbol{f}^*$, i.e. $\text{val}(\boldsymbol{f}^*) \leq c_G(S^*)$.

Proof. Consider any feasible flow $0 \leq \boldsymbol{f} \leq \boldsymbol{c}$ and a cut $S, T = V \setminus S$. Consider a path-cycle decomposition of $\boldsymbol{f}$, where each s-t path must cross the cut going forward from S to T at least once. We pick a cut S, T with source on one side and sink on the other side as shown in the Figure (10.2). Every time the path flow passes through the cut, it has to use one of the edges that connect S and T. Total amount of flow crossing the cut is bounded above by total amount of capacity of the cut, otherwise the capacities would be violated, thus $\text{val}(\boldsymbol{f}) \leq c_G(S, T) = \sum_{e \in E \cap S \times T} \boldsymbol{c}(e)$.

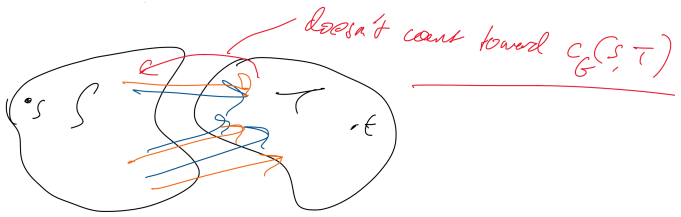


Figure 10.2: s-t Cut

Note that the edges from T to S do not count towards the cut. The above equation holds for all flows with all s-t cuts. This implies that **max flow** $\leq$ **min cut**. □

This theorem is an instance of a general pattern, known as *weak duality*. Weak duality is a relationship between two optimization programs, a maximization problem and a minimization problem, where any solution to the former has its value upper bounded by the value of any solution to the latter.

10.4 Algorithms for Max flow

How can we find a good flow?

Algorithm 4: A first attempt – bad idea?

1 $\boldsymbol{f} \leftarrow \boldsymbol{0}$;
2 **repeat**
3 Find an s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible with respect to $\boldsymbol{c} - \boldsymbol{f}$.
4 $\boldsymbol{f} \leftarrow \boldsymbol{f} + \tilde{\boldsymbol{f}}$

Does Algorithm 1 work? Consider the graph below with directed edges with capacities 1 at every edge. If we make a single path update as shown by the orange lines in Figure 10.3, then afterwards, using the remaining capacity, there's no path flow we can route, as shown in Figure 10.3. But the max flow is 2, as shown by the flow on orange edges in Figure 10.5. So, the above algorithm does not always find a maximum flow: It can get stuck at a point where it cannot make progress despite not having reached the maximum value.

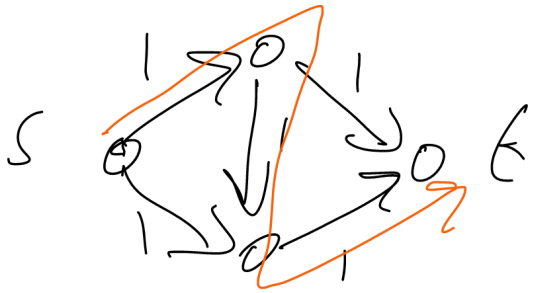


Figure 10.3: Sending a unit s-t path flow through the graph.

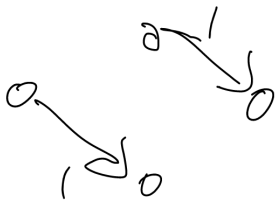


Figure 10.4: Remaining edge capacities after sending a path flow through the graph as depicted in Figure 10.3.

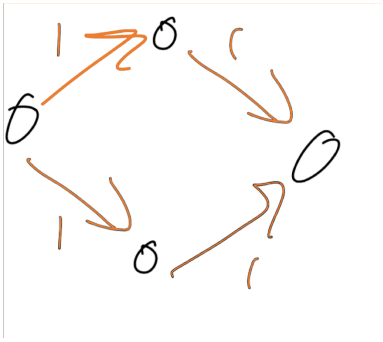


Figure 10.5: The flow depicted routes two units from s to t.

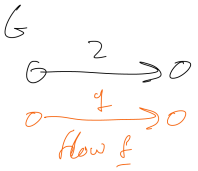
A better approach. It turns out we can fix the problem with the previous algorithm using a simple fix. This idea is known as *residual graphs*.

Algorithm 5: Better Idea (Residual Graph)

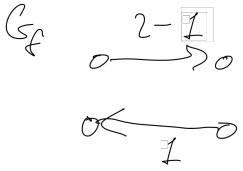
1 $\boldsymbol{f} \leftarrow \mathbf{0}$;
2 **repeat**
3 Find an s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible with respect to $-\boldsymbol{f} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$.
4 $\boldsymbol{f} \leftarrow \boldsymbol{f} + \tilde{\boldsymbol{f}}$

The $-\boldsymbol{f}(e)$ can be treated as an edge going in the other direction with capacity $\boldsymbol{f}(e)$. By convention, an edge in G with $\boldsymbol{f}(e) = \boldsymbol{c}(e)$ is called *saturated*, and we do not include the edge in G_f. The graph defined above with such capacities is called **the residual graph of $\boldsymbol{f}$**, $G_{\boldsymbol{f}}$. $G_{\boldsymbol{f}}$ is only defined for feasible $\boldsymbol{f}$, since otherwise the constraint $\tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$ gives trouble.

Suppose we start with a graph having a single edge with capacity 2 and we introduce a flow of 1 unit.



The residual graph $G_{\boldsymbol{f}}$ has an edge with capacity 1 going forward and -1 capacity going forward, but we can treat the latter as $+1$ capacity going backwards. So it is an edge that allows you to undo the choice made to send flow along that direction.



Let us consider the same example with its residual graph. The original graph is shown in Figure 10.6.

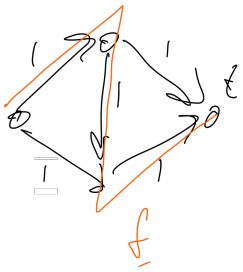


Figure 10.6: Original graph G and an s-t path flow in G shown in orange.

The residual graph for the same is shown in Figure 10.7.

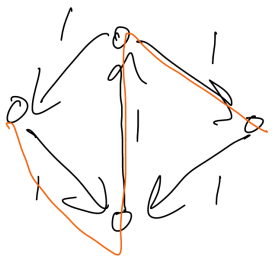


Figure 10.7: The residual graph w.r.t. the flow from Figure 10.6, and new s-t path flow which is feasible in the residual graph.

Adding both flows together, we get the paths as shown in Figure 10.8 with value 2, which is the optimum.

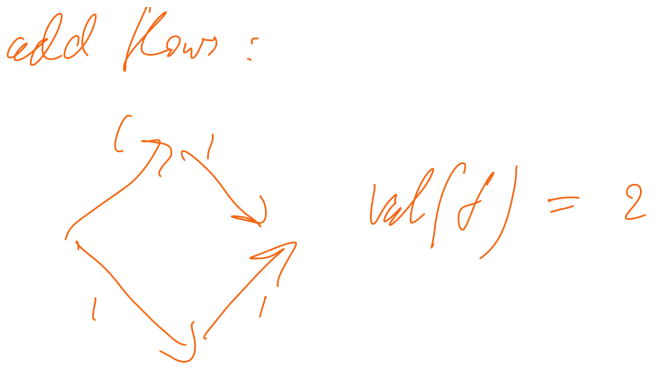


Figure 10.8: Maximum flow in the graph.

Let's prove some important properties of residual graphs.

Lemma 10.4.1. *Suppose $\boldsymbol{f}$, $\hat{\boldsymbol{f}}$ are feasible in G. Then this implies that $\hat{\boldsymbol{f}} - \boldsymbol{f}$ (where negative entries count as flow in opposite direction) is feasible in $G_{\boldsymbol{f}}$.*

Proof. $\mathbf{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$ and $\mathbf{0} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c}$, this implies $-\boldsymbol{f} \leq \tilde{\boldsymbol{f}} - \boldsymbol{f} \leq \boldsymbol{c} - \boldsymbol{f}$. Hence, proved. □

Lemma 10.4.2. *Suppose that $\boldsymbol{f}$ is feasible in G and $\hat{\boldsymbol{f}}$ is feasible in $G_{\boldsymbol{f}}$. Then, $\boldsymbol{f} + \hat{\boldsymbol{f}}$ is feasible in G.*

Proof. $\mathbf{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$ and $-\boldsymbol{f} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$, this implies $\mathbf{0} \leq \tilde{\boldsymbol{f}} + \boldsymbol{f} \leq \boldsymbol{c}$. □

Lemma 10.4.3. *A feasible $\boldsymbol{f}$ is optimal if and only if t is not reachable from s in $G_{\boldsymbol{f}}$.*

Proof. Let $\boldsymbol{f}$ be optimal, and suppose t is reachable from s in $G_{\boldsymbol{f}}$ then, we can find a s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible in $G_{\boldsymbol{f}}$, and $\text{val}(\boldsymbol{f} + \tilde{\boldsymbol{f}}) > \text{val}(\boldsymbol{f})$. $\boldsymbol{f} + \tilde{\boldsymbol{f}}$ is feasible in G by Lemma 10.4.2. This is a contradiction, as we assumed $\boldsymbol{f}$ was supposed to be optimal.

Suppose t is not reachable from s in $G_{\boldsymbol{f}}$, and $\boldsymbol{f}$ is feasible, but not optimal. Let $\boldsymbol{f}^*$ be optimal, then by Lemma 10.4.1, the flow $\boldsymbol{f}^* - \boldsymbol{f}$ is feasible in $G_{\boldsymbol{f}}$ and $\text{val}(\boldsymbol{f}^* - \boldsymbol{f}) > 0$. So there exists a s-t path flow from s to t in $G_{\boldsymbol{f}}$ (as we can do a path decomposition of $\boldsymbol{f}^* - \boldsymbol{f}$). But, this is a contradiction as t is not reachable from s in $G_{\boldsymbol{f}}$. □

Theorem 10.4.4 (Max Flow = Min Cut theorem)**.** *The maximum flow in a directed graph G equals the minimum cut.*

Proof. Consider the set $S = \{\text{vertices reachable from } s \text{ in } G_{\boldsymbol{f}}\}$. Note that $\boldsymbol{f}$ saturates the edge capacities in cut $S, V \setminus S$ in G: Consider any edge from S to $T := V \setminus S$ in G. Since this edge does not exist in the residual graph, we must have $\boldsymbol{f}(e) = \boldsymbol{c}(e)$.

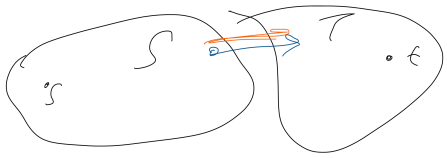


Figure 10.9: The cut between vertices reachable from S and everything else in G_f must have all outgoing edges saturated by $\boldsymbol{f}$.

This means that

$$\text{val}(\boldsymbol{f}) \geq c_G(S, V \setminus S).$$

Since we already know the opposite inequality by weak duality, we have shown that

$$\text{val}(\boldsymbol{f}) = c_G(S, V \setminus S)$$

which proves the **Max Flow = Min Cut** theorem; also called strong duality. □

Remark 10.4.5. Note that this proof also gives an algorithm to compute an s, t-min cut given an s, t maximum flow $\boldsymbol{f}^*$: Take one side of the cut to be the set S of nodes reachable from s in the residual graph w.r.t. $\boldsymbol{f}^*$.

Ford-Fulkerson Algorithm.

Algorithm 6: Ford-Fulkerson Algorithm

```
1 repeat
2 |  Add update by arbitrary s-t path flow in G_f (augment the flow f by the path flow)
```

Convergence properties and analysis of runtime of Ford-Fulkerson algorithm

- **Does this algorithm terminate?**
  The algorithm terminates if the capacities are integers. However for irrational capacities the algorithm may not terminate.

- **Does it converge towards the max flow value?**
  No, it does not converge to max flow value if the updates are poor and the capacities are irrational.

Lemma 10.4.6. *Consider Ford-Fulkerson algorithm with integer capacities. The algorithm terminates in* $\text{val}(\boldsymbol{f}^*)$ *augmentations i.e.* $O(m \, \text{val}(\boldsymbol{f}^*))$ *time.*

Proof. Each iteration increases the flow by at least one unit as the capacities in $G_{\boldsymbol{f}}$ are integral and each iteration can be computed in $O(m)$ time. □

Can we do better than this? Suppose we pick the maximum bottleneck capacity (minimum capacity along path) augmenting path. This gives an algorithm that is better in some regimes.

How to pick the maximum bottleneck capacity augmenting path? We are going to perform a binary search on the capacities in $G_{\boldsymbol{f}}$, to find a path with maximum bottleneck capacity. Each time our binary search has picked a threshold capacity, we then try to find an s-t path flow in $G_{\boldsymbol{f}}$ using only edges with capacity above that threshold. If we find a path, the binary search tries a higher capacity next. If we don't find a path, the binary search tries a lower capacity next.

Using this approach, the time to find a single path is $O(m \log(n))$ where m is number of edges in the graph. This path must carry at least a $\frac{1}{m}$ fraction of the total amount of flow left in $G_{\boldsymbol{f}}$. For instance, if $\hat{F}$ is the amount of flow left in $G_{\boldsymbol{f}}$, then the path must carry $\frac{\hat{F}}{m}$ flow (from the path decomposition lemma, we know that there exists a decomposition into at most m paths, and the one carrying the most flow must carry at least the average amount).

So if the flow is integral, the algorithms completes when

$$\left(1 - \frac{1}{m}\right)^T \text{val}(\boldsymbol{f}^*) < 1$$

where T is the number of augmentations.

This means

$$T = m \log F$$

$$\begin{aligned}
\text{Total time} &= O(m \log nT) \\
&= O(m^2 \log n \log F) \\
&\leq O(m^2 \log n \log mU) \text{ as } F \leq mU \text{ where U is the maximum capacity}
\end{aligned}$$

Current state-of-the-art approaches for Max Flow. For the interested reader, we'll briefly mention the current state-of-the-art for Maximum Flow algorithms, which is a very active area of research.

- Strongly polynomial time algorithm:
  - has to work with real valued capacities, then the best time $O(mn)$ by Orlin.
- General integer capacities: For a long time, the state-of-the-art was a result by Goldberg and Rao achieving a runtime of $O(m \min(\sqrt{m}, n^{2/3}) \log(n) \log(U))$ where U is the maximum capacity. Very recently, an algorithm that runs in almost-linear time $m^{1+o(1)} \log(U)$ was given (see `https://arxiv.org/abs/2203.00671`). There are many other recent results prior to this work that we don't have time to cover here.

Chapter 11

Classical Algorithms for Maximum Flow II

11.1 Overview

In this chapter we continue looking at classical max flow algorithms. We derive Dinic's algorithm which (unlike Ford-Fulkerson) converges in a polynomial number of iterations. We will also show faster convergence on unit capacity graphs. The setting is the same as last chapter: we have a directed graph $G = (V, E, \boldsymbol{c})$ with positive capacities on the edges. Our goal is to solve the maximum flow problem on this graph for a given source s and sink $t \neq s$:

$$\begin{aligned} \max_{F>0, \boldsymbol{f}} \ F \qquad \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{s,t} \\ & \boldsymbol{0} \leq \boldsymbol{f} \leq \boldsymbol{c} \end{aligned} \tag{11.1}$$

11.2 Blocking Flows

Let $\boldsymbol{f}$ be any feasible flow in G and let $G_{\boldsymbol{f}}$ be its residual graph. Observe that we can partition the vertices of $G_{\boldsymbol{f}}$ into 'layers': first s itself, then all vertices reachable from s in one hop, then all vertices reachable in two hops, etc. For each vertex $v \in V$ define its **level** in $G_{\boldsymbol{f}}$ as the length of the shortest path in $G_{\boldsymbol{f}}$ from s to v, denoted by $\ell_{G_{\boldsymbol{f}}}(v)$ (or just $\ell(v)$ if the graph is clear from context). An edge $(u, v) \in E$ can only take you up 'one level': if $\ell(v) \geq 2 + \ell(u)$ this would imply we can find a shorter s-v path by appending (u, v) to the shortest s-u path. However, edges can 'drop down' multiple levels (or be contained in the same level).

A key strategy in Dinic's algorithm will be to focus on 'short' augmenting paths. We can use the levels we defined above to isolate a subgraph of $G_{\boldsymbol{f}}$ containing all information we

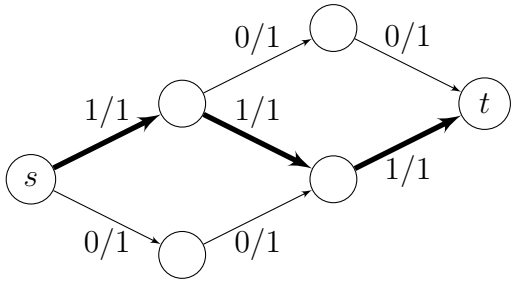


Figure 11.1: A blocking flow is not a maximum flow.

need to find shortest paths.

Definition 11.2.1. Call an edge (u, v) **admissible** if $\ell(u) + 1 = \ell(v)$. Let L be the set of admissible edges in $G_{\boldsymbol{f}}$ and define the **level graph** of $G_{\boldsymbol{f}}$ to be the subgraph induced by these edges.

Definition 11.2.2. Define a **blocking flow** in a residual graph $G_{\boldsymbol{f}}$ to be a flow $\hat{\boldsymbol{f}}$ feasible in $G_{\boldsymbol{f}}$, such that $\hat{\boldsymbol{f}}$ only uses admissible edges. Furthermore we require that for any s-t path in the level graph of $G_{\boldsymbol{f}}$, $\hat{\boldsymbol{f}}$ saturates at least one edge on this path.

The last condition makes a blocking flow 'blocking': it blocks any shortest augmenting path in $G_{\boldsymbol{f}}$ by saturating one of its edges. Note however that a blocking flow is not necessarily a maximum flow in the level graph.

11.3 Dinic's Algorithm

We can now formulate Dinic[1]'s algorithm: start with $\boldsymbol{f} = \boldsymbol{0}$, and then repeatedly add to $\boldsymbol{f}$ a blocking flow $\hat{\boldsymbol{f}}$ in $G_{\boldsymbol{f}}$, until no more s-t paths exist in $G_{\boldsymbol{f}}$.

Note that by doing a path decomposition on $\hat{\boldsymbol{f}}$ and adding these paths to $\boldsymbol{f}$ one by one, we see that our algorithm is a 'special case' of Ford-Fulkerson (with a particular augmenting path strategy) and hence inherits all the behaviors/bounds we proved in the last chapter.

Lemma 11.3.1. *Let $\boldsymbol{f}$ be a feasible flow, $\hat{\boldsymbol{f}}$ a blocking flow in $G_{\boldsymbol{f}}$ and define $\boldsymbol{f}' = \boldsymbol{f} + \hat{\boldsymbol{f}}$ (think of $\boldsymbol{f}$ and $\boldsymbol{f}'$ to be the flows at some steps k and $k+1$ in the algorithm). Then $\ell_{G_{\boldsymbol{f}'}}(t) \geq \ell_{G_{\boldsymbol{f}}}(t) + 1$.*

Proof. Let L, L' be the edge sets of the level graphs of $G_{\boldsymbol{f}}, G_{\boldsymbol{f}'}$ respectively. We would now like to show that $d_{L'}(s,t) \geq 1 + d_L(s,t)$. Let's first assume that in fact $d_{L'}(s,t) < d_L(s,t)$. Now take a shortest s-t path in the level graph of $G_{\boldsymbol{f}'}$, say $s = v_0, v_1, \ldots, v_{d_{L'}(s,t)} = t$. Let v_j be the first vertex along the path such that $d_{L'}(s, v_j) < d_L(s, v_j)$. As $d_{L'}(s,t) < d_L(s,t)$, such a v_j must exist.

[1]Sometimes also transliterated as Dinitz.

We'd like to understand the edges in the level graph L'. Let $E(G_{\boldsymbol{f}})$ denote the edges of the $G_{\boldsymbol{f}}$, and let $\text{rev}(L)$ denote the set of reversed edges of L. The level graph edges of $G_{\boldsymbol{f}'}$ must satisfy $L' \subseteq L \cup \text{rev}(L) \cup (E(G_{\boldsymbol{f}}) \setminus L)$.

In our s-t path in L', the vertex v_j is reached by an edge from v_{j-1}, and the level of v_{j-1} in $G_{\boldsymbol{f}}$ is at most the level it receives in $G_{\boldsymbol{f}'}$ so the edge $(v_{j-1}, v_j) \in L$ skips at least one level forward in $G_{\boldsymbol{f}}$. But, edges in L do not skip a level in $G_{\boldsymbol{f}}$, and edges in $\text{rev}(L)$ or $E(G_{\boldsymbol{f}}) \setminus L$ do not move from a lower level to higher level in $G_{\boldsymbol{f}}$. So this edge cannot exist and we have reached a contradiction.

Now suppose that $d_{L'}(s,t) = d_L(s,t)$. This means there is an s-t path in L' using edges in L – but such a path must contain an edge saturated by the blocking flow $\hat{\boldsymbol{f}}$.

(Note: the reason this path must use only edges in L is similar to the previous case: the other possible types of edges do not move to higher levels in $G_{\boldsymbol{f}}$ at all, making the path too long if we use any of them.)

So we must have $d_{L'}(s,t) \geq 1 + d_L(s,t)$. □

An immediate corollary of this lemma is the convergence of Dinic's algorithm:

Theorem 11.3.2. *Dinic's algorithm terminates in $O(n)$ iterations.*

Proof. By the previous lemma, the level of t increases by at least one each iteration. A shortest path can only contain each vertex once (otherwise it would contain a cycle) so the level of any vertex is never more than n. □

For graphs with unit capacities ($c = 1$) we can prove even better bounds on the number of iterations.

Theorem 11.3.3. *On unit capacity graphs, Dinic's algorithm terminates in*

$$O\left(\min\{m^{1/2}, n^{2/3}\}\right)$$

iterations.

Proof. We prove the two bounds separately.

1. Suppose we run Dinic's algorithm for k iterations, obtaining a flow $\boldsymbol{f}$ that is feasible but not necessarily yet optimal. Let $\tilde{\boldsymbol{f}}$ be an optimal flow in $G_{\boldsymbol{f}}$ (i.e. $\boldsymbol{f} + \tilde{\boldsymbol{f}}$ is a maximum flow for the whole graph), and consider a path decomposition of $\tilde{\boldsymbol{f}}$. Since after k iterations any s-t path has length k or more, we use up a total capacity of at least $\text{val}(\tilde{\boldsymbol{f}})k$ across all edges. But the edges in $G_{\boldsymbol{f}}$ are either edges from the original graph G or their reversals (but never both) meaning the total capacity of $G_{\boldsymbol{f}}$ is at most m, hence $\text{val}(\tilde{\boldsymbol{f}}) \leq m/k$.

   Recalling our earlier observation that our algorithm is a special case of Ford-Fulkerson, this implies that our algorithm will terminate after at most another m/k iterations.

Hence the number of iterations is bounded by $k + m/k$ for any $k > 0$. Substituting $k = \sqrt{m}$ gives the first desired bound.

2. Suppose again that we run Dinic's algorithm for k iterations obtaining a flow $\boldsymbol{f}$. The level graph of $G_{\boldsymbol{f}}$ partitions the vertices into sets $D_i = \{u \mid \ell(u) = i\}$ for $i \geq 0$. As shown before, the sink t must be at a level of at least k, meaning we have at least this many non-empty levels starting from $D_0 = \{s\}$. To simplify, discard all vertices and levels beyond $D_{\ell(t)}$.

   Now, consider choosing a level I uniformly at random from $\{1, \ldots, k\}$. Since there are at most $n-1$ vertices in total across these levels, $\mathbb{E}[|D_I|] \leq (n-1)/k$, and by Markov's inequality

$$\Pr[|D_I| \geq 2n/k] \leq \frac{(n-1)/k}{2n/k} < \frac{1}{2}.$$

   Thus strictly more than $k/2$ of the levels $i \in \{1, \ldots, k\}$ have $|D_i| < 2n/k$ and so there must be two adjacent levels j, $j+1$ for which this upper bound on the size holds. There can be at most $|D_j| \cdot |D_{j+1}| \leq 4n^2/k^2$ edges between these levels, and by the min-cut max-flow theorem we saw in the previous chapter, this is an upper bound on the flow in $G_{\boldsymbol{f}}$, and hence on the number of iterations still needed for our algorithm to terminate.

   This means the number of iterations is bounded by $k + 4n^2/k^2$ for any $k > 0$, which is $O(n^{2/3})$ at $k = 2n^{2/3}$.

□

11.4 Finding Blocking Flows

What has been missing from our discussion so far is the process of actually finding a blocking flow. We can achieve this using repeated depth-first search. We repeatedly do a search in the level graph (so only using edges L) for s-t paths and augment these. We erase edges whose subtrees have been exhausted and do not contain any augmenting paths to t. Pseudocode for the algorithm is given below.

Algorithm 7: FindBlockingFlow

```
1  f ← 0;
2  H ← L;
3  repeat
4  |  P ← Dfs(s, H, t);
5  |  if P ≠ ∅ then
6  |  |  Let f̂ be a flow that saturates path P.
7  |  |  f ← f + f̂;
8  |  |  Remove from H all edges saturated by f̂.
9  |  else
10 |  |  return f;
11 |  end
```

Algorithm 8: Dfs(u, H, t)

```
1  if u = t then
2  |  return the path P on the dfs-stack.
3  end
4  for (u, v) ∈ H do
5  |  P ← Dfs(v, H, t);
6  |  if P ≠ ∅ then
7  |  |  return P;
8  |  else
9  |  |  Erase (u, v) from H.
10 |  end
11 end
12 return ∅;
```

Lemma 11.4.1. *For general graphs,* FindBlockingFlow *returns a blocking flow in $O(nm)$ time. Hence Dinic's algorithm runs in $O(n^2m)$ time on general capacity graphs.*

Proof. First, consider the amount of work spent pushing edges onto the stack which eventually results in augmentation along an s-t path consisting of those edges (i.e. adding flow along that path). Since each augmenting path saturates at least one edge, we do at most m augmentations. Each path has length at most n. Thus the total amount of work pushing these edges to the stack, and removing them from the stack upon augmentation, and deleting saturated edges, can be bounded by $O(mn)$. Now consider the work spent pushing edges onto the stack which are later deleted because we "retreat" after not finding an s-t. An edge can only be pushed onto the stack once in this way, since it is then deleted. So the total amount spent pushing edges to the stack and deleting them this way is $O(m)$. □

Lemma 11.4.2. *For unit capacity graphs,* FindBlockingFlow *returns a blocking flow in $O(m)$ time. Hence Dinic's algorithm runs in $O\left(\min\{m^{3/2}, mn^{2/3}\}\right)$ time on unit capacity graphs.*

Proof. When our depth-first search traverses some edge (u, v), one of two things will happen: either we find no augmenting path in the subtree of v, leading to the erasure of this edge, or we find an augmenting path which will necessarily saturate (u, v), again leading to its erasure. This means each edge will be traversed at most once by the depth-first search. □

Another interesting bound on the runtime, which we will not prove here, is that Dinic's algorithm will run in $O\left(|E|\sqrt{|V|}\right)$ time on bipartite matching graphs.

11.5 Minimum Cut as a Linear Program

Finally we show that minimum cut may be formulated as a linear program. For a subset $S \subseteq V$ write $c_G(S)$ for the sum of $\boldsymbol{c}(e)$ over all edges $e = (u, v)$ that cross the cut, i.e. $u \in S, v \notin S$. Note that 'reverse' edges are not counted in this cut. The minimum cut problem asks us to find some $S \subseteq V$ such that $s \in S$ and $t \notin S$ such that $c_G(S)$ is minimal.

$$\begin{aligned} \min_{S \subseteq V} \quad & c_G(S) \\ s.t. \quad & s \in S \\ & t \notin S \end{aligned} \tag{11.2}$$

We claim this is equivalent to the following minimization problem:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^V} \quad & \sum_{e \in E} \boldsymbol{c}(e) \max\left\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\right\} \\ s.t. \quad & \boldsymbol{x}(s) = 0 \\ & \boldsymbol{x}(t) = 1 \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \end{aligned} \tag{11.3}$$

Recall $\boldsymbol{b}_e^\top \boldsymbol{x} = \boldsymbol{x}(v) - \boldsymbol{x}(u)$ for $e = (u, v)$. We can rewrite this as a proper linear program by introducing extra variables for the maximum:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^V, \boldsymbol{u} \in \mathbb{R}^E} \quad & \boldsymbol{c}^\top \boldsymbol{u} \\ s.t. \quad & \boldsymbol{b}_{s,t}^\top \boldsymbol{x} = 1 \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \\ & \boldsymbol{u} \geq \mathbf{0} \\ & \boldsymbol{u} \geq \boldsymbol{B}^\top \boldsymbol{x} \end{aligned} \tag{11.4}$$

And, in fact, we'll also see that we don't need the constraint $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$, leading to the

following simpler program:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^V, \boldsymbol{u}\in\mathbb{R}^E} \quad & \boldsymbol{c}^\top \boldsymbol{u} \\ s.t. \quad & \boldsymbol{b}_{s,t}^\top \boldsymbol{x} = 1 \\ & \boldsymbol{u} \geq \boldsymbol{0} \\ & \boldsymbol{u} \geq \boldsymbol{B}^\top \boldsymbol{x} \end{aligned} \tag{11.5}$$

This last linear program is the dual program to the maximum flow linear program.

Lemma 11.5.1. *Programs* (11.2), (11.4), *and* (11.5) *have equal optimal values.*

Proof. We start by considering equality between optimal values of Program (11.2) and Program (11.4) Let S be an optimal solution to Program (11.2) and take $\boldsymbol{x} = \mathbf{1}_{V\setminus S}$. Then $\boldsymbol{x}$ is a feasible solution to Program (11.4) with cost equal to $c_G(S)$.

Conversely, suppose $\boldsymbol{x}$ is an optimal solution to Program (11.4). Let α be uniform in $[0, 1]$ and define $S = \{v \in V \mid \boldsymbol{x}(v) \leq \alpha\}$. This is called a threshold cut. We can verify that

$$\Pr[e \text{ is cut by } S] = \max\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\}.$$

and hence $\mathbb{E}_t[c_G(S)]$ is exactly the optimization function of 11.3 (and hence 11.4). Since at least one outcome must do as well as the average, there is a subset $S \subseteq V$ achieving this value (or less).

We can use the same threshold cut to show that we can round a solution to Program (11.5) to an equal or smaller value cut feasible for Program (11.2). The only difference is that in this case, we get

$$\Pr[e \text{ is cut by } S] \leq \max\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\}$$

which is still sufficient. □

Chapter 12

Link-Cut Trees

In this chapter, we will learn about a dynamic data structure that allows us to speed-up Dinic's algorithm even more: *Link-Cut Trees*. This chapter is inspired by lecture notes of Richard Peng[1] and the presentation in a very nice book on data structures and network flows by Robert Tarjan [Tar83].

12.1 Overview

Model. We consider a directed graph $G = (V, E)$ that is undergoing *updates* in the form of edge insertions and/or deletions. We number the graph in its different *versions* $G^0, G^1, G^2, \ldots$ such that G^0 is the initial input graph, and G^i is the initial graph after the first i updates were applied. Such a graph is called a *dynamic* graph.

In this lecture, we restrict ourselves to *dynamic rooted forests* that is we assume that every G^i forms a directed forest where in each forest a single root vertex is reached by every other vertex in the tree. For simplicity, we assume that $G^0 = (V, \emptyset)$ is an empty graph.

The Interface. Let us now describe the interface of our data structure that we call a link-cut tree. We want to support the following operations:

- INITIALIZE(G): Creates the data structure initially and returns a pointer to it. Each vertex is initialized to have an associated cost $cost(v)$ equal to 0.
- FINDROOT(v): Returns the root of vertex v.
- ADDCOST(v, Δ): Add Δ to the cost of every vertex on the path from v to the root vertex FINDROOT(v).

[1] CS7510 Graph Algorithms Fall 2019, Lecture 17 at `https://faculty.cc.gatech.edu/~rpeng/CS7510_F19/`

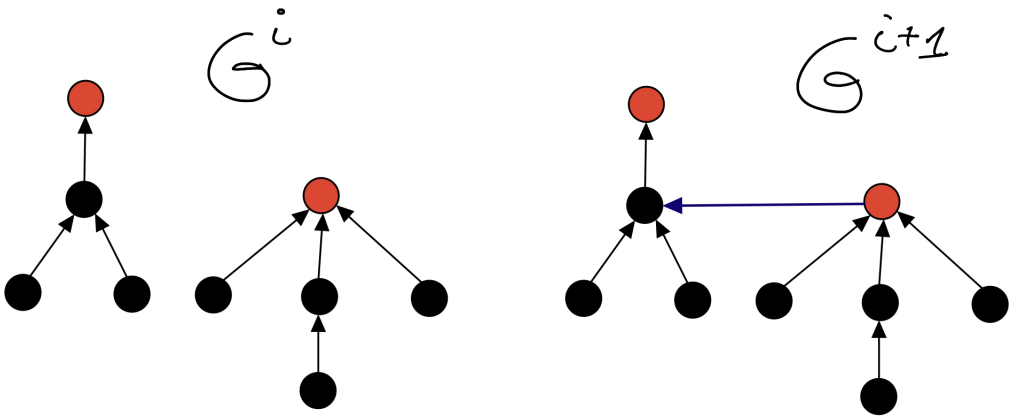


Figure 12.1: The i^{th} version of G is a rooted forest. Here, red vertices are roots and there are two trees. The $(i+1)^{th}$ version of G differs from G^i by a single edge that was inserted (the blue edge). Note that edge insertions are only valid if the tail of the edge was a root. In the right picture, the former root on the right side is turned into a normal node in the tree by the update.

- FindMin(v): Returns tuple $(w, cost(w))$ where w is the (first) vertex on the path from v to FindRoot(v) of lowest cost.
- Link(u, v): Links two trees that contain u and v into a single tree by inserting the edge (u, v). This assumes that u, v are initially in different trees and u was a root vertex.
- Cut(u, v): Cuts the edge (u, v) from the graph which causes the subtree rooted at u to become a tree and u to become a root vertex. Assumes (u, v) is in the current graph.

Main Result. The following theorem is the main result of today's lecture.

Theorem 12.1.1. *We can implement a link-cut tree such that any sequence of m operations takes total expected time $O(m \log^2 n + |V|)$.*

12.2 Balanced Binary Search Trees: A Recap

In Chapter 2 of the course Algorithms, Probability and Computing, we introduced (balanced) binary search trees, and studied *treaps*, which are a simple randomized approach to building a binary search tree with good performance, at least in expectation. If you are not familiar with balanced binary search trees, we encourage you to read this chapter from the APC script. That said, we will quickly recap the important properties of binary search trees and treaps. If you're already familiar with treaps, you can skip ahead to the next section to see how we use them for building Link-Cut trees.

Binary Search Trees. A binary search tree $\mathcal{T}$ is a data structure for keeping track of a set of *items* where each item has a *key* associated with it. The data structure stores the items in a rooted binary tree with a node for each item, and maintains the property for each node v, all items in the *left* subtree of v have keys less than v, and all items in the *right* subtree of v have keys greater than v. This is called the *search property* of the tree, because it makes it simple to search through the tree to determine if it contains an item with a given key.

The binary search trees we studied in APC supported the following operations, while maintaining the search property:

insert: We can add an item to the tree with a specified key.

delete: We can remove an item from the tree with a specified key.

find: Determine if the tree contains an item with a specified key.

split: Suppose the tree contains two items v_1 and v_2 with keys k_1 and k_2 where $k_1 < k_2$, and suppose no item in tree has a key k in the interval (k_1, k_2). Then the split operation applied to these two items should split the current tree into two: One tree containing all items with keys in $(-\infty, k_1]$ and the other with all items in $[k_2, \infty)$.

join: Given two binary search trees, one with keys in the interval $(-\infty, k]$, the other with keys in the interval (k, ∞), form a single binary search tree containing the items of both.

Search-Property-Preserving Tree rotations. In this chapter, we will use that enforcing the search property in a binary tree still leaves various options to have a tree. In fact, there is a classic tree operation that preserves the search property while changing the tree: Tree rotations.

A tree rotation makes a local change to the tree by changing only a constant number of pointers around, while preserving the search property. Given two items v_1 and v_2 such that v_1 is a child of v_2, a tree rotation applied to these two nodes will make v_2 the child of v_1, while moving their subtrees around to preserve the search property. This is shown in Figure 12.2. When the rotation makes v_2 the right child of v_1 (because v_2 has the larger key), this is called a *right rotation*, and when it makes v_2 the left child of v_1 (because v_2 has a smaller key), it is called a *left rotation*.

Balanced BSTs. To implement the operations given above efficiently, one needs to ensure that the BST always stays of bounded depth (normally $O(\log n)$). This is typically ensured by making tree rotations. In the rest of the chapter, we use a randomized strategy that ensures that the depth of the tree is $O(\log n)$ in expectation. The obtained data structure is often referred to as *treap*. We will not give the details here, but instead explicitly construct one in the next section and give a full (although brief) analysis.

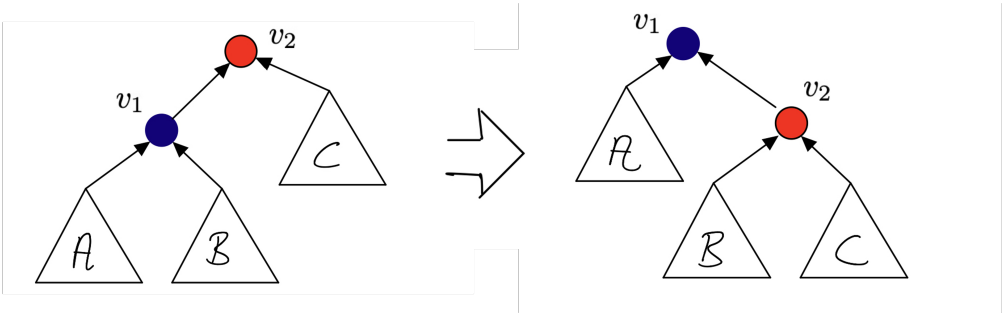


Figure 12.2: Given two items v_1 and v_2 such that v_1 is a child of v_2, a tree rotation applied to these two nodes will make v_2 the child of v_1, while moving their subtrees around to preserve the search property. This figure shows a *right rotation*.

12.3 A Data Structure for Path Graphs

Before we prove Theorem 12.1.1 in its full generality, let us reason about implementing a link-cut tree data structure in a weaker setting: we assume that every version G^i of G is just a collection of rooted vertex-disjoint paths. We will later build on the routines we develop for the path case to construct the data structure for the general tree case. To distinguish the routines we develop for paths from those we develop for the general case, we will prefix the path-case routines with a "P", i.e. we denote our implementations of FindRoot, AddCost, FindMin, Link, and Cut by PFindRoot, PAddCost, PFindMin, PLink, and PCut.

Representing Paths via Balanced Binary Search Trees. It turns out that paths can be represented rather straight-forwardly via Balanced Binary Search Trees, with an node/item for each vertex of the graph. For the sake of concreteness, we here use *treaps* to represent paths[2]. Most other balanced binary search trees would also work.

Note that we now represent each rooted path with a tree, and this tree has its own root, which is usually *not* the root of the path. To minimize confusion, we will refer to the root of the path as the *path-root* and the root of the associated tree as the *tree-root* (or *subtree-root* for the root of a subtree).

In our earlier discussion of binary trees, we always assumed each item has a key: Instead we will now let the key of each vertex correspond to its distance to the path-root in the path that contains it. We will make sure the treap respects the search property w.r.t. this set of keys/ordering, but we will not actually explicitly compute these keys or store them. Note one important difference to the scenario of treaps-with-keys: When we have two paths, with path-roots r_1 and r_2, we will allow ourselves to join these paths in two different ways: either so that r_1 or r_2 becomes the overall path-root.

[2]If you have not seen treaps before, don't worry, they are simple enough to understand them from our application here.

Let us describe how to represent a path P in G. First, we pick for each vertex v, we assign it a *priority* denoted prio(v), which we choose to be a uniformly random integer sampled from a large universe, say $[1, n^{100})$. We assume henceforth that prio(v) $\neq$ prio(w) for all $v \neq w \in V$.

Then, for each path P in G, we store the vertices of P in a binary tree, in particular a treap, $\mathcal{T}_P$ and enforce the following invariants:

- **Search-Property**: for all v, $left_{\mathcal{T}_P}(v)$ precedes v on P and $right_{\mathcal{T}_P}(v)$ appears later on P than v. See the Figure 12.3 below for an illustration.
- **Heap-Order**: for each vertex $v \in P$, its parent $w = parent_{\mathcal{T}_P}(v)$ in $\mathcal{T}_P$ is either $NULL$ (if v is the path-root) or has prio(v) $>$ prio(w).

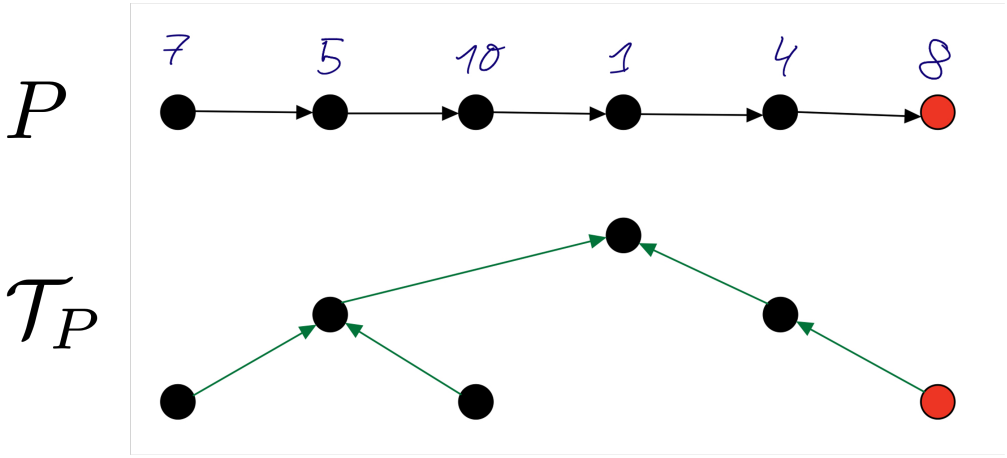


Figure 12.3: In the upper half of the picture, the original path P in G is shown, along with the random numbers prio(v) for each v. The lower half depicts the resulting treap with vertices on the same vertical line as before.

Depth of Vertex in a Treap. Let us next analyze the expected depth of a vertex v in a treap $\mathcal{T}_P$ representing a path P. Let $P = \langle x_1, x_2, \ldots, x_k = v, \ldots, x_{|P|} \rangle$, i.e. v is the k^{th} vertex on the path P. Observe that a vertex x_i with $i < k$ is an ancestor of v in $\mathcal{T}_P$ if and only if no vertex $\{x_{i+1}, x_{i+2}, \ldots, x_k\}$ has received a smaller random number than prio(x_i). Since we sample prio(w) uniformly at random for each w, we have that $\mathbb{P}[x_i \text{ ancestor of } v] = \frac{1}{k-i+1}$. The case where $i > k$ is analogous and has $\mathbb{P}[x_i \text{ ancestor of } v] = \frac{1}{i-k+1}$. Letting X_i be the indicator variable for the event that x_i is an ancestor of v, it is straight-forward to calculate the expected depth of v in $\mathcal{T}_P$:

$$\mathbb{E}[depth(v)] = \sum_{i \neq k} \mathbb{E}[X_i] = \sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{|P|} \frac{1}{i-k+1} = H_k + H_{|P|-k+1} - 2 = O(\log |P|)$$

It is straight-forward to see that the operation PFindRoot(v) can thus be implemented to run in expected $O(\log n)$ time by just iteratively following the parent pointers starting in v.

Implementing PFindRoot(v). From any vertex v, we can simply follow the parent pointers in $\mathcal{T}_P$ until we are at the tree-root of $\mathcal{T}_P$. Then, we find the right-most child of $\mathcal{T}_P$ by following the $right_{\mathcal{T}_P}$ pointers. Finally, we return the right-most child which is the path-root. Since the paths from v is at depth $O(\log n)$ in expectation, and so is the path-root of v, we have that this operation can be executed in $O(\log n)$ expected time.

Extra fields to help with PAddCost(v, Δ) and PFindMin(v). The key trick to do the Link-Cut tree operations efficiently is to store the change to subtrees instead of updating $cost(v)$ for each affected vertex.

To this end, we store two fields $\Delta cost(v)$ and $\Delta min(v)$ for every vertex v. We let $cost(v)$ be the cost of each vertex, and $mincost(v)$ denote the minimum cost of any descendant of v in $\mathcal{T}_P$ (where we let v be a descendant of itself). A warning: the $mincost(v)$ value is the minimum cost in the treap-subtree; *not* the minimum cost on the path between v and the path-root that could be different. Note also that we do not explicitly maintain these fields.

Then, we maintain for each v

$$\Delta cost(v) = cost(v) - mincost(v)$$

$$\Delta min(v) = \begin{cases} mincost(v) & parent_{\mathcal{T}_P}(v) = NULL, \\ mincost(v) - mincost(parent_{\mathcal{T}_P}(v)) & otherwise \end{cases}$$

With a bit of work, we can see that with these definitions, we obtain $mincost(v) = \sum_{w \in \mathcal{T}_P[v]} \Delta min(w)$ where $\mathcal{T}_P[v]$ is the v-to-tree-root path in $\mathcal{T}_P$. We can then recover $cost(v) = mincost(v) + \Delta cost(v)$. We say that henceforth, that an operation/manipulation to the tree is *field-preserving* if the fields $\Delta min(v)$ and $\Delta cost(v)$ satisfy the equations above.

Implementing PAddCost(v, Δ) and PFindMin(v) – the easy way. First, we will discuss ways to implement PAddCost and PFindMin in ways that rely on the fact that PLink and PCut are field-preserving.

PAddCost(v, Δ) is very easy: First, consider the case when v has no predecessor on the path. If this is the case, we can simply find the tree-root of the tree $\mathcal{T}_P$ containing v and at this root increase the value of the field *mincost* by Δ. This implicitly increases the cost of all nodes in the tree by Δ. Second, consider the case when v has a predecessor u in the path (we can store these explicitly, or we can search for it in the tree). Call PCut(u, v) – now v no longer has a predecessor and we can proceed as before. Finally, call PLink(u, v). That's it.

Implemeting PFindMin(v) is also very easy: First, consider the case when v has no predecessor on the path. Again, we can find the tree-root r of the tree $\mathcal{T}_P$ containing v. By definition $mincost(r)$ is the minimum cost across the whole tree, and hence across the path between v and the path-root. We can also find the node with the minimum cost: Search in the tree, starting from the tree-root, and, if possible follow the child where $\Delta min(v) = 0$

(going left if both are eligible). Eventually, we get to a node where there is no child with $\Delta min(v) = 0$. This must be a minimizer. Now, to deal with the case where v has a predecessor, we do the same we did for PAddCost: Find the predecessor u, call PCut(u, v), then find the minimizer, finally call PLink(u, v).

Implementing PAddCost(v, Δ) and PFindMin(v) – the hard way. We can also implement PAddCost(v, Δ) and PFindMin(v) without relying on PLink and PCut, which is more efficient, but also more unpleasant[3].

To implement the operation PFindMin(v) more efficiently, consider the sequence of nodes on the path in the tree $\mathcal{T}_P$ (not the path P) between v and the path-root. First let $v = x_1, x_2, \ldots, x_k = r$ be the nodes leading to the tree-root if v is left of the tree root, and then let $y_1, \ldots, y_l$ be the remaining nodes on the tree path to the path-root. Observe that either the minimizer one of these nodes (which we can check by checking their Δmin), or it is in a right subtree of some x_i, or in any subtree of some y_j. So we just search through these subtrees for their minimizers (looking for $\Delta min(v) = 0$) and pick the best one. There will only be $O(\log n)$ trees in expectation.

Next, let us discuss the implementation of the operation PAddCost(v, Δ) which is given in pseudo-code above. The first for-loop of this algorithm ensures that the tree is again *field-preserving*. This is ensured by walking down the path from the path-root of v to v and whenever we walk to the left, we make sure to increase all costs in the right subtree by adding Δ to the subtree-root of the right subtree r in the form of adding it to $\Delta min(r)$. On the vertices along the path it updates the costs by using the $\Delta cost(\cdot)$ fields. It is easy to prove from this that thereafter the tree is *field-preserving*.

While after the first for-loop the values can be computed efficiently, we might now be in the situation that $\Delta min(w)$ or/and $\Delta cost(w)$ are negative for some vertices w. We may also violate the invariants we want to preserve on the relationship between the $\Delta min(w)$ or/and $\Delta cost(w)$ fields and the true *cost* and *mincost* values at each node. We recover through the second for-loop that at each vertex on the tree path from v to its path-root first computes the correct minimum in the subtree again (using the helper variable *minval*) and then adjusts all values in its left/right subtree and its own fields. Since we argued that v is at expected depth $O(\log(n))$, we can see that this can be done in expected $O(\log n)$ time.

Implementing PCut(u, v). Let us first assume that we have a dummy node d_0 with $\text{prio}(d_0) = 0$ in the vertex set. The trick is to first treat the operation as splitting the edge (u, v) into (u, d_0) and (d_0, v) by inserting the vertex d_0 in between u and v in the tree $\mathcal{T}_P$ as a leaf (this is always possible). Then, we can do standard binary tree rotations to re-establish the heap-order invariant (see Figure 12.4). It is not hard to see that after $O(\log n)$ tree rotations in expectation, the heap-order is re-established and d_0 is at new tree-root of $\mathcal{T}_P$. It remains to remove d_0 and make $left_{\mathcal{T}_P}(d_0)$ and $right_{\mathcal{T}_P}(d_0)$ tree-roots.

[3]You don't have to know these more complicated approaches for the exam, but we include them for the interested reader.

Algorithm 9: PAddCost(v, Δ)

```
1  Store the vertices on the path in the tree T_P between v and the path-root, i.e.
     ⟨v = x_1, x_2, ..., x_k = PFindRoot(v)⟩.
2  for i = k down to 1 do
3  |  Δcost(x_i) = Δcost(x_i) + Δ.
4  |  if (r ← right_{T_P}(x_i)) ≠ NULL AND (i = 0 OR x_{i-1} ≠ r) then
5  |  |  Δmin(r) ← Δmin(r) + Δ
6  |  end
7  end
8  for i = 1 up to k do
9  |  minval ← Δcost(x_i).
10 |  foreach w ∈ {left_{T_P}(x_i), right_{T_P}(x_i)}, w ≠ NULL do
   |    minval ← min{minval, Δmin(w)}.
11 |  Δmin(x_i) ← Δmin(x_i) + minval.
12 |  Δcost(x_i) ← Δcost(x_i) − minval.
13 |  foreach w ∈ {left_{T_P}(x_i), right_{T_P}(x_i)}, w ≠ NULL do
   |    Δmin(w) ← Δmin(w) − minval.
14 end
```

In the exercises for this week, we ask you to come up with the pseudo-code for *field-preserving* tree rotations that still executes each rotation in $O(1)$ time. Given this routine, we can then show that the implementation of PCut(u, v) is field-preserving overall. And since the number of rotations is bound by $O(\max\{depth(u), depth(v)\})$ and thus the time to execute the entire procedure is $O(\log n)$ in expectation.

Implementing PLink(u, v). Implementing PLink(u, v) could be done by reversing the process described above: we use a dummy vertex d_0 with priority 0, and add it as the right child of u. We use tree rotations to then obtain heap-order again. Now, we can make v the right child of d_0. Note that the resulting tree has the search and the heap-order properties.

It remains to get rid of d_0. To achieve this, we change its priority to ∞ (or say n^{100}), and then use tree rotations to obtain heap-order again. Since d_0 is the only vertex that violates the heap-order, this step results in d_0 being rotated downwards until it is a leaf. Then we can remove d_0 entirely.

Again, it is not hard to see that the implementation runs in $O(\log n)$ expected time.

Notes. The operations PLink/PCut can be implemented using almost all Balanced Binary Search Trees (especially the ones you have seen in your first courses on data structures). Thus, it is not hard to get a $O(\log n)$ worst-case time bound for all operations discussed above.

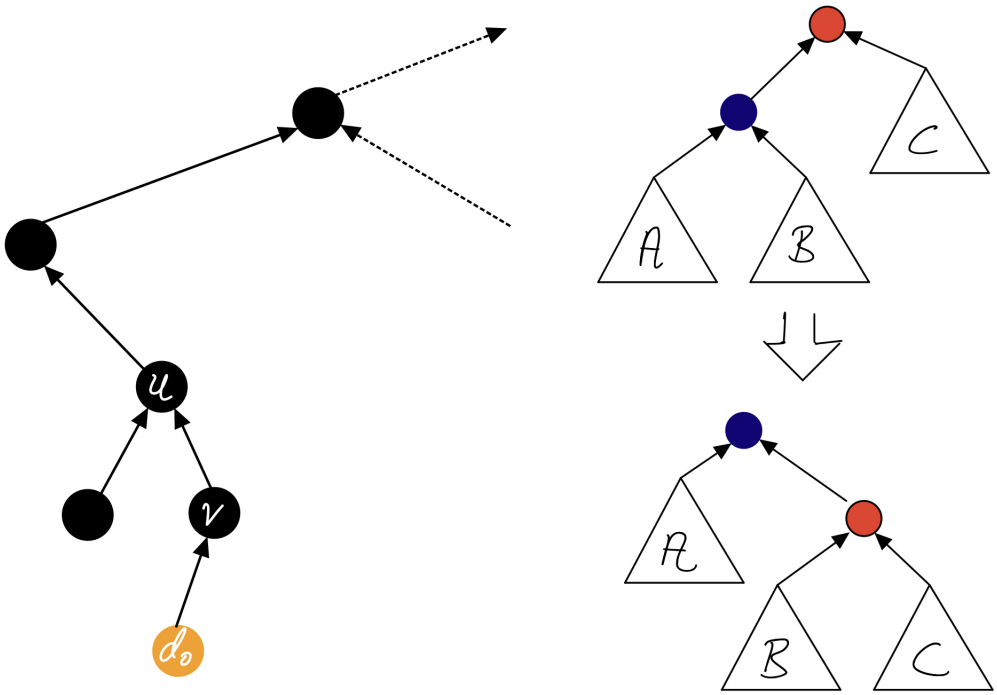


Figure 12.4: For PCut(u, v), we insert a vertex d_0 as a leaf of either u or v to formally split (u, v) into (u, d_0) and (d_0, v) (this is shown on the left). While this preserves that Search-Property, it will violate the Heap-Order. Thus, we need to use tree rotations (shown on the left) to push d_0 to the top of the tree $\mathcal{T}_P$ (the arrow between the tree rotations should point in both ways).

12.4 Implementing Trees via Paths

We now use the result from last section as a black box to obtain Theorem 12.1.1.

Path Decomposition. For each rooted tree T, the idea is to decompose T into paths. In particular, we decompose each T into a collection of vertex-disjoint paths $P_1, P_2, \ldots, P_k$ such that each internal vertex v in T has exactly one incoming edge in some P_i. We call the edges on some P_i *solid* edges and say that the other edges are *dashed*.

We maintain the paths $P_1, P_2, \ldots, P_k$ using the data structure described in the last section. To avoid confusion, we use the prefix P when we invoke operations of the path data structure, for example PFindRoot(v) finds the root of v in the path graph P_i where $v \in P_i$. We no longer need to think about the balanced binary trees that are used internally to represent each P_i. Instead, in this section, we use path-root to refer to the root of a path P_i and we use tree root (or just root) to refer to a root of one of the trees in our given collection of rooted trees.

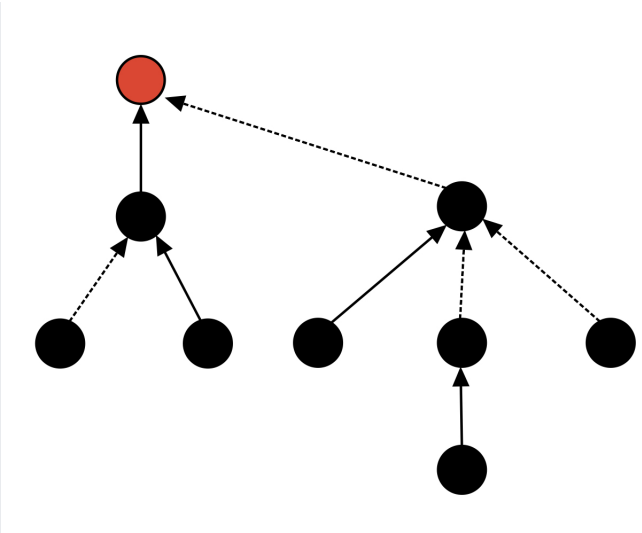


Figure 12.5: The dashed edges are the edges not on any path P_i. The collection of (non-empty) paths $P_1, P_2, \ldots, P_k$ can be seen to be the maximal path segments of solid edges.

The Expose(v) Operation. We start by discussing the most important operation of the data structure that will be used by all other operations internally: the operation Expose(v). This operation flips solid/dashed edges such that after the procedure the path from v to its tree root in G is solid (possibly as a subpath in the path collection). Below you can find an implementation of the procedure Expose(v).

Algorithm 10: Expose(v)

```
1 w ← v.
2 while (w' = parent_G(PFindRoot(w))) ≠ NULL do
3 |   Invoke PCut(z, w') for the solid edge (z, w') incoming to w'.
4 |   PLink(PFindRoot(w), w').
5 |   w ← w'.
6 end
```

Implementing Operations via Expose(v). We can now implement link-cut tree operations by invoking Expose(v) and then forwarding the operation to the path data structure.

Algorithm 11: AddCost(v, Δ)

```
1 Expose(v); PAddCost(v, Δ)
```

Algorithm 12: FindMin(v)

```
1 Expose(v); return PFindMin(v)
```

Algorithm 13: LINK(u,v)

1 $parent_G(u) \leftarrow v$;
2 **if** *v has an incoming solid edge (z, v)* **then** PCUT(z, v) ;
3 PLINK(u, v)

Algorithm 14: CUT(u,v)

1 EXPOSE(u); PCUT(u, v);
2 **if** *v has other incoming edge (z, v)* **then** PLINK(z, v);

Analysis. All of the operations above can be implemented using a single EXPOSE$(\cdot)$ operation plus $O(1)$ operations on paths. Since path operations can be executed efficiently, our main task is to bound the run-time of EXPOSE$(\cdot)$. More precisely, since each iteration of EXPOSE$(\cdot)$ also runs in time $O(\log n)$, the total number of while-loop iterations in EXPOSE$(\cdot)$.

To this end, we introduce a dichotomy over the vertices in G. We let $parent_G(v)$ denote the unique parent of v in G and let $size_G(v)$ denote the number of vertices in the subtree rooted at v (including v).

Definition 12.4.1. Then, we say that an edge (u, v) is *heavy* if $size_G(u) > size_G(v)/2$. Otherwise, we say (u, v) is *light*.

It can now be seen that the number of *light* edges on the v-to-root path for any v is at most $\lg n$: every time we follow a light edge (w, w'), i.e. when $size_G(w') > 2size_G(w)$, we double the size of the subtree so after taking more than $\lg n$ such edges, we have $> 2^{\lg n} = n$ vertices in the graph (which is a contradiction).

Thus, when EXPOSE$(\cdot)$ runs for many iterations, it must turn many *heavy* edges *solid* (this is also since each vertex has at most one incoming heavy edge, so when we make a heavy edge solid, we also don't make any other heavy edge solid).

Claim 12.4.2. *Each update can only increase the number of dashed, heavy edges by $O(\log n)$.*

Proof. First observe that every time EXPOSE$(\cdot)$ is invoked, it turns at most $\lg n$ heavy edges from solid to dashed (since it has to visit a light edge to do so).

The only two operations that can cause additional dashed, heavy edges are LINK(u, v) and CUT(u, v) by toggling heavy/light. For LINK(u, v), we observe that only the vertices on the v-to-root path increase their sizes. Since there are at most $\lg n$ light edges on this path that can turn heavy, this increases the number of dashed, heavy edges by at most $\lg n$.

The case for CUT(u, v) is almost analogous: only vertices on the v-to-root path decrease their sizes which can cause any heavy edge on such a path to become light, and instead for a sibling of such a vertex might becomes heavy. But there can be at most $\lg n$ such

new heavy edges, otherwise the total size of the tree must exceed again n which leads to a contradiction. □

Thus, after m updates, we have created at most $O(m \log n)$ dashed heavy edges. The iterations of EXPOSE$(\cdot)$ either visit a dashed light edge (at most $\ln n$ of them), or consumes a dashed heavy edge, i.e. turning them . We conclude that after m updates, the while-loop in EXPOSE$(\cdot)$ runs for at most $O(m \log n)$ iterations, *in total*, i.e. summed across the updates so far. Each iteration can be implemented in $O(\log n)$ expected time. This dominates the total running time and proves Theorem 12.1.1.

12.5 Fast Blocking Flow via Dynamic Trees

Recall from Section 11.4 that computing blocking flows in a level graph L from a vertex s to t can be done by successively running DFS(s) and routing flow along the s-t path found if one such path exists and otherwise we know that we have found a blocking flow.

We can now speed-up this procedure by storing the DFS-tree explicitly as a dynamic tree. To simplify exposition, we transform L to obtain a graph TRANSFORM(L) that has capacities on vertices instead of edges. To obtain TRANSFORM(L), we simply split each edge in L and assign the edge capacity to the mid-point vertex while assigning capacity ∞ to all vertices that were already in L. This creates an identical flow problem with at most $O(m)$ vertices and edges.

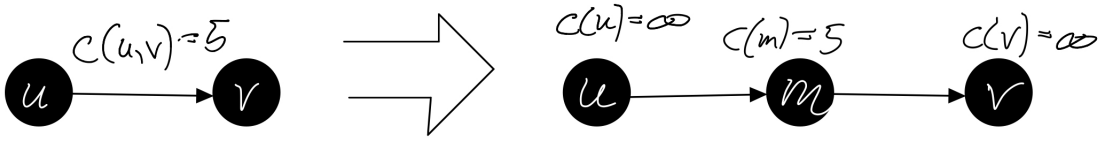


Figure 12.6: Each edge (u, v) with capacity $c(u, v)$ is split into two edges (u, m) and (m, v). The capacity is then on the vertex m.

Finally, we give the new pseudo-code for the blocking flow procedure below.

Algorithm 15: FindBlockingFlow(s, t, L)

```
H ← Transform(L);
LC-Tree ← Initialize(H);
while s ∈ H do
    u ← LC-Tree.FindRoot(s);
    if there is an edge (u, v) ∈ H then
        LC-Tree.Link(u, v);
        if LC-Tree.FindRoot(v) = t then
            (w, c) ← LC-Tree.FindMin(s);
            LC-Tree.AddCost(s, −c);
            Remove w and all its incident edges from H and LC-Tree (via Cut(·)).
        end
    else
        Remove u and all its incident edges from H and LC-Tree (via Cut(·)).
    end
end
Construct f by setting for each edge (u, v) of L, with mid-point m in Transform(L),
 the flow equal to c(m) minus the cost on m just before it was removed from H.
```

Claim 12.5.1. *The running time of* FindBlockingFlow(s, t, L) *is* $O(m \log^2 n + |V|)$.

Proof. Each edge (u, v) in the graph Transform(L) enters the link-cut tree at most once (we only invoke Cut(u, v) when we delete (u, v) from H).

Next, observe that the first **if**-case requires $O(1 + \#edgesDeletedFromH)$ many tree operations. The *else*-case requires $O(\#edgesDeletedFromH)$ many tree operations.

But each edge is only deleted once from H, thus we have a total of $O(m)$ tree operations over all iterations. Since each link-cut tree operation takes amortized expected time $O(\log^2 n)$, we obtain the bound on the total running time. □

The correctness of this algorithm follows almost immediately using that the level graph L (and therefore Transform(L)) is an acyclic graph.

Chapter 13

The Cut-Matching Game: Expanders via Max Flow

In this chapter, we learn about a new algorithm to compute expanders that employs max flow as a subroutine.

13.1 Introduction

We start with a review of expanders where make a subtle change the notion of an expander in comparison to chapter 5 to ease the exposition.

Definitions. We let $G = (V, E)$ be an unweighted, connected graph in this chapter, and let $\boldsymbol{d}$ be the degree vector, $E(S, V \setminus S)$ denote the set of edges crossing the cut A, B

Given set $\emptyset \subset S \subset V$, then we define the **sparsity** $\psi(S)$ of S by

$$\psi(S) = \frac{|E(S, V \setminus S)|}{\min\{|S|, |V \setminus S|\}}$$

Note that **sparsity** $\psi(S)$ differs from **conductance** $\phi(S) = \frac{|E(S,V\setminus S)|}{\min\{\text{vol}(S),\text{vol}(V\setminus S)\}}$, as defined in 5, in the denominator. It is straight-forward to see that in a connected graph $\psi(S) \geq \phi(S)$ for all S.

Clearly, we again have $\psi(S) = \psi(V \setminus S)$. We define the *sparsity* of a graph G by $\psi(G) = \min_{\emptyset \subset S \subset V} \psi(S)$. For any $\psi \in (0, n]$, we say a graph G is a ψ-expander with regard to sparsity, if $\psi(G) \geq \psi$. When the context is clear, we simply say that G is a ψ-expander.

The Main Result. The main result of this chapter is the following theorem.

Theorem 13.1.1. *There is an algorithm* SparsityCertifyOrCut(G, ψ) *that given a graph* G *and a parameter* $0 < \psi \leq 1$ *either:*

- Certifies *that* G *is a* $\Omega(\psi/\log^2 n)$*-expander with regard to sparsity, or*
- *Presents a* cut S *such that* $\psi(S) \leq O(\psi)$*.*

The algorithm runs in time $O(\log^2 n) \cdot T_{max\_flow}(G) + \tilde{O}(m)$ *where* $T_{max\_flow}(G)$ *is the time it takes to solve a Max Flow problem on* G[1]*.*

The bounds above can further be extended to compute ϕ-expanders (with regard to conductance). Using current state-of-the-art Max Flow results, the above problem can be solved in $m^{1+o(1)}$ time (see [**?**]).

13.2 Embedding Graphs into Expanders

Let us start by exploring the first key idea behind the algorithm. We therefore need a definition of what it means to embed one graph into another.

Definition of Embedding. Given graphs H and G that are defined over the same vertex set, then we say that a function $\text{Embed}_{H \to G}$ is an *embedding* if it maps each edge $(u, v) \in H$ to a u-to-v path $P_{u,v} = \text{Embed}_{H \to G}(u, v)$ in G.

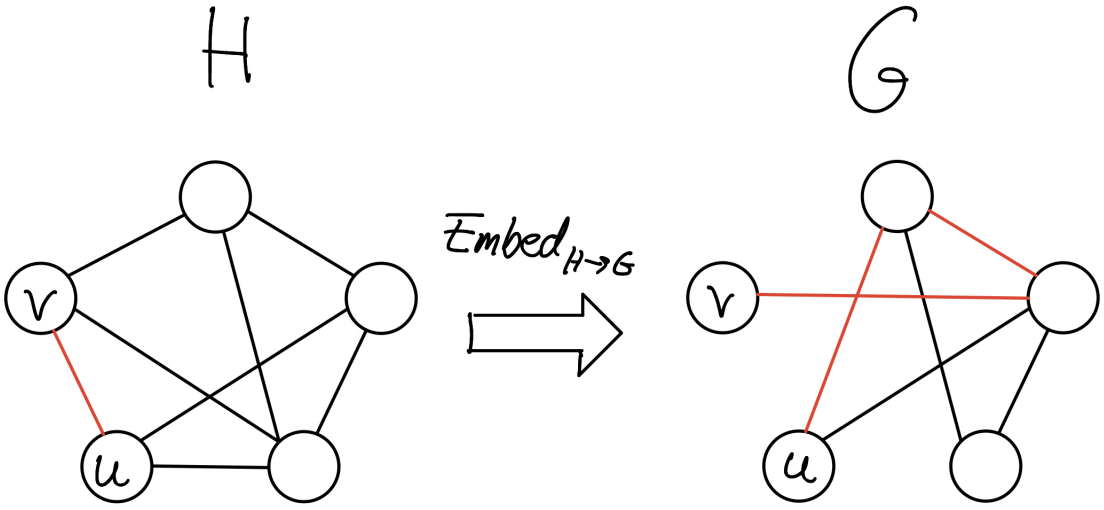


Figure 13.1: In this example the red edge (u, v) in H is mapped to the red u-to-v path in G.

[1]Technically, we will solve problems with two additional vertices and n additional edges but this will not change the run-time of any known max-flow algorithm asymptotically.

We say that the *congestion* of $\textsc{Embed}_{H\to G}$ is the maximum number of times that any edge $e \in E(G)$ appears on any embedding path:

$$cong(\textsc{Embed}_{H\to G}) = \max_{e\in E(G)} |\{e' \in E(H) \mid e \in \textsc{Embed}_{H\to G}(e')\}|.$$

Certifying Expander Graphs via Embeddings. Let us next prove the following lemma that is often consider Folklore.

Lemma 13.2.1. *Given a $\frac{1}{2}$-expander graph H and an embedding of H into G with congestion C, then G must be an $\Omega\left(\frac{1}{C}\right)$-expander.*

Proof. Consider any cut $(S, V \setminus S)$ with $|S| \leq |V \setminus S|$. Since H is a ψ-expander, we have that $|E_H(S, V \setminus S)| \geq |S|/2$. We also know by the embedding of H into G, that for each edge $(u, v) \in E_H(S, V \setminus S)$, we can find path a $P_{u,v}$ in G that also has to cross the cut $(S, V \setminus S)$ at least once. But since each edge in G is on at most C such paths, we can conclude that at least $|E_H(S, V \setminus S)|/C \geq |S|/2C$ edges in G cross the cut $(S, V \setminus S)$. □

Unfortunately, the reverse of the above lemma is not true, i.e. even if there exists no embedding from 1/2-expander H into G of congestion C, then G might still be an $\Omega\left(\frac{1}{C}\right)$-expander.

13.3 The Cut-Matching Algorithm

Although there are still some missing pieces, let us next discuss the algorithm 16. The algorithm runs for T iterations where we will later find that the right value to set T to is in $\Theta(\log^2 n)$.

Algorithm 16: $\textsc{SparsityCertifyOrCut}(G, \psi)$

for $i = 1, 2, \ldots, T$ **do**
  $(S_i, \overline{S}_i) \leftarrow \textsc{FindBiPartition}(G, \{M_1, M_2, \ldots, M_i\})$; `// Assume` $|S| = |\overline{S}| = n/2$
  Solve the flow problem on G where each vertex $e \in E$ receives capacity $\boldsymbol{c}(e) = 1/\psi$ and the demand at each edge $v \in S$ is $+1$ and for each $v \in \overline{S}$ is -1, by introducing a super-source s and super-sink t;
  if *flow procedure returns a flow* $\boldsymbol{f}$ *with* $\text{val}(f) = n/2$ **then**
    Remove s, t to derive S-$\overline{S}$ flow; then decompose flow $\boldsymbol{f}$ into flow paths $P_1, P_2, \ldots, P_{n/2}$;
    Create a matching M_i where for each s-to-$\overline{s}$ flow path P_i, we add $(s, \overline{s})$ to M_i.
  else
    Let $(X \cup \{s\}, (V \setminus X) \cup \{t\})$ be the st-minimum cut in the above flow problem.
    return $(X, V \setminus X)$
return $H = \bigcup_i M_i$

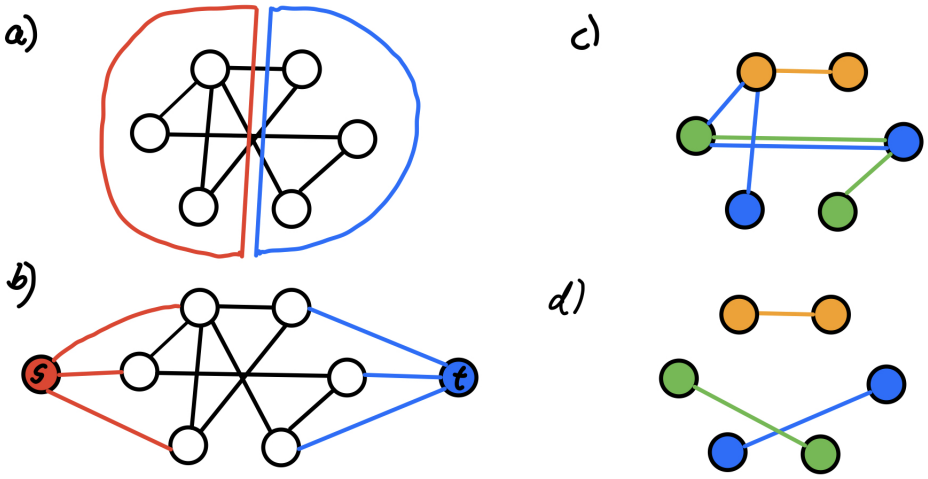


Figure 13.2: Illustration of the steps of the Algorithm. In a), a bi-partition of V is found. In b), the bi-partition is used to obtain a flow problem where we inject one unit of flow to each vertex in S via super-source s and extract one unit via super-sink t. c) A path flow decomposition. For each 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.

Beginning of an Iteration. In each iteration of the algorithm, we first invoke a sub-procedure FindBiPartition$(\cdot)$ that returns a cut $(S, \overline{S})$ (where $\overline{S} = V \setminus S$) that partitions the vertex set into two equal-sized sides. Here, we implicitly assume that the number of vertices n is even which is w.l.o.g.

Next, the algorithm creates a flow problem where each vertex $s \in S$ has to send exactly one unit of flow and each vertex $\overline{s} \in S$ has to receive exactly one unit of flow. We therefore introduce dummy nodes s and t, add for each vertex $v \in S$ an edge of capacity 1 between s and v; and for each vertex $v \in \overline{S}$ and edge between t and v of capacity 1. We set the capacity of edges in the original graph G to $1/\psi$ (which we assume wlog to be integer).

The If Statement. If the flow problem can be solved exactly, then we can find a path decomposition of the flow $\boldsymbol{f}$ in $\tilde{O}(m)$ time (for example using a DFS) where each path starts in S ends in $\overline{S}$ and carries one unit of flow[2]. This defines a one-to-one correspondence between the vertices in S and the vertices in $\overline{S}$. We capture this correspondences in the matching M_i. We will later prove the following lemma.

Lemma 13.3.1. *If the algorithm returns after constructing T matchings, for an appropriately chosen $T = \Theta(\log^2 n)$, then the graph H returned by the algorithm is a $\frac{1}{2}$-expander and H can be embedded into G with congestion $O(\log^2 n/\psi)$.*

[2]For simplicity, assume that the returned flow is integral.

The Else Statement. On the other hand, if the flow problem on G could not be solved, then we return the min-cut of the flow problem. Such a cut can be found in $O(m)$ time by using the above reduction to an s-t flow problem on which one can compute maximum flow $\boldsymbol{f}$ from which the s-t min-cut can be constructed by following the construction in the proof of Theorem 10.4.4.

It turns out that this min-cut already is a sparse cut by the way our flow problem is defined.

Lemma 13.3.2. *If the algorithm terminates with a cut $(X, V \setminus X)$, then $\psi(X) = O(\psi)$.*

Proof. Let $X_s = X \cup \{s\}, \overline{X}_s = (V \setminus X) \cup \{t\}$. First observe that since the flow was not routed, and $(X_s, \overline{X}_s)$ is an st-min cut for this flow problem, we have that $\boldsymbol{c}(E_G(X, \overline{X}_s) < n/2$ (otherwise we could have routed the demands).

Let n_s be the number of edges incident to the super-source s that cross the cut $(X_s, \overline{X}_s)$. Let n_t be the number of edges incident to t that cross the cut $(X_s, \overline{X}_s)$.

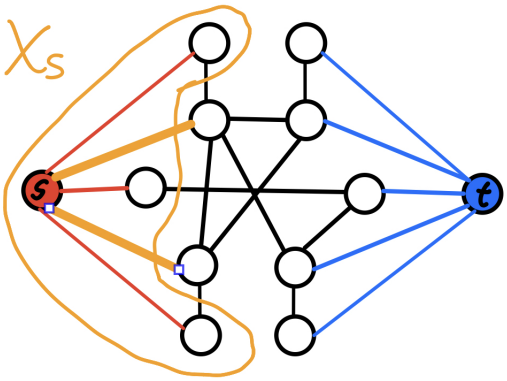


Figure 13.3: Set X_s is enclosed by the orange circle. The thick orange edges are in the cut and incident to super-source s. Thus they count towards n_s. Here $n_s = 2, n_t = 0$. Note that all remaining edges in the cut are black, i.e. were originally in G and therefore have capacity $1/\psi$.

Observe that after taking away the vertices s and t, the cut $(X, V \setminus X)$ has less than $n/2 - n_s - n_t$ capacity. But each remaining edge has capacity $1/\psi$, so the total number of edges in the cut can be at most $\psi \cdot (n/2 - n_s - n_t)$. Since $X = X_s \setminus \{s\}$ is of size at least $n/2 - n_s$, and $V \setminus X = \overline{X}_s \setminus \{t\}$ is of size at least $n/2 - n_t$, we have that the induced cut in G has

$$\psi(X) < \frac{\psi \cdot (n/2 - n_s - n_t)}{\min\{n/2 - n_s, n/2 - n_t\}} \leq \psi.$$

□

13.4 Constructing an Expander via Random Walks

Next, we give the implementation and analysis for the procedure FindBiPartition$(\cdot)$. We start however by giving some more preliminaries.

Random Walk on Matchings. Let $\{M_1, M_2, \ldots, M_T\}$ be the set of matchings we compute (if we never find a cut). In the i^{th}-step of the lazy random walk, we let the mass at each vertex j stay put with probability $1/2$, and otherwise traverses the edge in matching M_i incident to j with probability $1/2$.

We let $\boldsymbol{p}^t_{j\mapsto i}$ denote the probability that a particle that started at vertex j is at vertex i after a t-step lazy random walk. We let $\boldsymbol{p}^t_i = [\boldsymbol{p}^t_{1\mapsto i}\ \boldsymbol{p}^t_{2\mapsto i}\ \ldots\ \boldsymbol{p}^t_{n\mapsto i}]$. Note that for each edge $(i,j) \in M_{t+1}$, we have that

$$\boldsymbol{p}^{(t+1)}_i = \frac{1}{2}\boldsymbol{p}^t_i + \frac{1}{2}\boldsymbol{p}^t_j = \boldsymbol{p}^{(t+1)}_j.$$

We define the projection matrix $\boldsymbol{\Pi}^t = [\boldsymbol{p}^t_1, \boldsymbol{p}^t_2, \ldots, \boldsymbol{p}^t_n]^\top$ that maps an initial probability distribution $\boldsymbol{d}$ to the probability distribution over the vertices that the random walk visits them at step t. You will prove in the exercises that $\boldsymbol{\Pi}^t$ is *doubly-stochastic*.

We say that a lazy random walk is *mixing* at step t, if for each i, j, $\boldsymbol{p}^t_{j\mapsto i} \geq 1/(2n)$.

Lemma 13.4.1. *If t-step lazy random walk is* mixing, *then $H = \bigcup_{i\leq t} M_i$ is a $\frac{1}{2}$-expander.*

Proof. Consider any cut $(S, \overline{S})$ with $|S| \leq |\overline{S}|$. It is convenient to think about the random walks in terms of probability mass that is moved around. Observe that each vertex $j \in \overline{S}$ has to push at least $1/(2n)$ units of mass from j to i (by definition of mixing).

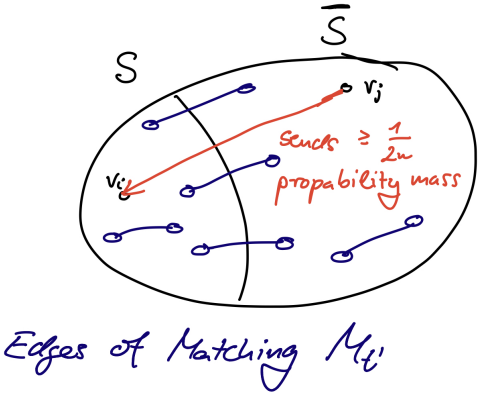


Figure 13.4: Each vertex $j \in \overline{S}$ sends at least $1/(2n)$ probability mass to i (red arrow). But in order to transport it, it does have to push the mass through edges in the matchings $M_1, M_2, \ldots, M_t$ that cross the cut.

Clearly, to move the mass from $\overline{S}$ to S it has to use the matching edges that also cross the cut.

Now observe that since there are $\geq n/2$ vertices in $\overline{S}$, and each of them has to push $\geq 1/(2n)$ mass to i, the total amount of probability mass pushed through the cut for i is $\geq 1/4$. Since there are $|S|$ such vertices i, the total amount of mass that has to cross the cut is $\geq |S|/4$. But note, that after each step of the random walk, the total probability mass at each vertex is exactly 1. Thus, at each step $i \leq t$, each edge in M_i crossing the cut can push at most $1/2$ units of probability mass over the cut (and thereafter the edge is gone).

It follows that there must be at least $|S|/2$ edges in the matchings $M_1, M_2, \ldots M_t$. But this implies that $H = \cup_i M_i$ is a $\frac{1}{2}$-expander. □

Implementing FindBiPartition$(\cdot)$. We can now give away the implementation of FindBiPartition$(\cdot)$ which you can find below.

Algorithm 17: FindBiPartition$(G, \{M_1, M_2, \ldots, M_t\})$

1 Choose random n-dimensional vector $\boldsymbol{r}$ orthogonal to $\mathbf{1}$.;
2 Compute vector $\boldsymbol{u} = \boldsymbol{\Pi}^t \boldsymbol{r}$, i.e. each $\boldsymbol{u}(i) = \boldsymbol{p}_i^t \cdot \boldsymbol{r}$;
3 Let S be the $n/2$ smallest vertices w.r.t. $\boldsymbol{u}$; and $\overline{S}$ be the $n/2$ largest w.r.t $\boldsymbol{u}$ (ties broken arbitrarily but consistently).;
4 **return** $(S, \overline{S})$

The central claim, we want to prove is the following: given a potential function for the random walk at step t

$$\Phi^t = \sum_{i,j} (\boldsymbol{p}_{j \mapsto i}^t - 1/n)^2 = \sum_i \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2.$$

Claim 13.4.2. *In the algorithm* SparsityCertifyOrCut$(\cdot)$*, we have* $\mathbb{E}[\Phi^t - \Phi^{(t+1)}] = \Omega(\Phi^t / \log n) - O(1/n^{-5})$*. Further, we have that* $\Phi^t - \Phi^{(t+1)}$ *is always non-negative. The expectation is over the random vector* $\boldsymbol{r}$ *chosen in the current round.*

Corollary 13.4.3. *For appropriate* $T = \Theta(\log^2 n)$*, the algorithm* SparsityCertifyOrCut$(\cdot)$ *has* $\Phi^{(T+1)} \leq 4/n^2$ *w.h.p. (i.e. with probability* $\geq 1 - 1/n$*).*

To obtain the Corollary, one can simply set up a sequence of random 0-1 variables $X^1, X^2, \ldots, X^{(T)}$ where each $X^{(t+1)}$ is 1 if and only if the decrease in potential is at least an $\Omega(1/\log n)$-fraction of the previous potential. Since the expectation is only over the current $\boldsymbol{r}$ in each round and we choose these independently at random, one can then use a Chernoff bound to argue that after T rounds (for an appropriate hidden constant), one has at least $\Omega(T)$ rounds during which the potential is decreased substantially (unless it is already tiny and the $O(n^{-5})$ factor dominates).

We further observe that this implies that $\{M_1, M_2, \ldots, M_T\}$ is mixing (otherwise, the contribution of the violating coordinate $\boldsymbol{p}_{j \mapsto i}^T$ would contribute $> 1/(2n)^2$ to the potential, but all terms are non-negative), and thereby we conclude the proof of our main theorem.

Let us now give the prove of Claim 13.4.2:

Interpreting the Potential Drop. Let us start by writing out the amount by which the potential decreases

$$\Phi^t - \Phi^{(t+1)} = \sum_i \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2 - \sum_i \|\boldsymbol{p}_i^{(t+1)} - \mathbf{1}/n\|_2^2$$

Considering now matching M_{t+1}, and an edge $(i,j) \in M_{t+1}$. We can re-write the former sum as $\sum_i \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2 = \sum_{(i,j)\in M_{t+1}} \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2 + \|\boldsymbol{p}_j^t - \mathbf{1}/n\|_2^2$ as each vertex occurs as exactly one endpoint of a matching edge. We can do the same for the $t+1$-step walk probabilities. Further, recall that for $(i,j) \in M_{t+1}$, we have $\boldsymbol{p}_i^{(t+1)} = \boldsymbol{p}_j^{(t+1)} = \frac{\boldsymbol{p}_i^t + \boldsymbol{p}_j^t}{2}$. Thus,

$$\begin{aligned}\Phi^t - \Phi^{(t+1)} &= \sum_{(i,j)\in M_{t+1}} \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2 + \|\boldsymbol{p}_j^t - \mathbf{1}/n\|_2^2 - \|\boldsymbol{p}_i^{(t+1)} - \mathbf{1}/n\|_2^2 - \|\boldsymbol{p}_j^{(t+1)} - \mathbf{1}/n\|_2^2 \\ &= \sum_{(i,j)\in M_{t+1}} \|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2 + \|\boldsymbol{p}_j^t - \mathbf{1}/n\|_2^2 - 2\left\|\frac{\boldsymbol{p}_i^t + \boldsymbol{p}_j^t}{2} - \mathbf{1}/n\right\|_2^2.\end{aligned}$$

Finally, we can use the formula $\|\boldsymbol{x}\|_2^2 + \|\boldsymbol{y}\|_2^2 - 2\|(\boldsymbol{x}+\boldsymbol{y})/2\|_2^2 = \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2$ term-wise to derive

$$\Phi^t - \Phi^{(t+1)} = \frac{1}{2}\sum_{(i,j)\in M_{t+1}} \|(\boldsymbol{p}_i^t - \mathbf{1}/n) - (\boldsymbol{p}_j^t - \mathbf{1}/n)\|_2^2 = \frac{1}{2}\sum_{(i,j)\in M_{t+1}} \|\boldsymbol{p}_i^t - \boldsymbol{p}_j^t\|_2^2.$$

The potential thus drops by a lot if vertices i and j are matched where $\boldsymbol{p}_i^t$ and $\boldsymbol{p}_j^t$ differ starkly. Note that this equality implies directly the remark in our claim that $\Phi^t - \Phi^{(t+1)}$ is non-negative.

Understanding the Random Projection. Next, we want to further lower bound the potential drop using the random vector $\boldsymbol{u}$. This intuitively helps a lot in our analysis since we are matching vertices i, j with high value $\boldsymbol{u}(i)$ and low value $\boldsymbol{u}(j)$ (or vice versa). We will show that (w.p. $\geq 1 - n^{-3}$)

$$\Phi^t - \Phi^{(t+1)} = \frac{1}{2}\sum_{(i,j)\in M_{t+1}} \|\boldsymbol{p}_i^t - \boldsymbol{p}_j^t\|_2^2 \geq \frac{n-1}{64\cdot\log n}\sum_{(i,j)\in M_{t+1}} |\boldsymbol{u}(i) - \boldsymbol{u}(j)|^2. \tag{13.1}$$

We prove this claim again term-wise, showing that for each pair of vertices $i, j \in V$, we have $\|\boldsymbol{p}_i^t - \boldsymbol{p}_j^t\|_2^2 \geq \frac{n-1}{16\cdot\log n}|\boldsymbol{u}(i) - \boldsymbol{u}(j)|^2$ w.h.p. It will then suffice to talk a union bound over all pairs i, j.

To this end, let us make the following observations: since $\boldsymbol{u}(i) = \boldsymbol{p}_i^t \cdot \boldsymbol{r}$, we have that $\boldsymbol{u}(i) - \boldsymbol{u}(j) = (\boldsymbol{p}_i^t - \boldsymbol{p}_j^t)\cdot\boldsymbol{r}$ by linearity. Also note that since $\sum_j \boldsymbol{p}_{j\mapsto i}^t = 1$ for all i (since $\boldsymbol{\Pi}^t$ is doubly-stochastic), we further have that the projection $(\boldsymbol{p}_i^t - \boldsymbol{p}_j^t)$ is orthogonal to $\mathbf{1}$.

We can now use the following statement about random vector $\boldsymbol{r}$ to argue about the effect of projecting $(\boldsymbol{p}_i^t - \boldsymbol{p}_j^t)$ onto $\boldsymbol{r}$. Below, we note that we have $d = n-1$ since $\boldsymbol{r}$ is chosen from the $(n-1)$-dimensional space orthogonal to $\mathbf{1}$.

Theorem 13.4.4 (see for example [?] or any lecture notes on Johnson-Lindenstrauss)**.** *If $\boldsymbol{y}$ is a vector of length ℓ in $\mathbb{R}^d$, and $\boldsymbol{r}$ a unit random vector in $\mathbb{R}^d$, then*

- $\mathbb{E}[(\boldsymbol{y}^\top \boldsymbol{r})^2] = \frac{\ell^2}{d}$*, and*
- *for $x \leq d/16$, then $\mathbb{P}[(\boldsymbol{y}^\top \boldsymbol{r})^2 \geq x\ell^2/d] \leq e^{-x/4}$*

This allows us to pick $x = 32 \cdot \log n$, and we then we obtain that

$$\mathbb{P}\left[((\boldsymbol{p}_i^t - \boldsymbol{p}_j^t) \cdot \boldsymbol{r})^2 \geq \frac{32 \log n}{n-1} \|\boldsymbol{p}_i^t - \boldsymbol{p}_j^t\|_2^2\right] \leq e^{-4 \log n} = n^{-8}. \tag{13.2}$$

Multiplying both sides of the event by $(n-1)/(64 \log n)$, we derive the claimed Inequality (13.1). We can further union bound over the $n/2$ matching pairs to all satisfy this bound with probability $\geq 1 - n^{-7}$, as desired.

Relating to the Lengths of the Projections. Let $\mu = \max_{i \in S} \boldsymbol{u}(i)$, then we have by definition that $\boldsymbol{u}(i) \leq \mu \leq \boldsymbol{u}(j)$ for all $i \in S, j \in \overline{S}$.

Now we can write

$$\begin{aligned}
\Phi^t - \Phi^{(t+1)} &\geq \frac{n-1}{64 \cdot \log n} \sum_{(i,j) \in M_{t+1}} |\boldsymbol{u}(i) - \boldsymbol{u}(j)|^2 \\
&\geq \frac{n-1}{64 \cdot \log n} \sum_{(i,j) \in M_{t+1}} (\boldsymbol{u}(i) - \mu)^2 + (\boldsymbol{u}(j) - \mu)^2 \\
&= \frac{n-1}{64 \cdot \log n} \sum_{i \in V} (\boldsymbol{u}(i) - \mu)^2 \\
&= \frac{n-1}{64 \cdot \log n} \left(\sum_{i \in V} \boldsymbol{u}(i)^2 - 2\mu \cdot \sum_{i \in V} \boldsymbol{u}(i) + n\mu^2 \right)
\end{aligned}$$

by standard calculations. We then observe that $\sum_i \boldsymbol{u}(i) = \sum_i \boldsymbol{p}_i^t \cdot \boldsymbol{r} = \mathbf{1} \cdot \boldsymbol{r} = 0$ by the fact that $\boldsymbol{\Pi}^t$ is doubly-stochastic and since $\boldsymbol{r}$ is orthogonal to the all-ones vector. We can therefore conclude

$$\frac{n-1}{64 \cdot \log n} \left(\sum_{i \in V} \boldsymbol{u}(i)^2 - 2\mu \cdot \sum_i \boldsymbol{u}(i) + n\mu^2 \right) \geq \frac{n-1}{64 \cdot \log n} \sum_{i \in V} \boldsymbol{u}(i)^2. \tag{13.3}$$

Taking the Expectation. Then, from the second fact of Theorem 13.4.4, we obtain that

$$\mathbb{E}\left[\sum_{i \in V} \boldsymbol{u}(i)^2\right] = \sum_{i \in V} \mathbb{E}[\boldsymbol{u}(i)^2] = \sum_{i \in V} \mathbb{E}[(\boldsymbol{p}_i^t \cdot \boldsymbol{r})^2] = \sum_{i \in V} \mathbb{E}[(\boldsymbol{p}_i^t - \mathbf{1}/n) \cdot \boldsymbol{r})^2] \tag{13.4}$$

$$= \sum_{i \in V} \frac{\|\boldsymbol{p}_i^t - \mathbf{1}/n\|_2^2}{n-1} = \frac{\Phi^t}{n-1} \tag{13.5}$$

where we used again that $\boldsymbol{r}$ is orthogonal to $\mathbf{1}$.

Unfortunately, we cannot directly use this expectation since we already conditioned on the high probability events in Equation (13.2). But a simple trick allows us to recover: Let $\mathcal{E}$ denote the union of all of these events. We have by the law of total expectation

$$\mathbb{E}\left[\sum_{i\in V}\boldsymbol{u}(i)^2\right] = \mathbb{P}[\neg\mathcal{E}]\cdot\mathbb{E}\left[\sum_{i\in V}\boldsymbol{u}(i)^2|\neg\mathcal{E}\right] + \mathbb{P}[\mathcal{E}]\cdot\mathbb{E}\left[\sum_{i\in V}\boldsymbol{u}(i)^2|\mathcal{E}\right]$$

But note that $\mathbb{E}\left[\sum_{i\in V}\boldsymbol{u}(i)^2|\mathcal{E}\right]$ has to be smaller than n because $\sum_{i\in V}\boldsymbol{u}(i)^2 = \sum_{i\in V}(\boldsymbol{p}_i^t\cdot\boldsymbol{r})^2 \leq n$ with probability 1 (because each $\boldsymbol{p}_i^t$ is a unit vector). Recall that we calculated $\mathbb{P}[\mathcal{E}]\leq n^{-7}$. Thus, we can conclude that $\mathbb{E}\left[\sum_{i\in V}\boldsymbol{u}(i)^2|\neg\mathcal{E}\right]\geq\frac{\Phi^t}{n-1}-n^{-6}$.

It remains to combine our insights to conclude

$$\mathbb{E}[\Phi^t-\Phi^{(t+1)}|\neg\mathcal{E}]\geq\frac{n-1}{64\cdot\log n}\left(\frac{\Phi^t}{n-1}-n^{-6}\right)=\Omega(\Phi^t/\log n)-O(1/n^{-5}).$$

Since again the event $\mathcal{E}$ occurs with very low probability, and since $\Phi^t-\Phi^{t+1}$ is non-negative always, we can then conclude that in unconditionally, in expectation, the potential decreases by $\Omega(\Phi^t/\log n)-O(1/n^{-5})$.

Chapter 14

Distance Oracles

In this chapter, we learn about distance oracles as presented in the seminal paper [TZ05]. Distance Oracles are data structures that allow for any undirected graph $G = (V, E)$ to be stored compactly in a format that allows to query for the (approximate) distance between any two vertices u, v in the graph. The main result of this chapter is the following data structure.

Theorem 14.0.1. *There is an algorithm that, for any integer $k \geq 1$ and undirected graph $G = (V, E)$, computes a data structure that can be stored using $\tilde{O}(kn^{1+1/k})$ bits such that on querying any two vertices $u, v \in V$ returns in $O(k)$ time a distance estimate $\widetilde{\mathbf{dist}}(u, v)$ such that*

$$\mathbf{dist}(u, v) \leq \widetilde{\mathbf{dist}}(u, v) \leq (2k - 1) \cdot \mathbf{dist}(u, v).$$

The algorithm computes the data structure in expected time $\tilde{O}(kmn^{1/k})$.

Remark 14.0.2. Note that for $k = 1$, the theorem above is trivial: it can be solved by computing APSP and storing the distance matrix of G.

Remark 14.0.3. We point out that given space $O(n^{1+1/k})$, approximation $(2k-1)$ is the best that we can hope for according to a popular and widely believed conjecture that essentially says that there are unweighted graphs that have no cycle of length $(2k+1)$ but have $\tilde{\Omega}(n^{1+1/k})$ edges. A more careful analysis than we will carry out allows to shave all logarithmic factors from Theorem 14.0.1 and therefore the data structure is only a factor k off in space from optimal while also answering queries *extremely* efficiently. It turns out that the factor k can also be removed in space and query time (although currently preprocessing is quite expensive), see therefore the following (really involved) articles [Che14, Che15].

Remark 14.0.4. Also note that in directed graphs no such distance oracle is possible. Even maintaining the transitive closure (the information of who reaches who) can only be preserved if one stores $\tilde{\Omega}(n^2)$ bits.

14.1 Warm-up: A Distance Oracle for $k = 2$

Let us first describe the data structure for the case where $k = 2$. See therefore the pseudo-code below. Here we use the convention that $\mathbf{dist}(x, X)$ for some vertex $x \in V$ and some subset $X \subseteq V$ is the minimum distance from x to any $y \in X$, formally $\mathbf{dist}(x, X) = \min_{y \in X} \mathbf{dist}(x, y)$.

Algorithm 18: PREPROCESS(G)

```
1  Obtain S by sampling every vertex v ∈ V i.i.d. with probability n^{-1/2};
2  foreach s ∈ S do
3  |   Compute all distances from s to any other vertex v ∈ V;
4  |   In a hash table H_s store for each v ∈ V an entry with key v and value dist_G(s, v).
5  end
6  foreach u ∈ V \ S do
7  |   Find the pivot p(u) of u to be some vertex in S that minimizes the distance to u;
8  |   Store p(u) along with dist_G(u, p(u)) = dist_G(u, S);
9  |   Find the bunch B(u) = {v ∈ V | dist_G(u, v) < dist_G(u, S)};
10 |   In a hash table H_u store for each v ∈ B(v) an entry with key v and value dist_G(u, v).
11 end
```

The key to the algorithm is the definition of *pivots* and *bunches*. Below is an example that illustrates their definitions.

Without further due, let us discuss the query procedure which is depicted below. It essentially consists of checking whether the vertex v is already in the bunch of u in which case we have stored the distance $\mathbf{dist}_G(u, v)$ explicitly. Otherwise, it uses a detour via its pivot.

Algorithm 19: QUERY(u,v)

```
1  if v ∈ H_u then return value dist_G(u, v) ;
2  return dist_G(u, p(u)) + dist_G(p(u), v) (the latter from H_{p(u)})
```

The second case is illustrated below.

Approximation Analysis. It is straight-forward to see that if we return in line 1 of the query algorithm, then we return the exact distance.

If we return in the second line, then we know that $v \notin B(u)$. By definition of a bunch this implies that

$$\mathbf{dist}(u, v) \geq \mathbf{dist}(u, S) = \mathbf{dist}(u, p(u)).$$

We can further use the triangle inequality to conclude that

$$\mathbf{dist}_G(p(u), v) \leq \mathbf{dist}(u, p(u)) + \mathbf{dist}(u, v)$$

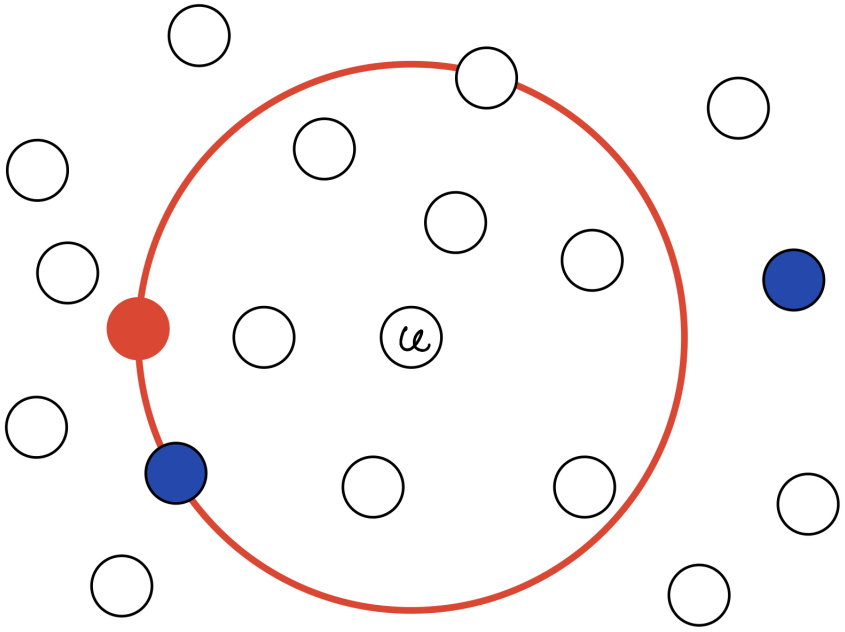


Figure 14.1: Graph G (without the edges) where vertices are drawn according to their distance from u. The blue and red vertices are in S. The red vertex is chosen to be the pivot $p(u)$. Note that another vertex in S could have been chosen to become the pivot. The bunch of u is the vertices that are strictly withing the red circle. In particular, both blue vertices, the red vertex and also the white vertex on the boundary of the red circle are *not* in the bunch $B(u)$.

Combining these two inequalities, we obtain

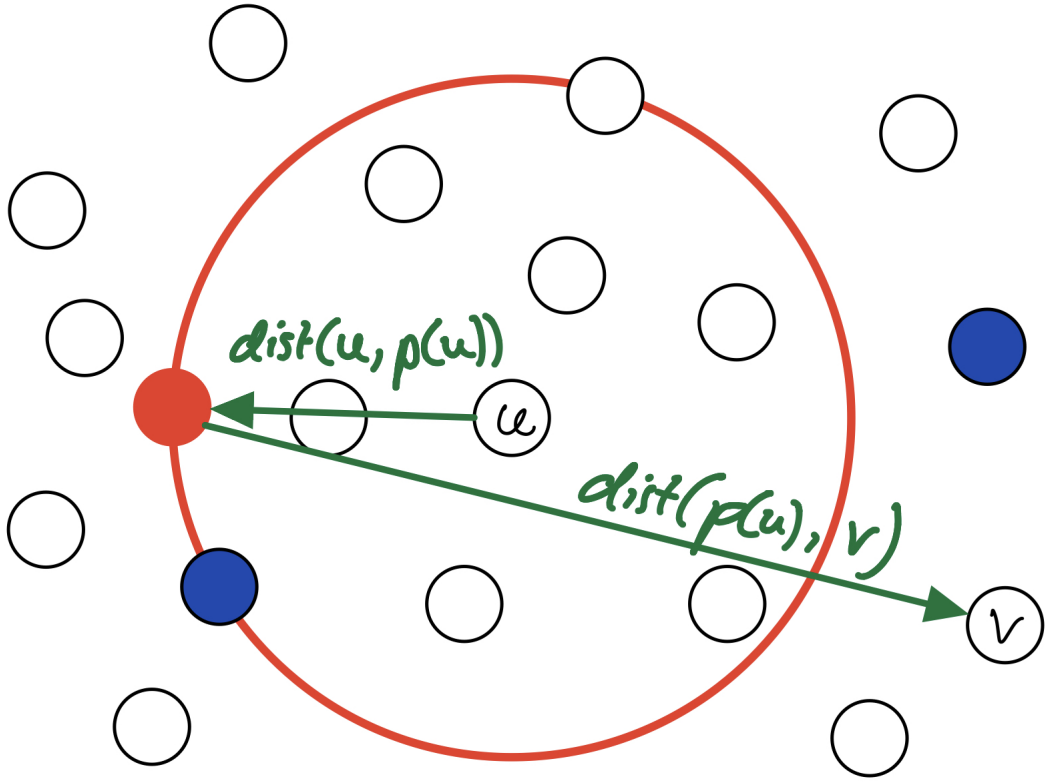
$$\mathbf{dist}_G(u, p(u)) + \mathbf{dist}_G(p(u), v) \leq 2 \cdot \mathbf{dist}(u, p(u)) + \mathbf{dist}(u, v) \leq 3 \cdot \mathbf{dist}(u, v).$$

We conclude that we obtain a 3-approximation.

Space Analysis. For each vertex $s \in S$, we store a hash-table with one entry for each vertex v. This can be stored with space $O(|S|n)$. We have by a standard Chernoff-bound that $|S| = \tilde{O}(\sqrt{n})$ w.h.p. so this becomes $\tilde{O}(n^{3/2})$.

Next, fix some $u \in V \setminus S$, and let us argue about the size of $B(u)$ (which asymptotically matches $|\mathcal{H}_u|$). We order the vertices $v_1, v_2, \ldots, v_n$ in V by their distance from u. Since we sample uniformly at random, by a simple Chernoff bound, we obtain that the first vertex v_i in $v_1, v_2, \ldots, v_n$ that is in S has $i = \tilde{O}(\sqrt{n})$ w.h.p.. But note that since only vertices that are *strictly* closer to u than v_i are in $B(u)$, this implies that $|B(u)| = \tilde{O}(\sqrt{n})$.

It remains to take a careful union bound over all bad events at every vertex $u \in V \setminus S$ to conclude that with high probability, the hash tables $\mathcal{H}_u$ for all $u \in V \setminus S$ combined take total space $\tilde{O}(|V \setminus S|\sqrt{n}) = \tilde{O}(n^{3/2})$. For the rest of the section, we condition on the event that each $|B(v)| = \tilde{O}(\sqrt{n})$ to use it as if it was a deterministic guarantee.



Preprocessing and Query Time. In order to find the distances stored in the hash tables $\mathcal{H}_s$ for $s \in S$, we can simply run Dijkstra from each $s \in S$ on the graph in total time $\tilde{O}(m|S|) = \tilde{O}(mn^{1/2})$.

To compute the *pivots* for each vertex u, we can insert a super-vertex s' and add an edge from s' to each $s \in S$ of weight 0. We can then run Dijkstra from s' on $G \cup \{s'\}$. Note that for each u, we have $\mathbf{dist}_{G\cup\{s'\}}(s', u) = \mathbf{dist}_G(p(u), u)$ and that $p(u)$ can be chosen to be the closest vertex on the path from s' to u that Dijkstra outputs along with the distances (recall that Dijkstra can output a shortest path tree). This takes $\tilde{O}(m)$ time.

It remains to compute the bunches $B(u)$. Here, we use duality: we define the cluster $C(w)$ for every vertex $w \in V \setminus S$ to be the set

$$C(w) = \{v \in V \mid \mathbf{dist}_G(v, w) < \mathbf{dist}_G(v, p(v))\}.$$

Note the subtle difference to the bunches in that membership of v now depends on $p(v)$ and *not* on $p(u)$! It is not hard to see that $u \in C(w) \iff w \in B(u)$. And it is straight-forward to compute the bunches from the clusters in time $O(\sum_v |B(v)|) = O(\sum_w |C(w)|) = \tilde{O}(n^{3/2})$.

Finally, it turns out that we can compute each $C(w)$ by running Dijkstra with a small modification.

Lemma 14.1.1 (Lemma 4.2 in [TZ05])**.** *Consider running Dijkstra from a vertex w but only relaxing edges incident to vertices v that satisfy $\mathbf{dist}_G(v, w) < \mathbf{dist}_G(v, p(v))$. Then,*

the algorithm computes $C(w)$ and all distances $\mathbf{dist}(v,w)$ *for* $v \in C(w)$ *in time* $\tilde{O}(|E(C(w))|)$ *where* $E(C(w))$ *are the edges that touch a vertex in* $C(w)$*.*

It remains to observe that the total time required to compute all clusters is

$$\tilde{O}(\sum_w |E(C(w))|) = \tilde{O}(\sum_{w,v \in C(w)} |E(v)|) = \tilde{O}(\sum_{v,w \in B(v)} |E(v)|) = \tilde{O}(\sum_v |E(v)||B(v)|).$$

But we have upper bounded $|B(v)| = \tilde{O}(\sqrt{n})$ for all v, thus each vertex just pays its degree $\tilde{O}(\sqrt{n})$ times and we get running time $\tilde{O}(m\sqrt{n})$.

Monte Carlo vs. Las Vegas. Note that the analysis above only guarantees that the algorithm works well with high probability. However, it is not hard to see that the algorithm can be transformed into a Las Vegas algorithm: whenever we find a bunch $B(v)$ whose size exceeds our $\tilde{O}(\sqrt{n})$ bound, we simply re-run the algorithm. This guarantees that the final data structure that we output indeed satisfies the guarantees stipulated in the theorem.

14.2 Distance Oracles for any $k \geq 2$

The generalization for all k's is rather straight-forward except for the query operation which works a bit magically. Let's first define the data structure by giving our new pre-processing algorithm.

Algorithm 20: PREPROCESS(G)

```
S_1 = V; S_{k+1} = ∅;
foreach i ∈ [1,k] do  Obtain S_{i+1} by sampling every v ∈ S_i i.i.d. with prob. n^{-1/k} ;
foreach u ∈ V do
    foreach i ∈ [1,k] do
        Let p_i(u) be some vertex in S_i that minimizes the distance to u;
        Store p_i(u) along with dist_G(u, p_i(u)).
    end
    Let bunch B(u) = ⋃_i B_i(u) where B_i(u) = {v ∈ S_i | dist_G(u,v) < dist_G(u, S_{i+1})};
    In a hash table H_u store for each v ∈ B(u) an entry with key v and value
     dist_G(u,v).
end
```

Note that in a sense this algorithm is almost easier than the one for $k = 2$ since it treats each level in the same fashion. Here we ensure that the last set S_{k+1} is empty (which would happen with constant probability otherwise).

We make the implicit assumption throughout that $S_k \neq \emptyset$ so that $p_k(u)$ is well-defined for each u. We also define $\mathbf{dist}(x,X) = \infty$ if X is the empty set.

The drawing below illustrates the new definition of a bunch where we have chosen $k = 3$ to keep things simple.

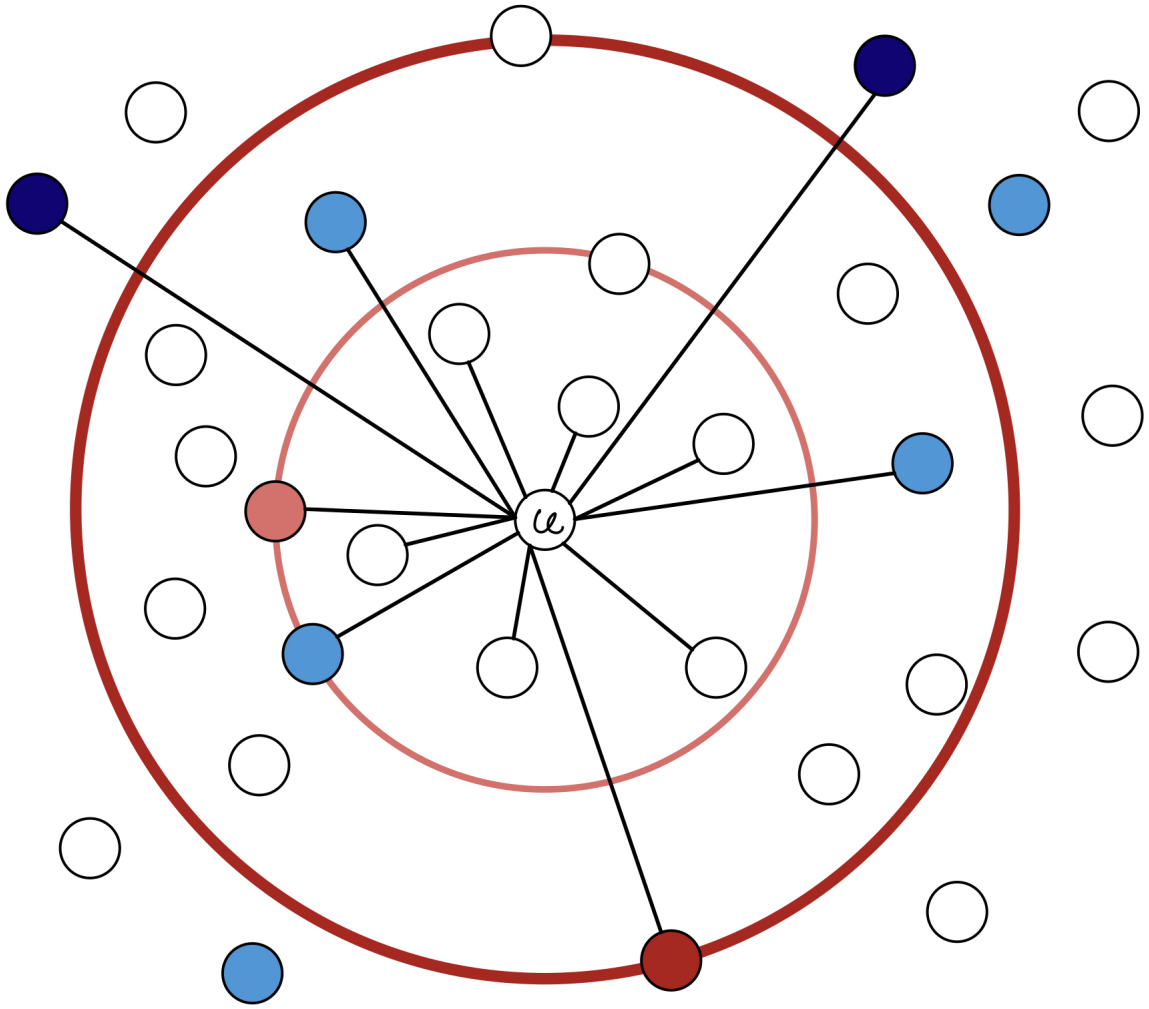


Figure 14.2: Graph G (without the edges) where vertices are drawn according to their distance from u. All vertices are in S_1. The blue and red vertices are in S_2. The dark blue and dark red vertices are also in S_3. Finally, we have $S_4 = \emptyset$. The light red vertex is chosen as pivot $p_2(u)$; the dark red vertex is chosen as pivot $p_3(u)$. The bunch $B(u)$ includes all vertices that have a black edge in this graph to u; in particular these are the white vertices within the circle drawn in light red ($B_1(u)$); the light blue vertices encircled by the dark red circle ($B_2(u)$); and all dark blue vertices ($B_3(u)$).

Before we explain the query procedure, let us give a (rather informal) analysis of the space required by our new data structure.

Space Analysis. We have for each vertex $u \in V$, and each $1 \leq i \leq k$ that the bunch $B_i(u)$ consists of all vertices in S_i that are closer to u than the closest vertex in S_{i+1}.

Now, order the vertices $x_1, x_2, \ldots$ in S_i by their distance from u. Since each vertex in S_i is sampled into S_{i+1} with probability $n^{-1/k}$, we have that with high probability some vertex x_i with $i = O(n^{1/k} \log n)$ is sampled into S_{i+1}. This ensures that $|B_i(u)| = \tilde{O}(n^{1/k})$ with high probability.

Applying this argument for all i, we have that $|B(u)| = \tilde{O}(k \cdot n^{1/k})$ for each u w.h.p. and therefore our space bound follows.

Preprocessing Time. Much like in the $k = 2$ construction, we can define for each $u \in S_i \setminus S_{i+1}$ the cluster $C(u) = \{v \in V \mid \mathbf{dist}_G(u, v) < \mathbf{dist}_G(v, S_{i+1})\}$. Extending our analysis from before using this new definition, we get construction time $\tilde{O}(kmn^{1/k})$.

Query Operation. A straight-forward way to query our new data structure for a tuple (u, v) would be to search for the smallest i such that $v \in B(p_i(u))$ and then return $\mathbf{dist}_G(u, p_i(u)) + \mathbf{dist}_G(p_i(u), v)$. This can be analyzed in the same way as we did for $k = 2$ to obtain stretch $4k - 3$[1].

However, we aim for stretch approximation $2k - 1$. We state below the pseudo-code to achieve this guarantee.

Algorithm 21: QUERY(u,v)

```
1 w ← u; i ← 1;
2 while w ∉ B(v) do
3 |   i ← i + 1;
4 |   (u, v) ← (v, u);
5 |   w ← p_i(u)
6 end
7 return dist_G(u, w) + dist_G(w, v)
```

Our main tool in the analysis is the claim below where we define $\Delta = \mathbf{dist}_G(u, v)$.

Claim 14.2.1. *After the i^{th} iteration of the while-loop, we have $\mathbf{dist}_G(u, w) \leq i\Delta$.*

This implies our theorem, since we have at most $k - 1$ iterations, then w is a vertex in S_k and $S_k \subseteq B(x)$ for all vertices $x \in V$. Therefore we have that for the final w, we have $\mathbf{dist}_G(u, w) \leq (k - 1)\Delta$. It remains to conclude by the triangle inequality that

$$\mathbf{dist}_G(u, w) + \mathbf{dist}_G(w, v) \leq 2\mathbf{dist}_G(u, w) + \Delta \leq (2k - 1)\mathbf{dist}_G(u, v).$$

Proof of Claim 14.2.1. Let w_i, u_i, v_i denote the variables w, u, v after the i^{th} while-loop iteration (or right before for w_0, u_0, v_0).

[1] One can actually prove that this strategy gives an $4k - 5$ stretch with a little trick.

For $i = 0$, we have that $w_0 = u_0$; thus, $\mathbf{dist}_G(u_0, w_0) = 0$.

For $i \geq 1$, we want to prove that if the i^{th} while-loop iteration is executed then $\mathbf{dist}_G(u_i, w_i) \leq \mathbf{dist}_G(u_{i-1}, w_{i-1}) + \Delta$ (if it is not executed then the statement follows trivially).

In order to prove this, observe that by the while-loop condition, we must have had $w_{i-1} \notin B(v_{i-1})$, thus $\mathbf{dist}_G(v_{i-1}, w_{i-1}) \geq \mathbf{dist}_G(v_{i-1}, p_i(v_{i-1}))$.

But the while-iteration sets $u_i = v_{i-1}$ and $w_i = p_i(v_{i-1})$, and therefore we have

$$\begin{aligned}
\mathbf{dist}_G(u_i, w_i) &= \mathbf{dist}_G(v_{i-1}, p_i(v_{i-1})) \\
&\leq \mathbf{dist}_G(v_{i-1}, w_{i-1}) \\
&\leq \mathbf{dist}_G(u_{i-1}, w_{i-1}) + \mathbf{dist}_G(v_{i-1}, u_{i-1}) \\
&= \mathbf{dist}_G(u_{i-1}, w_{i-1}) + \Delta.
\end{aligned}$$

□

Part IV

Further Topics in Convex Optimization

Chapter 15

Separating Hyperplanes, Lagrange Multipliers, KKT Conditions, and Convex Duality

15.1 Overview

First part of this chapter introduces the concept of a separating hyperplane of two sets followed by a proof that for two closed, convex and disjoint sets a separating hyperplane always exists. This is a variant of the more general *separating hyperplane theorem*[1] due to Minkowski. Then *Lagrange multipliers* $\boldsymbol{x}$, $\boldsymbol{s}$ of a convex optimization problem

$$\begin{aligned} \min_{y} \ & \mathcal{E}(\boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b} \\ & c(\boldsymbol{y}) \leq 0 \end{aligned}$$

are introduced and with that, the *Lagrangian*

$$\boldsymbol{L}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) = \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) + \boldsymbol{s}^\top c(\boldsymbol{y})$$

is defined. Finally, we deal with the dual problem

$$\max_{\boldsymbol{x},\boldsymbol{s},\boldsymbol{s}\geq 0} L(\boldsymbol{x}, \boldsymbol{s}),$$

where $L(\boldsymbol{x}, \boldsymbol{s}) = \min_y L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$. We show *weak duality*, i.e. $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \leq \mathcal{E}(\boldsymbol{y})$ and that assuming *Slater's condition* the values of both the primal and dual is equal, which is referred to as *strong duality*.

[1] Wikipedia is good on this: https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

15.2 Separating Hyperplane Theorem

Suppose we have two convex subsets $A, B \subseteq \mathbb{R}^n$ that are disjoint ($A \cup B = \emptyset$). We wish to show that there will always be a (hyper-)plane H that separates these two sets, i.e. A lies on one side, and B on the other side of H.

So what exactly do we mean by Hyperplane? Let's define it.

Definition 15.2.1 (Hyperplane)**.** A *hyperplane* H of dimension n is the subset $H := \{\boldsymbol{x} \in \mathbb{R}^n : \langle \boldsymbol{n}, \boldsymbol{x} \rangle = \mu\}$. We say H has *normal* $\boldsymbol{n} \in \mathbb{R}^n$ and *threshold* μ. It is required that $\boldsymbol{n} \neq \boldsymbol{0}$.

Every hyperplane divides $\mathbb{R}^n$ into two halfspaces $\{\boldsymbol{x} : \langle \boldsymbol{v}, \boldsymbol{x} \rangle \geq \mu\}$ and $\{\boldsymbol{x} : \langle \boldsymbol{v}, \boldsymbol{x} \rangle \leq \mu\}$. It separates two sets, if they lie in different halfspaces. We formally define separating hyperplane as follows.

Definition 15.2.2 (Separating Hyperplane)**.** We say a hyperplane H *separates* two sets A, B iff

$$\forall \boldsymbol{a} \in A : \langle \boldsymbol{n}, \boldsymbol{a} \rangle \geq \mu$$
$$\forall \boldsymbol{b} \in B : \langle \boldsymbol{n}, \boldsymbol{b} \rangle \leq \mu$$

If we replace $\geq$ with $>$ and $\leq$ with $<$ we say H *strictly* separates A and B.

It is easy to see that there exists disjoint non-convex sets that can not be separated by a hyperplane (e.g. a point cannot be separated from a ring around it). But can two disjoint convex sets always be strictly separated by a hyperplane? The answer is no: consider the two-dimensional case depicted in Figure 15.1 with $A = \{(x, y) : x \leq 0\}$ and $B = \{(x, y) : x > 0 \text{ and } y \geq \frac{1}{x}\}$. Clearly they are disjoint; however the only separating hyperplane is $H = \{(x, y) : x = 0\}$ but it intersects A.

One can prove that there exists a non-strictly separating hyperplane for any two disjoint convex sets. We will prove that if we further require A,B to be closed and bounded, then a strictly separating hyperplane always exists. (Note in the example above how our choice of B is not bounded.)

Theorem 15.2.3 (Separating Hyperplane Theorem; closed, bounded sets)**.** *For two closed, bounded, and disjoint convex sets $A, B \in \mathbb{R}^n$ there exists a strictly separating hyperplane H. One such hyperplane is given by normal $\boldsymbol{n} = \boldsymbol{d} - \boldsymbol{c}$ and threshold $\mu = \frac{1}{2}\left(\|\boldsymbol{d}\|_2^2 - \|\boldsymbol{c}\|_2^2\right)$, where $\boldsymbol{c} \in A$, $\boldsymbol{d} \in B$ are the minimizers of the distance between A and B*

$$dist(A, B) = \min_{\boldsymbol{a} \in A, \boldsymbol{b} \in B} \|\boldsymbol{a} - \boldsymbol{b}\|_2 > 0.$$

Proof. We omit the proof that $\text{dist}(A, B) = \min_{\boldsymbol{a} \in A, \boldsymbol{b} \in B} \|\boldsymbol{a} - \boldsymbol{b}\|_2 > 0$, which follows from A, B being disjoint, closed, and bounded. Now, we want to show that $\langle \boldsymbol{n}, \boldsymbol{b} \rangle > \mu$ for all $\boldsymbol{b} \in B$; then $\langle \boldsymbol{n}, \boldsymbol{a} \rangle < \mu$ for all $\boldsymbol{a} \in A$ follows by symmetry. Observe that

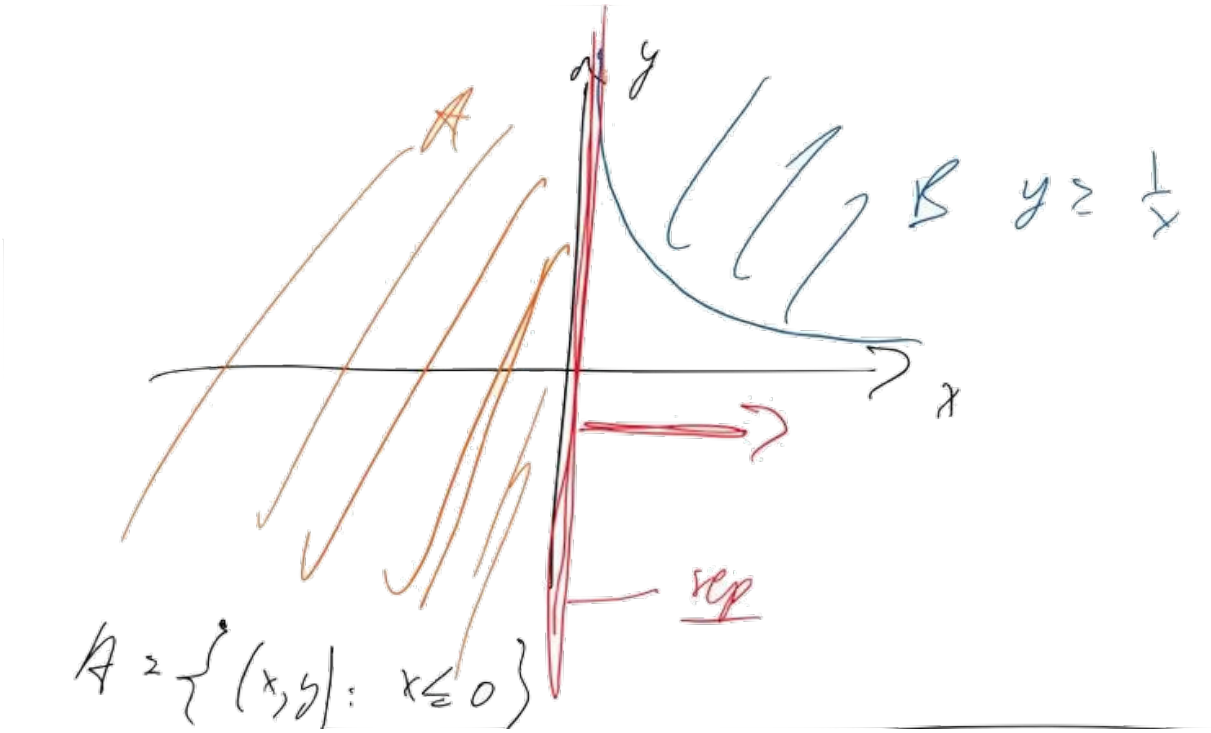


Figure 15.1: The sets $A = \{(x, y) : x \leq 0\}$ and $B = \{(x, y) : x > 0 \text{ and } y \geq \frac{1}{x}\}$ only permit a non-strictly separating hyperplane.

$$\begin{aligned}
\langle \boldsymbol{n}, \boldsymbol{d} \rangle - \mu &= \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{d} \rangle - \frac{1}{2} \left(\|\boldsymbol{d}\|_2^2 - \|\boldsymbol{c}\|_2^2 \right) \\
&= \|\boldsymbol{d}\|_2^2 - \boldsymbol{d}^\top \boldsymbol{c} - \frac{1}{2} \|\boldsymbol{d}\|_2^2 + \frac{1}{2} \|\boldsymbol{c}\|_2^2 \\
&= \frac{1}{2} \|\boldsymbol{d} - \boldsymbol{c}\|_2^2 > 0.
\end{aligned}$$

So suppose there exists $\boldsymbol{u} \in B$ such that $\langle \boldsymbol{n}, \boldsymbol{u} \rangle - \mu \leq 0$. We now look at the line defined by the distance minimizer $\boldsymbol{d}$ and the point on the "wrong side" $\boldsymbol{u}$. Define $\boldsymbol{b}(\lambda) = \boldsymbol{d} + \lambda(\boldsymbol{u} - \boldsymbol{d})$, and take the derivative of the distance between $\boldsymbol{b}(\lambda)$ and $\boldsymbol{c}$. Evaluated at $\lambda = 0$ (which is when $\boldsymbol{b}(\lambda) = \boldsymbol{d}$), this yields

$$\frac{d}{d\lambda} \|\boldsymbol{b}(\lambda) - \boldsymbol{c}\|_2^2 \Big|_{\lambda=0} = 2 \langle \boldsymbol{d} - \lambda \boldsymbol{d} + \lambda \boldsymbol{u} - \boldsymbol{c}, \boldsymbol{u} - \boldsymbol{d} \rangle|_{\lambda=0} = 2 \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{u} - \boldsymbol{d} \rangle .$$

However, this would imply that the gradient is strictly negative since

$$\begin{aligned}
\langle \boldsymbol{n}, \boldsymbol{u} \rangle - \mu &= \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{u} \rangle - \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{d} \rangle + \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{d} \rangle - \mu \\
&= \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{u} - \boldsymbol{d} \rangle + \|\boldsymbol{d}\|_2^2 - \langle \boldsymbol{c}, \boldsymbol{d} \rangle - \frac{1}{2} \|\boldsymbol{d}\|_2^2 + \frac{1}{2} \|\boldsymbol{c}\|_2^2 \\
&= \langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{u} - \boldsymbol{d} \rangle + \frac{1}{2} \|\boldsymbol{d} - \boldsymbol{c}\|_2^2 \leq 0.
\end{aligned}$$

This contradicts the minimality of $\boldsymbol{d}$ and thus concludes this proof. □

A more general separating hyperplane theorem holds even when the sets are not closed and bounded:

Theorem 15.2.4 (Separating Hyperplane Theorem)**.** *Given two disjoint convex sets* $A, B \in \mathbb{R}^n$ *there exists a hyperplane* H *separating them.*

15.3 Lagrange Multipliers and Duality of Convex Problems

In this Section, we'll learn about *Langrange Multipliers* and how they lead to convex duality. But first, let's see an example to help illustrate where these ideas come from.

Imagine you were to prove that for all $\boldsymbol{x} \in \mathbb{R}^n$ we have $\|\boldsymbol{x}\|_p \leq n^{\frac{1}{2}-\frac{1}{p}} \|\boldsymbol{x}\|_2$ for some $1 \leq p \leq 2$;. We can look at this as optimizing $\max_{\boldsymbol{x}} \|\boldsymbol{x}\|_p$ subject to $\|\boldsymbol{x}\|_2$ being constant, e.g. simply $\|\boldsymbol{x}\|_2 = 1$. Similarly, we could also compute $\max_{\|\boldsymbol{x}\|_2^2=1} \|\boldsymbol{x}\|_p^p$. Then the statement above follows from a scaling argument.

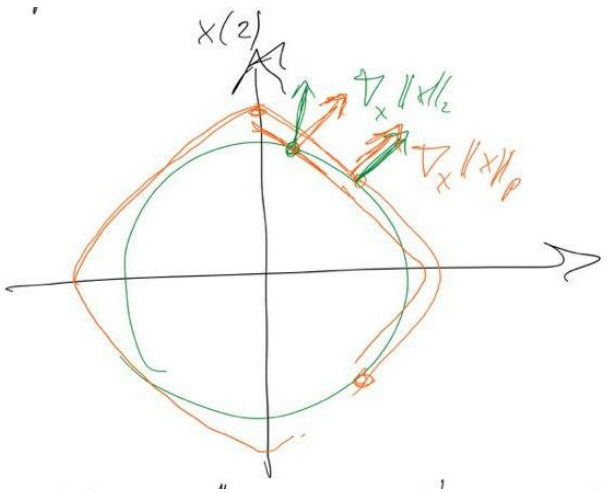


Figure 15.2: Looking at fixed $\|\boldsymbol{x}\|_p = \alpha$ and $\|\boldsymbol{x}\|_2 = 1$. (Here, $p = 1.5$.)

If we move from $\boldsymbol{x}$ to $\boldsymbol{x}+\boldsymbol{\delta}$ with $\boldsymbol{\delta} \perp \boldsymbol{\nabla}_{\boldsymbol{x}} \|\boldsymbol{x}\|_2^2$ and $\boldsymbol{\delta} \not\perp \boldsymbol{\nabla}_{\boldsymbol{x}} \|\boldsymbol{x}\|_p^p$ means that for infinitesimally small $\boldsymbol{\delta}$ the 2-norm stays constant but the p-norm changes. That means for either $\boldsymbol{x} - \boldsymbol{\delta}$ or $\boldsymbol{x} + \boldsymbol{\delta}$ the p-norm increases while the 2-norm stays constant. Hence at the maximum of $\|\boldsymbol{x}\|_p$ the gradients of both norms have to be parallel, i.e.

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \left(\|\boldsymbol{x}\|_p^p - \lambda \|\boldsymbol{x}\|_2^2 \right) = 0.$$

This insight is the core idea of Lagrange multipliers (in this case λ). Note that once we know this, we can conclude that there exists a maximizer where $\boldsymbol{x} = \alpha \mathbf{1}_S$, i.e. $\boldsymbol{x}$ is the indicator of some set S, because otherwise the gradients cannot be parallel. Finally, we can argue that among such vectors, the maximum gap between $\|\boldsymbol{x}\|_2$ and $\|\boldsymbol{x}\|_p$ arises when $\boldsymbol{x} = \mathbf{1}$.

We caution that the whole argument is rather informal and it takes some work to make it formal.

Note that here the problem is not convex, because $\{\boldsymbol{x} : \|\boldsymbol{x}\|_2^2 = 1\}$ is not convex and because we are asking to *maximize* a norm. In the following we will study Lagrange multipliers for general convex problems.

15.3.1 Karush-Kuhn Tucker Optimality Conditions for Convex Problems

A full formal treament of convex duality would require us to be more careful about using inf and sup in place of min and max, as well as considering problems that have no feasible solutions. Today, we'll ignore these concerns.

Let us consider a general convex optimization problem with convex objective, linear equality constraints and convex inequality constraints

$$\begin{aligned} &\min_{y \in S} \mathcal{E}(\boldsymbol{y}) \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b} \\ &\qquad \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}, \end{aligned} \tag{15.1}$$

where $\mathcal{E}(\boldsymbol{y}) : S \to \mathbb{R}$ is defined on a convex subset $S \subseteq \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{c}(\boldsymbol{y})$ is a vector of constraints $\boldsymbol{c}(\boldsymbol{y}) = (c_i(\boldsymbol{y}))_{i \in [k]}$. For every $i \in [k]$ the function $c_i : S \to \mathbb{R}$ should be convex, which implies the sublevel set $\{y : c_i(\boldsymbol{y}) \leq 0\}$ is convex. Given a solution $\boldsymbol{y}$, we say an inequality constraint is *tight* at $\boldsymbol{y}$ if $c_i(\boldsymbol{y}) = 0$. In the following we will denote by $\alpha^* = \mathcal{E}(\boldsymbol{y}^*)$ the optimal value of this program where $\boldsymbol{y}^*$ is a minimizer.

We will call this *the primal program* – and later we will see that we can associate another related convex program with any such convex program – and we will call this second program *the dual program.*

Definition 15.3.1 (Primal feasibility)**.** We say that $\boldsymbol{y} \in S$ is *primal feasible* if all constraints are satisfied, i.e. $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ and $\boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}$.

Now, as we did in our example with the 2- and p-norms, we will try to understand the relationship between the gradient of the objective function and of the constraint functions at an optimal solution $\boldsymbol{y}^*$.

An (not quite true!) intuition. Suppose $\boldsymbol{y}^*$ is an optimal solution to the convex program above. Let us additionally suppose that $\boldsymbol{y}^*$ is *not* on the boundary of S. Then, generally speaking, because we are at a constrained minimum of $\mathcal{E}(\boldsymbol{y}^*)$, we must have that for any infinitesimal $\boldsymbol{\delta}$ s.t. $\boldsymbol{y}^* + \boldsymbol{\delta}$ is also feasible, $\boldsymbol{\delta}^\top \boldsymbol{\nabla} \mathcal{E}(\boldsymbol{y}^*) \geq 0$, i.e. the infinitesimal does not decrease the objective. We can also view this as saying that if $\boldsymbol{\delta}^\top \boldsymbol{\nabla} \mathcal{E}(\boldsymbol{y}^*) < 0$, the update must be infeasible. But what kind of updates will make $\boldsymbol{y}^* + \boldsymbol{\delta}$ infeasible? This will be true

if $\boldsymbol{a}_j^\top \boldsymbol{\delta} \neq 0$ for some linear constraint j or, roughly speaking, $\boldsymbol{\nabla} c_i(\boldsymbol{y}^*)^\top \boldsymbol{\delta} \neq 0$ for some *tight* inequality constraint. But, if this is true for *all* directions that have a negative inner product with $\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*)$, then we must have that $-\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*)$ can be written as a linear combination of $\boldsymbol{a}_j$, i.e. gradients of the linear constraint, and of gradients $\boldsymbol{\nabla} c_i(\boldsymbol{y}^*)$ of tight inequality constraints, and furthermore the cofficients of the gradients of these tight inequality constraints must be *positive*, so that moving along this direction will increase the function value and violate the constraint.

To recap, given cofficients $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{s} \in \mathbb{R}^k$ with $\boldsymbol{s}(i) \geq 0$ if $c_i(\boldsymbol{y}^*) = 0$, and $\boldsymbol{s}(i) = 0$ otherwise, we should be able to write

$$-\boldsymbol{\nabla}_y \mathcal{E}(\boldsymbol{y}^*) = \sum_j \boldsymbol{x}(j)\boldsymbol{a}_j + \sum_i \boldsymbol{s}(i)\boldsymbol{\nabla} c_i(\boldsymbol{y}^*) \tag{15.2}$$

Note that since $\boldsymbol{y}^*$ is feasible, and hence $\boldsymbol{c}(\boldsymbol{y}^*) \leq 0$, we can write the condition that $\boldsymbol{s}(i) \geq 0$ if $c_i(\boldsymbol{y}^*) = 0$, and $\boldsymbol{s}(i) = 0$ otherwise, in a very slick way: namely as $\boldsymbol{s} \geq \boldsymbol{0}$ and $\boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}^*) = 0$. Traditionally, the variables in $\boldsymbol{s}$ are called *slack variables*, because of this, i.e. they are non-zero only if there is no *slack* in the constraint. This condition has a fancy name: when $\boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}) = 0$ for some feasible $\boldsymbol{y}$ and $\boldsymbol{s} \geq \boldsymbol{0}$, we say that $\boldsymbol{y}$ and $\boldsymbol{s}$ satisfy *complementary slackness*. We will think of the vectors $\boldsymbol{s}$ and $\boldsymbol{x}$ as variables that help us prove optimality of a current solution $\boldsymbol{y}$, and we call them *dual variables*.

Definition 15.3.2 (Dual feasibility)**.** We say $(\boldsymbol{x}, \boldsymbol{s})$ is dual feasible if $\boldsymbol{s} \geq 0$. If additionally $\boldsymbol{y}$ is primal feasible, we say $(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ is primal-dual feasible.

Now, we have essentially argued that at any optimal solution $\boldsymbol{y}^*$, we must have that complementary slackness holds, and that Equation (15.2) holds for some $\boldsymbol{x}$ and some $\boldsymbol{s} \geq \boldsymbol{0}$. However, while this intuitive explanation is largely correct, it turns out that it can fail for technical reasons in some weird situations[2]. Nonetheless, under some mild conditions, it is indeed true that the conditions we argued for above must hold at any optimal solution. These conditions have a name: The Karush-Kuhn-Tucker Conditions. For convenience, we will state the conditions using $\boldsymbol{\nabla}\boldsymbol{c}(\boldsymbol{y})$ to denote the matrix whose ith column is given by $\boldsymbol{\nabla} c_i(\boldsymbol{y})$. This is sometimes called the Jacobian of $\boldsymbol{c}$.

Definition 15.3.3 (The Karush-Kuhn-Tucker (KKT) Conditions)**.** Given a convex optimization problem of form (15.1) where the domain S is *open*. Suppose $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}$ satisfy the following conditions:

[2]Consider the following single-variable optimization problem

$$\begin{aligned} &\min_{x \in \mathbb{R}} x \\ &\text{s.t. } x^2 = 0. \end{aligned}$$

This has only a single feasible point $x = 0$, which must then be optimal. But at this point, the gradient of the constraint function is zero, while the gradient of the objective is non-zero. Thus our informal reasoning breaks down, because there exists an infeasible direction δ we can move along where the constraint function grows, but at a rate of $O(\delta^2)$.

- $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ and $\boldsymbol{c}(\boldsymbol{y}) \leq 0$ (primal feasibility)
- $\boldsymbol{s} \geq 0$ (dual feasibility)
- $\boldsymbol{\nabla}_y \mathcal{E}(\boldsymbol{y}) + \boldsymbol{A}^\top \boldsymbol{x} + \boldsymbol{\nabla}\boldsymbol{c}(\boldsymbol{y})\boldsymbol{s} = \boldsymbol{0}$ (KKT gradient condition, i.e. Eq. (15.2) restated)
- $\boldsymbol{s}(i) \cdot \boldsymbol{c}_i(\boldsymbol{y}) = 0$ for all i (complementary slackness)

Then we say that $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}$ satisfy the *Karush-Kuhn-Tucker* (KKT) conditions. Note that because of primal-dual feasibility, we can also write complementary slackness as $\boldsymbol{s}^T \boldsymbol{c}(\boldsymbol{y}) = 0$.

Note that we tried to informally argued that the KKT conditions must hold at an optimality solution (i.e. KKT is necessary for optimality), but this is not quite true without additional assumptions, as shown by our simple counter-example $\min_{x \in R, x^2 \leq 0} x$. On the other hand, it turns out that KKT is always sufficient for optimality, i.e. 'KKT at $(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$' $\Rightarrow$ '$\boldsymbol{y}$ is optimal' – as we will prove shortly (Theorem 15.3.12).

15.3.2 Slater's Condition

There exists many different mild technical conditions under which the KKT conditions do indeed hold at any optimal solution $\boldsymbol{y}$. The simplest and most useful is probably *Slater's condition*.

Definition 15.3.4 (Slater's condition with full domain)**.** A (primal) problem as defined in (15.1) with $S = \mathbb{R}^n$ fulfills Slater's condition if there exists a *strictly feasible* point, i.e. there exists $\tilde{\boldsymbol{y}}$ s.t. $\boldsymbol{A}\tilde{\boldsymbol{y}} = \boldsymbol{b}$ and $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$. This means that the strictly feasible point $\tilde{\boldsymbol{y}}$ lies strictly inside the set $\{\boldsymbol{y} : \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}\}$ defined by the inequality constraints.

One way to think about Slater's condition is that your inequality constraints should not restrict the solution space to be lower-dimensional. This is a degenerate case as the sublevel sets of the inequality constraints are generically full-dimensional and you want to avoid this degenerate case.

We can also extend Slater's condition to the case when the domain S is an open set. To extend Slater's condition to this case, we need the notion of a "relative interior".

Definition 15.3.5 (Relative interior)**.** Given a convex set $S \subset \mathbb{R}^n$, the *relative interior* of S is

$$\text{relint}(S) = \{\boldsymbol{x} \in S : \text{ for all } \boldsymbol{y} \in S \text{ there exists } \epsilon > 0 \text{ such that } \boldsymbol{x} + \epsilon(\boldsymbol{x} - \boldsymbol{y}) \in S\}.$$

In other words, $\boldsymbol{x} \in \text{relint}(S)$ if starting at $\boldsymbol{x} \in S$ we can move "away" from any $\boldsymbol{y} \in S$ by a little and still be in $\boldsymbol{S}$. As an example, suppose $S = \{(s,t) \in \mathbb{R}^2 \text{ such that } s \geq 0 \text{ and } t = 0\}$. Then $(0,0) \in S$ but $(0,0) \notin \text{relint}(S)$, while $(1,0) \in \text{relint}(S)$.

Now, we can state a more general version of Slater's condition.

Definition 15.3.6 (Slater's condition)**.** A (primal) problem as defined in (15.1) fulfills Slater's condition if there exists a *strictly feasible* point $\tilde{\boldsymbol{y}} \in \text{relint}(S)$. We require $\boldsymbol{A}\tilde{\boldsymbol{y}} = \boldsymbol{b}$ and $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$. This means that the strictly feasible point $\tilde{\boldsymbol{y}}$ lies strictly inside the set $\{\boldsymbol{y} : \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}\}$ defined by the inequality constraints.

Finally, we end with a proposition that tells us that given Slater's condition, the KKT are indeed necessary for optimality of our convex programs.

Proposition 15.3.7 (Given Slater's condition, KKT is necessary for optimality, i.e. optimal solution $\Rightarrow$ KKT.)**.** *Consider a convex program in the form* (15.1) *that satisfies Slater's condition and has an open set S as its domain. Suppose $\boldsymbol{y}$ is a primal optimal (feasible) solution, then $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}$ satisfy the KKT conditions.*

We will prove this proposition later, after developing some tools we will use in the proof. In fact, we will also see later that assuming Slater's condition, the KKT conditions are sufficient for optimality.

15.3.3 The Lagrangian and The Dual Program

Notice that we can also rewrite the KKT gradient condition as

$$\boldsymbol{\nabla}_{\boldsymbol{y}} \left[\underbrace{\mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) + \boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y})}_{(\star)} \right] = \boldsymbol{0}$$

That is, we can write this condition as the gradient of the quantity $(\star)$ is zero. But what is this quantity $(\star)$? We call it *the Lagrangian* of the program.

Definition 15.3.8. Given a convex program (15.1), we define the *Lagrangian* of the program as

$$L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) = \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) + \boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}).$$

We can think of $\boldsymbol{x}$ as assigning a price to violating the linear constraints, and of $\boldsymbol{s}$ as assigning a price to violating the inequality constraints. The KKT gradient condition tells us that at the given prices, the there is no benefit gained from locally violating the constraints – i.e. changing the primal solution $\boldsymbol{y}$ would not improve the cost.

Notice that if $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}$ are primal-dual feasible, then

$$\begin{aligned} \mathcal{E}(\boldsymbol{y}) &= \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) && \text{as } \boldsymbol{b} - \boldsymbol{A}\boldsymbol{y} = \boldsymbol{0}. \\ &\geq \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) + \boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}) && \text{as } \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0} \text{ and } \boldsymbol{s} \geq \boldsymbol{0}. \\ &= L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) && (15.3) \end{aligned}$$

Thus, for primal-dual feasible variables, the Lagrangian is always a lower bound on the objective value.

The primal problem can be written in terms of the Lagrangian.

$$\alpha^* = \min_{\boldsymbol{y}} \max_{\boldsymbol{x};\boldsymbol{s}\geq \mathbf{0}} L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \tag{15.4}$$

This is because for a minimizing $\boldsymbol{y}$ all constraints have to be satisfied and the Lagrangian simplifies to $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) = \mathcal{E}(\boldsymbol{y})$. If $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \mathbf{0}$ was violated, making $\boldsymbol{x}$ large sends $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \to \infty$. And if $\boldsymbol{c}(\boldsymbol{y}) \leq \mathbf{0}$ is violated, we can make $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \to \infty$ by choosing large $\boldsymbol{s}$.

Note that we require $\boldsymbol{s} \geq \mathbf{0}$, as we only want to penalize the violation of the inequality constraints in one direction, i.e. when $\boldsymbol{c}(\boldsymbol{y}) > 0$.

We also define a Lagrangian only in terms of the dual variables by minimizing over $\boldsymbol{y}$ as

$$L(\boldsymbol{x}, \boldsymbol{s}) = \min_{\boldsymbol{y}} L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}).$$

When $\boldsymbol{s} \geq 0$, we have that $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ is a convex function of $\boldsymbol{y}$. For each $\boldsymbol{y}$, the Lagrangian $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ is linear in $(\boldsymbol{x}, \boldsymbol{s})$ and hence also concave in them. Hence $L(\boldsymbol{x}, \boldsymbol{s})$ is a concave function, because it is the pointwise minimum (over $\boldsymbol{y}$), of a collection of concave functions in $(\boldsymbol{x}, \boldsymbol{s})$.

$L(\boldsymbol{x}, \boldsymbol{s})$ is defined by minimizing $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ over $\boldsymbol{y}$, i.e. what is the worst case value of the lower bound $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ across all $\boldsymbol{y}$. We can think of this as computing how good the given "prices" are at approximately enforcing the constraints. This naturally leads to a new optimization problem: How can we choose our prices $\boldsymbol{x}, \boldsymbol{s}$ to get the best (highest) possible lower bound?

Definition 15.3.9 (Dual problem)**.** We define the *dual problem* as

$$\max_{\substack{\boldsymbol{x},\boldsymbol{s} \\ \boldsymbol{s}\geq 0}} \min_{\boldsymbol{y}} L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) = \max_{\substack{\boldsymbol{x},\boldsymbol{s} \\ \boldsymbol{s}\geq 0}} L(\boldsymbol{x}, \boldsymbol{s}) \tag{15.5}$$

and denote the optimal dual value by β^*.

The dual problem is really a convex optimization problem in disguise, because we can flip the sign of $-L(\boldsymbol{x}, \boldsymbol{s})$ to get a convex function and minimizing this is equivalent to maximizing $L(\boldsymbol{x}, \boldsymbol{s})$.

$$\max_{\substack{\boldsymbol{x},\boldsymbol{s} \\ \boldsymbol{s}\geq 0}} L(\boldsymbol{x}, \boldsymbol{s}) = -\min_{\substack{\boldsymbol{x},\boldsymbol{s} \\ \boldsymbol{s}\geq 0}} -L(\boldsymbol{x}, \boldsymbol{s})$$

When $\boldsymbol{x}$ and $\boldsymbol{s}$ are optimal for the dual program, we say they are dual optimal. And for convenience, when we also have a primal optimal $\boldsymbol{y}$, altogether, we will say that $(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ are primal-dual optimal.

For any primal-dual feasible $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}$ we have $L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \leq \mathcal{E}(\boldsymbol{y})$ (see Equation (15.3)) and hence also $L(\boldsymbol{x}, \boldsymbol{s}) = \min_y L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) \leq \mathcal{E}(\boldsymbol{y})$.

In other words $\max_{\boldsymbol{x};\boldsymbol{s}\geq 0} L(\boldsymbol{x}, \boldsymbol{s}) = \beta^* \leq \alpha^*$. This is referred to as *weak duality*.

Using the forms in Equations (15.5) and (15.4), we can also state this as

Theorem 15.3.10 (The Weak Duality Theorem)**.** *For any convex program* (15.5) *and its dual, we have*

$$\alpha^* = \min_{\boldsymbol{y}} \max_{\boldsymbol{x};\boldsymbol{s}\geq \boldsymbol{0}} L(\boldsymbol{y},\boldsymbol{x},\boldsymbol{s}) \geq \max_{\boldsymbol{x};\boldsymbol{s}\geq \boldsymbol{0}} \min_{\boldsymbol{y}} L(\boldsymbol{y},\boldsymbol{x},\boldsymbol{s}) = \beta^*.$$

15.3.4 Strong Duality and KKT

So now that we have proved weak duality $\beta^* \leq \alpha^*$, what is strong duality? $\beta^* = \alpha^*$? The answer is yes, but strong duality only holds under some conditions. Again, a simple sufficient condition is Slater's condition (Definition 15.3.6).

Theorem 15.3.11. *For a program* (15.1) *satisfying Slater's condition, strong duality holds, i.e.* $\alpha^* = \beta^*$*. In other words, the optimal value of the primal problem* α^* *is equal to the optimal value of the dual.*

Note that at primal optimal $\boldsymbol{y}^*$ and dual optimal $\boldsymbol{x}^*, \boldsymbol{s}^*$, we have

$$\alpha^* = \mathcal{E}(\boldsymbol{y}^*) \geq L(\boldsymbol{y}^*,\boldsymbol{x}^*,\boldsymbol{s}^*) \geq \beta^* = \alpha^*.$$

Thus, we can conclude that $L(\boldsymbol{y}^*,\boldsymbol{x}^*,\boldsymbol{s}^*) = \alpha^* = \beta^*$.

KKT is sufficient for optimality. We introduced the KKT conditions by informally thinking about conditions that should be true at a local extremum. In fact, for convex problems (with sufficient differentiability), the KKT conditions are sufficient for optimality, as the next theorem shows. As a bonus, we also get strong duality when they hold.

Theorem 15.3.12. *Consider a convex program* (15.1) *with an open domain set* S*. Then if the KKT conditions hold at* $(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}})$*, they must be primal-dual optimal, and strong duality must hold.*

Proof. $\tilde{\boldsymbol{y}}$ is global minimizer of $\boldsymbol{y} \mapsto L(\boldsymbol{y},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}})$, since this function is convex with vanishing gradient at $\tilde{\boldsymbol{y}}$. Hence,

$$L(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}}) = \inf_{\boldsymbol{y}} L(\boldsymbol{y},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}}) = L(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}}) \leq \beta^*.$$

On the other hand, due to primal feasibility and complementary slackness,

$$L(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}}) = \mathcal{E}(\tilde{\boldsymbol{y}}) + \tilde{\boldsymbol{x}}^\top(\boldsymbol{b} - \boldsymbol{A}^\top\tilde{\boldsymbol{y}}) + \tilde{\boldsymbol{s}}^\top\boldsymbol{c}(\tilde{\boldsymbol{y}}) = \mathcal{E}(\tilde{\boldsymbol{y}}) \geq \alpha^*.$$

Thus, $\beta^* \geq \alpha^*$. But also $\beta^* \leq \alpha^*$ by weak duality. Therefore, $\beta^* = \alpha^*$ and $\tilde{\boldsymbol{y}},\tilde{\boldsymbol{x}},\tilde{\boldsymbol{s}}$ are primal/dual optimal. □

When strong duality holds, KKT is necessary for optimality. When strong duality holds, the KKT conditions must necessarily hold at any optimal solution.

Consider a convex program (15.1) with an open set S as its domain. Suppose $\boldsymbol{y}^*$ is an optimizer of the primal problem and $\boldsymbol{x}^*, \boldsymbol{s}^*$ for the dual, and suppose that strong duality holds. We thus have

$$L(\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{s}^*) = \alpha^* = \beta^*.$$

Because $L(\boldsymbol{y}, \boldsymbol{x}^*, \boldsymbol{s}^*)$ is a convex function in $\boldsymbol{y}$, it also follows that as $\mathcal{E} : S \to \mathbb{R}$ and $\boldsymbol{c}$ are differentiable then we must have that the gradient w.r.t. $\boldsymbol{y}$ is zero, i.e.

$$\boldsymbol{\nabla}_{\boldsymbol{y}} L(\boldsymbol{y}, \boldsymbol{x}^*, \boldsymbol{s}^*)|_{\boldsymbol{y}=\boldsymbol{y}^*} = 0 \tag{15.6}$$

This says exactly that the KKT gradient condition holds at $(\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{s}^*)$.

Next, we want to confirm that complementary slackness holds for our optimal pair $(\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{s}^*)$. We can see that

$$\mathcal{E}(\boldsymbol{y}^*) = \alpha^* = \mathcal{E}(\boldsymbol{y}^*) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}^*) + \boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}^*) = \mathcal{E}(\boldsymbol{y}^*) + \boldsymbol{s}^\top \boldsymbol{c}(\boldsymbol{y}^*)$$

and hence when the i-th convex constraint is not active, i.e. $c_i(\boldsymbol{y}^*) < 0$ the *slack* must be zero, i.e. $\boldsymbol{s}(i) = 0$. Conversely if the slack is non-zero, that is $\boldsymbol{s}(i) \neq 0$ implies that the constraint is active, i.e. $c_i(\boldsymbol{y}^*) = 0$. This says precisely that the complementary slackness condition holds at primal-dual optimal $(\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{s}^*)$. Combined with our previous observation Equation (15.6), we get the following result.

Theorem 15.3.13. *Consider a convex program* (15.1) *with an open domain set S and whose dual satisfies strong duality. Then KKT conditions necessarily hold at primal-dual optimal* $(\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{s}^*)$.

Theorem 15.3.13 combined with Theorem 15.3.11 immediately imply Proposition 15.3.7.

A good reference for basic convex duality theory is Boyd's free online book "Convex optimization" (linked to on the course website). It provides a number of different interpretations of duality. One particularly interesting one comes from economics: economists see the slack variables $\boldsymbol{s}$ as prices for violating the constraints.

15.3.5 Proof that Slater's Condition Implies Strong Duality

In this section, we'll prove Theorem 15.3.11. But, before we prove the theorem, let's make a few observations to get us warmed up. If you get bored, skip ahead to the proof.

It is sufficient to prove that $\alpha^* \leq \beta^*$, as the statement then follows in conjunction with weak duality. We define the set

$$G = \{(\mathcal{E}(\boldsymbol{y}), \boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}, \boldsymbol{c}(\boldsymbol{y})) : \boldsymbol{y} \in S\},$$

where $S \subseteq \mathbb{R}^n$ is the domain of $\mathcal{E}$.

Immediately, we observe that we can write the optimal primal value as

$$\alpha^* = \min\{t : (t, \boldsymbol{v}, \boldsymbol{u}) \in G, \boldsymbol{v} = \boldsymbol{0}, \boldsymbol{u} \leq \boldsymbol{0}\}.$$

Similarly, we can write the Lagrangian (after minimizing over $\boldsymbol{y}$)

$$L(\boldsymbol{x}, \boldsymbol{s}) = \min_{(t,\boldsymbol{v},\boldsymbol{u})\in G} (1, \boldsymbol{x}, \boldsymbol{s})^\top (t, \boldsymbol{v}, \boldsymbol{u}).$$

This is equivalent to the inequality, for $(t, \boldsymbol{v}, \boldsymbol{u}) \in G$,

$$(1, \boldsymbol{x}, \boldsymbol{s})^\top (t, \boldsymbol{v}, \boldsymbol{u}) \geq L(\boldsymbol{x}, \boldsymbol{s}).$$

which defines a hyperplane with $\boldsymbol{n} = (1, \boldsymbol{x}, \boldsymbol{s})$ and $\mu = L(\boldsymbol{x}, \boldsymbol{s})$ such that G is on one side.

To establish strong duality, we would like to show the existence of a hyperplane such that for $(t, \boldsymbol{v}, \boldsymbol{u}) \in G$

$$\boldsymbol{n}^\top (t, \boldsymbol{v}, \boldsymbol{u}) \geq \alpha^* \text{ and } \boldsymbol{n} = (1, \widehat{\boldsymbol{x}}, \widehat{\boldsymbol{s}}) \text{ with } \widehat{\boldsymbol{s}} \geq \boldsymbol{0}.$$

Then we would immediately get

$$\beta^* \geq L(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{s}}) = \min_{(t,\boldsymbol{v},\boldsymbol{u})\in G} (1, \boldsymbol{x}, \boldsymbol{s})^\top (t, \boldsymbol{v}, \boldsymbol{u}) \geq \alpha^*.$$

Perhaps not surprisingly, we will use the Separating Hyperplane Theorem. What are the challenges we need to deal with?

- We need to replace G with a convex set (which we will call A) and separate A from some other convex set (which we will call B).
- We need to make sure the hyperplane normal $\boldsymbol{n}$ has 1 in the first coordinate and $\boldsymbol{s} \geq \boldsymbol{0}$, and the hyperplane threshold is α^*.

Proof of Theorem 15.3.11. For simplicity, our proof will assume that $S = \mathbb{R}^n$, but only a little extra work is required to handle the general case.

Let's move to on finding two convex disjoints sets A, B to enable the use of the separating hyperplane Theorem 15.2.4.

First set we define A, roughly speaking, as a multi-dimensional epigraph of G. More precisely

$$A = \{(t, \boldsymbol{v}, \boldsymbol{u}) : \exists \boldsymbol{y} \in S, t \geq \mathcal{E}(\boldsymbol{y}), \boldsymbol{v} = \boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}, \boldsymbol{u} \geq \boldsymbol{c}(\boldsymbol{y})\}.$$

Note that A is a convex set. The proof is similar to the proof that the epigraph of a convex function is a convex set. The optimal value of the primal program can be now written as

$$\alpha^* = \min_{(t,\boldsymbol{0},\boldsymbol{0})\in A} t.$$

And we define another set B of the same dimensionality as A by

$$B := \{(r \in \mathbb{R}, \boldsymbol{0} \in \mathbb{R}^m, \boldsymbol{0} \in \mathbb{R}^k) : r < \alpha^*\}.$$

This set B is convex, as it is a ray. An example of two such sets A, B is illustrated in Figure 15.3.

We show that $A \cap B = \emptyset$ by contradiction. Suppose A, B are not disjoint; then there exists $\boldsymbol{y}$ such that

$$(\mathcal{E}(\boldsymbol{y}), \boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}, \boldsymbol{c}(\boldsymbol{y})) = (r, \boldsymbol{0}, \boldsymbol{u})$$

with $\boldsymbol{u} \leq \boldsymbol{0}$. But this means that $\boldsymbol{y}$ is feasible and $\mathcal{E}(\boldsymbol{y}) = r < \alpha^*$; contradicting the optimality of α^*.

To make things simpler, we assume that our linear constraint matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, has full row rank and $m < n$ (but very little extra work is required to deal with the remaining cases, which we omit).

As we just proved, A and B are convex and disjoint sets and hence the separating hyperplane theorem (Theorem 15.2.4) we introduced earlier in this chapter implies the existence a separating hyperplane. This means there exists a normal $\boldsymbol{n} = (\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}})$ and threshold μ and with A on one side, i.e.

$$(t, \boldsymbol{v}, \boldsymbol{u}) \in A \implies (t, \boldsymbol{v}, \boldsymbol{u})^\top (\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}}) \geq \mu \tag{15.7}$$

and the set B on the other side:

$$(t, \boldsymbol{v}, \boldsymbol{u}) \in B \implies (t, \boldsymbol{v}, \boldsymbol{u})^\top (\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}}) \leq \mu. \tag{15.8}$$

Now, we claim that $\tilde{\boldsymbol{s}} \geq 0$. Suppose $\tilde{\boldsymbol{s}}(i) < 0$, then for $\boldsymbol{u}(i) \to \infty$ the threshold would grow unbounded, i.e. $\mu \to -\infty$ contradicting that the threshold μ is finite by the separating hyperplane theorem. Similarly we claim $\tilde{\rho} \geq 0$, as if this were not the case, having $t \to \infty$ implies that $\mu \to -\infty$ again contradicting the finiteness of μ.

From Equation (15.8) it follows that $t\tilde{\rho} \leq \mu$ for all $t < \alpha^*$ which implies that $t\tilde{\rho} \leq \mu$ for $t = \alpha^*$ by taking the limit. Hence we have $\alpha^* \tilde{\rho} \leq \mu$. From $(t, \boldsymbol{v}, \boldsymbol{u}) \in A$ we get from Equation (15.7)

$$(\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}})^\top (t, \boldsymbol{v}, \boldsymbol{u}) \geq \mu \geq \alpha^* \tilde{\rho}$$

and thus

$$(\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}})^\top (\mathcal{E}(\boldsymbol{y}), \boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}, \boldsymbol{c}(\boldsymbol{y})) \geq \alpha^* \tilde{\rho}. \tag{15.9}$$

Now we consider two cases; starting with the "good" case where $\tilde{\rho} > 0$. Dividing Equation (15.9) by $\tilde{\rho}$ gives

$$\mathcal{E}(\boldsymbol{y}) + \frac{\tilde{\boldsymbol{x}}^\top}{\tilde{\rho}} (\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}) + \frac{\tilde{\boldsymbol{s}}^\top}{\tilde{\rho}} \boldsymbol{c}(\boldsymbol{y}) \geq \alpha^*.$$

Noting that the left hand side above is $L(\boldsymbol{y}, \frac{\tilde{\boldsymbol{x}}}{\tilde{\rho}}, \frac{\tilde{\boldsymbol{s}}}{\tilde{\rho}})$ and that the equation holds for arbitrary $\boldsymbol{y}$; therefore also for the minimum we get

$$\min_{\boldsymbol{y}} L\left(\boldsymbol{y}, \frac{\tilde{\boldsymbol{x}}}{\tilde{\rho}}, \frac{\tilde{\boldsymbol{s}}}{\tilde{\rho}}\right) \geq \alpha^*$$

and hence via definition of β^* finally

$$\beta^* \geq L\left(\frac{\tilde{\boldsymbol{x}}}{\tilde{\rho}}, \frac{\tilde{\boldsymbol{s}}}{\tilde{\rho}}\right) \geq \alpha^*.$$

Next consider the "bad" case $\tilde{\rho} = 0$. As $\alpha^* \tilde{\rho} \leq \mu$, we have $0 \leq \mu$. From Equation (15.7) we get

$$\boldsymbol{c}(\boldsymbol{y})^\top \boldsymbol{s} + \boldsymbol{x}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) \geq \mu \geq 0.$$

As Slater's condition holds, there is an interior point $\tilde{\boldsymbol{y}}$, i.e. it satisfies $\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{y}} = \boldsymbol{0}$ and $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$. Together with the equation above this yields

$$\boldsymbol{c}(\tilde{\boldsymbol{y}})^\top \tilde{\boldsymbol{s}} + \tilde{\boldsymbol{x}}^\top \boldsymbol{0} \geq 0$$

which implies $\boldsymbol{c}(\tilde{\boldsymbol{y}})^\top \tilde{\boldsymbol{s}} \geq 0$ and as $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$ this means $\tilde{\boldsymbol{s}} = \boldsymbol{0}$.

As the normal $(\tilde{\rho}, \tilde{\boldsymbol{s}}, \tilde{\boldsymbol{x}})$ of the hyperplane can not be all zeroes, this means the last "component" $\tilde{\boldsymbol{x}}$ must contain a non-zero entry, i.e. $\tilde{\boldsymbol{x}} \neq \boldsymbol{0}$. Furthermore $\tilde{\boldsymbol{x}}^\top (\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{y}}) = \boldsymbol{0}$, $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$ and $\boldsymbol{A}$ has full row rank, hence there exists $\boldsymbol{\delta}$ such that

$$\tilde{\boldsymbol{x}}^\top (\boldsymbol{b} - \boldsymbol{A}(\tilde{\boldsymbol{y}} + \boldsymbol{\delta})) < \boldsymbol{0} \text{ and } \boldsymbol{c}(\tilde{\boldsymbol{y}} + \boldsymbol{\delta}) < 0.$$

This, however, means that there is a point in A on the wrong side of the hyperplane, as

$$(\tilde{\rho}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{s}})^\top (\mathcal{E}(\tilde{\boldsymbol{y}} + \boldsymbol{\delta}), \boldsymbol{b} - \boldsymbol{A}(\tilde{\boldsymbol{y}} + \boldsymbol{\delta}), \boldsymbol{c}(\tilde{\boldsymbol{y}} + \boldsymbol{\delta})) < 0$$

but the threshold is $\mu \geq 0$. □

Remark. Note that our reasoning about why $\boldsymbol{s} \geq \boldsymbol{0}$ in the proof above is very similar to our reasoning for why the primal program can be written as Problem (15.4).

Example. As an example of A and B as they appear in the above proof, consider

$$\min_{\substack{y \in (0,\infty) \\ 1/y - 1 \leq 0}} y^2$$

This leads to $\alpha^* = 1, y^* = 1$, and $A = \{(t, u) : y \in (0, \infty) \text{ and } t > y^2 \text{ and } u \geq 1/y - 1\}$, and $B = \{(t, 0) : t < 1\}$ and the separating hyperplane normal is $\boldsymbol{n} = (1, 2)$. These two sets A, B are illustrated in Figure 15.3.

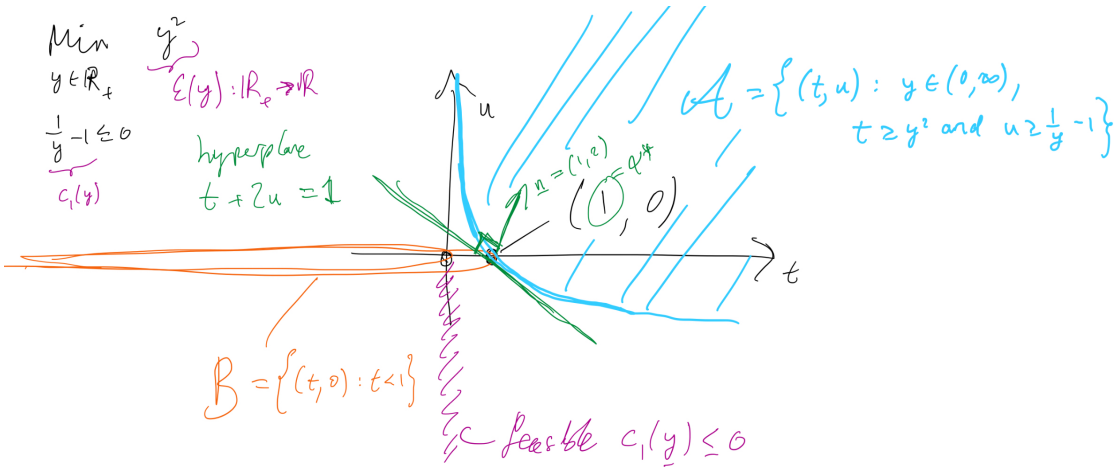


Figure 15.3: Example of the convex sets A and B we wish to separate by hyperplane.

Chapter 16

Fenchel Conjugates and Newton's Method

16.1 Lagrange Multipliers and Convex Duality Recap

Recall the convex optimization program we studied last chapter,

$$\begin{aligned} \min \quad & \mathcal{E}(\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b} \\ & \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}, \end{aligned} \tag{16.1}$$

where $\mathcal{E}(\boldsymbol{y}) : S \to \mathbb{R}$ is defined on a subset $S \subseteq \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{c}(\boldsymbol{y})$ is a vector of constraints $\boldsymbol{c}(\boldsymbol{y}) = (c_i(\boldsymbol{y}))_{i \in [k]}$. For every $i \in [k]$ the function $c_i : S \to \mathbb{R}$ is convex. We call (16.1) the primal (program) and denote its optimal value by α^*.

The associated Lagrangian is defined by

$$L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}) = \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) + \boldsymbol{s}^T\boldsymbol{c}(\boldsymbol{y}).$$

where $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{s} \in \mathbb{R}^k$ are dual variables. The dual (program) is given by

$$\max_{\substack{\boldsymbol{x}, \boldsymbol{s} \\ \boldsymbol{s} \geq 0}} \quad L(\boldsymbol{x}, \boldsymbol{s}) \tag{16.2}$$

whose optimal value is denoted by β^*. The dual is always a convex optimization program even though the primal is non-convex. The optimal value of the primal (16.1) can also be written as

$$\alpha^* = \inf_{\boldsymbol{y}} \sup_{\boldsymbol{x}; \boldsymbol{s} \geq \boldsymbol{0}} L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}), \tag{16.3}$$

where no constraint is imposed on the primal variable $\boldsymbol{y}$. The optimal value of the dual (16.2) is

$$\beta^* = \sup_{\boldsymbol{x}; \boldsymbol{s} \geq \boldsymbol{0}} \inf_{\boldsymbol{y}} L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s}). \tag{16.4}$$

Note the only difference between (16.3) and (16.4) is that the positions of "inf" and "sup" are swapped. The weak duality theorem states that the dual optimal value is a lower bound of the primal optimal value, i.e. $\beta^* \leq \alpha^*$.

The Slater's condition for (16.1) open domain S requires the existence of a *strictly feasible* point, i.e. there exists $\tilde{\boldsymbol{y}} \in S$ s.t. $\boldsymbol{A}\tilde{\boldsymbol{y}} = \boldsymbol{b}$ and $\boldsymbol{c}(\tilde{\boldsymbol{y}}) < \boldsymbol{0}$. This means that the strictly feasible point $\tilde{\boldsymbol{y}}$ lies inside the interior of the set $\{\boldsymbol{y} : \boldsymbol{c}(\boldsymbol{y}) \leq \boldsymbol{0}\}$ defined by the inequality constraints. If the domain S is not open, Slater's condition also requires that a strictly feasible point is in the relative interior of S (NB: when S is open, S is equal to its relative interior). The strong duality theorem says that Slater's condition implies strong duality, $\beta^* = \alpha^*$.

We were also introduced to the KKT conditions, and we saw that for our convex programs with continuously differentiable objective and constraints functions, when the domain is open, the conditions are sufficient to imply strong duality and primal-dual optimality of the points that satisfy them, KKT $\implies$ '$(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{s})$ primal-dual optimal and we have strong duality'. Finally, we saw that when strong duality holds, KKT must hold at the primal-dual optimal solutions.

In summary: *Slater's condition* $\implies$ *strong duality* and *strong duality* $\iff$ *KKT*.

Example. In Chapter 11, we gave a combinatorial proof of the min-cut max-flow theorem, and showed that the min-cut program can be expressed as a linear program. Now, we will use the strong duality theorem to give an alternative proof, and directly find the min-cut linear program is the dual program to our maximum flow linear program.

We will assume that Slater's condition holds for our primal program. Since scaling the flow down enough will always ensure that capacity constraints are strictly satisfied i.e. $\boldsymbol{f} < \boldsymbol{c}$, the only concern is to make sure that non-negativity constraints are satisfied. This means that there is an s-t flow that sends a non-zero flow on every edge. In fact, this may not always be possible, but it is easy to detect such edges and remove them without changing the value of the program: an edge (u, v) should be removed if there is no path s to u or no path v to t. We can identify all such edges using a BFS from s along the directed edges and a BFS along reversed directed edges from t.

$$\min_{\substack{F\in\mathbb{R} \\ \boldsymbol{B}\boldsymbol{f}=F\boldsymbol{b}_{s,t} \\ \boldsymbol{0}\leq\boldsymbol{f}\leq\boldsymbol{c}}} -F = \min_{F;\boldsymbol{f}\geq\boldsymbol{0}} \max_{\boldsymbol{x};\boldsymbol{s}\geq\boldsymbol{0}} -F + \boldsymbol{x}^\top(F\boldsymbol{b}_{s,t} - \boldsymbol{B}\boldsymbol{f}) + (\boldsymbol{f}-\boldsymbol{c})^\top\boldsymbol{s}$$

(Slater's condition $\implies$ strong duality)

$$-\max_{\substack{F\in\mathbb{R} \\ \boldsymbol{B}\boldsymbol{f}=F\boldsymbol{b}_{s,t} \\ \boldsymbol{0}\leq\boldsymbol{f}\leq\boldsymbol{c}}} F = \max_{\boldsymbol{x};\boldsymbol{s}\geq\boldsymbol{0}} \min_{F;\boldsymbol{f}\geq\boldsymbol{0}} F(\boldsymbol{b}_{s,t}^\top\boldsymbol{x} - 1) + \boldsymbol{f}^\top(\boldsymbol{s} - \boldsymbol{B}^\top\boldsymbol{x}) - \boldsymbol{c}^\top\boldsymbol{s}$$

$$= \max_{\substack{\boldsymbol{x};\,\boldsymbol{s}\geq\boldsymbol{0} \\ \boldsymbol{b}_{s,t}^\top\boldsymbol{x} = 1 \\ \boldsymbol{s}\geq\boldsymbol{B}^\top\boldsymbol{x}}} -\boldsymbol{c}^\top\boldsymbol{s}$$

Thus switching signs gives us

$$\max_{\substack{F\in\mathbb{R}\\ \boldsymbol{B}\boldsymbol{f}=F\boldsymbol{b}_{s,t}\\ \boldsymbol{0}\le\boldsymbol{f}\le\boldsymbol{c}}} F = \min_{\substack{\boldsymbol{x};\,\boldsymbol{s}\ge\boldsymbol{0}\\ \boldsymbol{b}_{s,t}^\top\boldsymbol{x}=1\\ \boldsymbol{s}\ge\boldsymbol{B}^\top\boldsymbol{x}}} \boldsymbol{c}^\top\boldsymbol{s} \tag{16.5}$$

The LHS of (16.5) is exactly the LP formulation of max-flow, while the RHS is exactly the LP formulation of min-cut. Note that we treated the "constraint" $\boldsymbol{0}\le\boldsymbol{f}$ as a restriction on the domain of $\boldsymbol{f}$ rather than a constraint with a dual variable associated with it. We always have this kind of flexibility when deciding how to compute a dual, and some choices may lead to a simpler dual program than others.

16.2 Fenchel Conjugates

In this section we will learn about Fenchel conjugates. This is a notion of dual function that is closely related to duality of convex programs, and we will learn more about how dual programs behave by studying these functions.

Definition 16.2.1 (Fenchel conjugate)**.** 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

$$\mathcal{E}^*(\boldsymbol{z}) = \sup_{\boldsymbol{y}\in S} \langle \boldsymbol{z}, \boldsymbol{y}\rangle - \mathcal{E}(\boldsymbol{y}).$$

Remark 16.2.2. $\mathcal{E}^*$ is a convex function whether $\mathcal{E}$ is convex or not, since $\mathcal{E}^*(z)$ is pointwise supremum of a family of convex (here, affine) functions of z.

In this course, we have only considered convex functions that are real-valued and continuous and defined on a convex domain. For any such $\mathcal{E}$, we have $\mathcal{E}^{**} = \mathcal{E}$, i.e. the Fenchel conjugate of the Fenchel conjugate is the original function. This is a consequence of the Fenchel-Moreau theorem, which establishes this under slightly more general conditions. We will not prove this generally, but as part of Theorem 16.2.3 below, we sketch a proof under more restrictive assumptions.

Example. Let $\mathcal{E}(\boldsymbol{y}) = \frac{1}{p}\|\boldsymbol{y}\|_p^p$ $(p > 1)$. We want to evaluate its Fenchel conjugate $\mathcal{E}^*$ at any given point $\boldsymbol{z} \in \mathbb{R}^n$. Since $\mathcal{E}$ is convex and differentiable, the supremum must be achieved at some $\boldsymbol{y}$ with vanishing gradient

$$\boldsymbol{\nabla}_{\boldsymbol{y}}\langle \boldsymbol{z}, \boldsymbol{y}^*\rangle - \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*) = \boldsymbol{z} - \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*) = \boldsymbol{0} \iff \boldsymbol{z} = \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*).$$

It's not difficult to see, for all i,

$$\boldsymbol{z}(i) = \operatorname{sgn}(\boldsymbol{y}(i))|\boldsymbol{y}(i)|^{p-1}.$$

Then,

$$\begin{aligned}\mathcal{E}^*(\boldsymbol{z}) &= \langle \boldsymbol{z}, \boldsymbol{y} \rangle - \mathcal{E}(\boldsymbol{y}) \\ &= \sum_i |\boldsymbol{z}(i)|^{\frac{1}{p-1}+1} - \frac{1}{p}|\boldsymbol{z}(i)|^{\frac{p}{p-1}} \\ &\quad (\text{define } q \text{ s.t. } \frac{1}{q} + \frac{1}{p} = 1) \\ &= \frac{1}{q}\|\boldsymbol{z}\|_q^q.\end{aligned}$$

More generally, given a convex and differentiable function $\mathcal{E} : S \to \mathbb{R}$, if there exists $\boldsymbol{y} \in S$ s.t. $\boldsymbol{z} = \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})$, then $\mathcal{E}^*(\boldsymbol{z}) = (\boldsymbol{y}^*)^\top \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*) - \mathcal{E}(\boldsymbol{y}^*)$. The Fenchel conjugate and Lagrange duality are closely related, which is demonstrated in the following example.

Example. Consider a convex optimization program with only linear constraints,

$$\begin{aligned}\min_{\boldsymbol{y}\in\mathbb{R}^n} \quad & \mathcal{E}(\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}\end{aligned}$$

where $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ is a convex function and $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. Then the corresponding dual program is

$$\begin{aligned}\sup_{\boldsymbol{x}\in\mathbb{R}^n} \inf_{\boldsymbol{y}\in\mathbb{R}^n} \mathcal{E}(\boldsymbol{y}) + \boldsymbol{x}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{y}) &= \sup_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{b}^\top\boldsymbol{x} - \sup_{\boldsymbol{y}\in\mathbb{R}^n} \left(\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{y} - \mathcal{E}(\boldsymbol{y})\right) \\ &= \sup_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{b}^\top\boldsymbol{x} - \mathcal{E}^*(\boldsymbol{A}^\top\boldsymbol{x})\end{aligned}$$

Theorem 16.2.3 (Properties of the Fenchel conjugate)**.** *Consider a strictly convex function $\mathcal{E} : S \to \mathbb{R}$ where $S \subseteq R^n$ is an open convex set. When $\mathcal{E}$ is differentiable with a Hessian that is positive definite everywhere and its gradient $\boldsymbol{\nabla}\mathcal{E}$ is surjective onto $\mathbb{R}^n$, we have the following three properties:*

1. $\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{z})) = \boldsymbol{z}$ *and* $\boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})) = \boldsymbol{y}$

2. $(\mathcal{E}^*)^* = \mathcal{E}$*, i.e. the Fenchel conjugate of the Fenchel conjugate is the original function.*

3. $\boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})) = \boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})$

primal point $\boldsymbol{y}$	$\xrightarrow{\nabla_y \mathcal{E}}$ $\xleftarrow{\nabla_z \mathcal{E}^*}$	dual point $\boldsymbol{z}$
gradient $\nabla_y \mathcal{E}(\boldsymbol{y}) = \boldsymbol{z}$		gradient $\nabla_z \mathcal{E}^*(\boldsymbol{z}) = \boldsymbol{y}$
Hessian $\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}) = \boldsymbol{H}^{-1}_{\mathcal{E}^*}(\boldsymbol{z})$		Hessian $\boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{y}) = \boldsymbol{H}^{-1}_{\mathcal{E}}(\boldsymbol{y})$

Figure 16.1: Properties of Fenchel conjugate

Proof sketch. Part 1. Because the gradient $\nabla_y \mathcal{E}$ is surjective onto $\mathbb{R}^n$, given any $\boldsymbol{z} \in \mathbb{R}^n$, there exists a $\boldsymbol{y}$ such that $\nabla \mathcal{E}(\boldsymbol{y}) = \boldsymbol{z}$. Let $\boldsymbol{y}(\boldsymbol{z})$ be a $\boldsymbol{y}$ s.t. $\nabla \mathcal{E}(\boldsymbol{y}) = \boldsymbol{z}$. It can be shown that because $\mathcal{E}$ is strictly convex, $\boldsymbol{y}(\boldsymbol{z})$ is unique.

The function $\boldsymbol{y} \mapsto \langle \boldsymbol{z}, \boldsymbol{y} \rangle - \mathcal{E}(\boldsymbol{y})$ is concave in $\boldsymbol{y}$ and has gradient $\boldsymbol{z} - \nabla \mathcal{E}(\boldsymbol{y})$ and is hence maximized at $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{z})$. This follows because we know a differentiable convex function is minimized when its gradient is zero and so a differentiable concave function is maximized when its gradient is zero.

Then, using the product rule and composition rule of derivatives,

$$\begin{aligned}
\nabla \mathcal{E}^*(\boldsymbol{z}) &= \nabla_z \left(\langle \boldsymbol{z}, \boldsymbol{y}(\boldsymbol{z}) \rangle - \mathcal{E}(\boldsymbol{y}(\boldsymbol{z})) \right) \\
&= \boldsymbol{y}(\boldsymbol{z}) + \left(\nabla_z (\boldsymbol{y}(\boldsymbol{z})^\top) \right) \boldsymbol{z} - \left(\nabla_z (\boldsymbol{y}(\boldsymbol{z})^\top) \right) \underbrace{\nabla_y \mathcal{E}(\boldsymbol{y}(\boldsymbol{z}))}_{=\boldsymbol{z}} \\
&= \boldsymbol{y}(\boldsymbol{z})
\end{aligned}$$

Thus we have $\nabla_y \mathcal{E}(\boldsymbol{y}(\boldsymbol{z})) = \boldsymbol{z}$ and $\nabla_z \mathcal{E}^*(\boldsymbol{z}) = \boldsymbol{y}(\boldsymbol{z})$. Combining the two, we have $\nabla \mathcal{E}(\nabla \mathcal{E}^*(\boldsymbol{z})) = \boldsymbol{z}$.

We can also see that for any $\boldsymbol{y}$, there exists a $\boldsymbol{z}$ such that $\nabla_z \mathcal{E}^*(\boldsymbol{z}) = \boldsymbol{y}$, namely, this is attained by $\boldsymbol{z} = \nabla_y \mathcal{E}(\boldsymbol{y})$. Thus, $\nabla \mathcal{E}^*(\nabla \mathcal{E}(\boldsymbol{y})) = \boldsymbol{y}$.

Part 2. Observe that

$$\mathcal{E}^{**}(\boldsymbol{u}) = \sup_{\boldsymbol{z} \in \mathbb{R}^n} \langle \boldsymbol{u}, \boldsymbol{z} \rangle - \mathcal{E}^*(\boldsymbol{z})$$

and let $\boldsymbol{z}(\boldsymbol{u})$ denote the $\boldsymbol{z}$ obtaining the supremum, in the above program. We then have $\boldsymbol{u} = \nabla \mathcal{E}^*(\boldsymbol{z}(\boldsymbol{u}))$. Letting $\boldsymbol{y}(\boldsymbol{z})$ be defined as in Part 1, we get $\boldsymbol{y}(\boldsymbol{z}(\boldsymbol{u})) = \nabla_z \mathcal{E}^*(\boldsymbol{z}(\boldsymbol{u})) = \boldsymbol{u}$

$$\mathcal{E}^{**}(\boldsymbol{u}) = \langle \boldsymbol{u}, \boldsymbol{z}(\boldsymbol{u}) \rangle - (\langle \boldsymbol{z}(\boldsymbol{u}), \boldsymbol{y}(\boldsymbol{z}(\boldsymbol{u})) \rangle - \mathcal{E}(\boldsymbol{y}(\boldsymbol{z}(\boldsymbol{u})))) = \mathcal{E}(\boldsymbol{u}).$$

Part 3. Now we add two infinitesimals $\boldsymbol{\tau}$ and $\boldsymbol{\delta}$ to $\boldsymbol{z}$ and $\boldsymbol{y}$ respectively s.t.

$$\nabla_z \mathcal{E}^*(\boldsymbol{z} + \boldsymbol{\tau}) = \boldsymbol{y} + \boldsymbol{\delta}, \quad \nabla_y \mathcal{E}(\boldsymbol{y} + \boldsymbol{\delta}) = \boldsymbol{z} + \boldsymbol{\tau}.$$

Then,

$$\nabla_y \mathcal{E}(\boldsymbol{y} + \boldsymbol{\delta}) - \nabla_y \mathcal{E}(\boldsymbol{y}) = \boldsymbol{\tau}, \quad \nabla_z \mathcal{E}^*(\boldsymbol{z} + \boldsymbol{\tau}) - \nabla_z \mathcal{E}^*(\boldsymbol{z}) = \boldsymbol{\delta}.$$

Since $\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y})$ measures the change of $\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})$ when $\boldsymbol{y}$ changes by an infinitesimal $\boldsymbol{\delta}$, then

$$\begin{aligned}
&\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}) - \boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}) \approx \boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y})\boldsymbol{\delta} \\
\iff &\boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})\left(\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}) - \boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})\right) \approx \boldsymbol{\delta} \\
\iff &\boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})\boldsymbol{\tau} \approx \boldsymbol{\delta} = \boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{z}+\boldsymbol{\tau}) - \boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{z}) \\
\iff &\boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})\boldsymbol{\tau} \approx \boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{z}+\boldsymbol{\tau}) - \boldsymbol{\nabla}\mathcal{E}^*(\boldsymbol{z}) \qquad (16.6)
\end{aligned}$$

Similarly,

$$\boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{z})\boldsymbol{\tau} \approx \boldsymbol{\nabla}_{\boldsymbol{z}}\mathcal{E}^*(\boldsymbol{z}+\boldsymbol{\tau}) - \boldsymbol{\nabla}_{\boldsymbol{z}}\mathcal{E}^*(\boldsymbol{z}) \qquad (16.7)$$

Comparing (16.6) and (16.7), it is easy to see

$$\boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{z}) = \boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y}) \iff \boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})) = \boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y}).$$

□

Remark 16.2.4. Theorem 16.2.3 can be generalized to show that the Fenchel conjugate has similar nice properties under much more general conditions, e.g. see [BV04].

16.3 Newton's Method

16.3.1 Warm-up: Quadratic Optimization

First, let us play with a toy example, minimizing a quadratic function

$$\mathcal{E}(\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{y} + \boldsymbol{b}^\top \boldsymbol{y} + \boldsymbol{c}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is positive definite. By setting the gradient w.r.t. $\boldsymbol{y}$ to zero,

$$\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}) = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{b} = \boldsymbol{0},$$

we obtain the global minimizer

$$\boldsymbol{y}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}.$$

To make it more like gradient descent, let us start at some "guess" point $\boldsymbol{y}$ and take a step $\boldsymbol{\delta}$ to move to the new point $\boldsymbol{y}+\boldsymbol{\delta}$. Then we try to minimize $\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta})$ by setting the gradient w.r.t. $\boldsymbol{\delta}$ to zero,

$$\begin{aligned}
\boldsymbol{\nabla}_{\boldsymbol{\delta}}\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}) = \boldsymbol{A}(\boldsymbol{y}+\boldsymbol{\delta}) + \boldsymbol{b} &= \boldsymbol{0} \\
\boldsymbol{\delta} &= -\boldsymbol{y} - \boldsymbol{A}^{-1}\boldsymbol{b} \\
\boldsymbol{y}+\boldsymbol{\delta} &= -\boldsymbol{A}^{-1}\boldsymbol{b}
\end{aligned}$$

This gives us exactly global minimizer in just one step. However, the situation changes when the function is not quadratic anymore and thus we do not have a constant Hessian. But taking a step which tries to set the gradient to zero might still be a good idea.

16.3.2 K-stable Hessian

Next, consider a convex function $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ whose Hessian is "nearly constant". Recall the Hessian $\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y})$ aka $\boldsymbol{\nabla}^2\mathcal{E}(\boldsymbol{y})$ at a point $\boldsymbol{y}$ is just a matrix of pairwise 2nd order partial derivatives $\frac{\partial^2\mathcal{E}(\boldsymbol{y})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_j}$. We say $\mathcal{E}$ has a k-stable Hessian if there exists a constant matrix $\boldsymbol{A}$ s.t. for all $\boldsymbol{y}$

$$\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}) \approx_K \boldsymbol{A} \iff \frac{1}{1+K}\boldsymbol{A} \preceq \boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}) \preceq (1+K)\boldsymbol{A}.$$

Note that we just require the existence of $\boldsymbol{A}$ and do not assume we know $\boldsymbol{A}$. Then a natural question is to ask what convergence rate can be achieved if we take a gradient step "guided" by the Hessian, which is called a "Newton step". Such method is also known as the 2nd order method. Note that this is very similar to preconditioning.

Now, let us make our setting precise. We want to minimize a convex function $\mathcal{E}$ with k-stable Hessian $\boldsymbol{A} \succ \boldsymbol{0}$. And $\boldsymbol{y}^*$ is a global minimizer of $\mathcal{E}$. Start from some initial point $\boldsymbol{y}_0$. The update rule is

$$\boldsymbol{y}_{i+1} = \boldsymbol{y}_i - \alpha \cdot \boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y}_i)\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}_i),$$

where α is the step size and it will be decided later.

Theorem 16.3.1. *$\mathcal{E}(\boldsymbol{y}_k) - \mathcal{E}(\boldsymbol{y}^*) \le \epsilon\,(\mathcal{E}(\boldsymbol{y}_0) - \mathcal{E}(\boldsymbol{y}^*))$ when $k > (K+1)^6 \log(1/\epsilon)$.*

Proof. By Taylor's theorem, there exists $\tilde{\boldsymbol{y}} \in [\boldsymbol{y}, \boldsymbol{y}+\boldsymbol{\delta}]$ s.t.

$$\begin{aligned}\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}) &= \mathcal{E}(\boldsymbol{y}) + \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top\boldsymbol{H}_{\mathcal{E}}(\tilde{\boldsymbol{y}})\boldsymbol{\delta} \\ &\le \underbrace{\mathcal{E}(\boldsymbol{y}) + \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top\boldsymbol{\delta} + \frac{(K+1)^2}{2}\boldsymbol{\delta}^\top\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y})\boldsymbol{\delta}}_{=:f(\boldsymbol{\delta})} \qquad (16.8)\end{aligned}$$

where the inequality comes from the K-stability of the Hessian,

$$\boldsymbol{H}_{\mathcal{E}}(\tilde{\boldsymbol{y}}) \preceq (1+K)\boldsymbol{A} \preceq (1+K)^2\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y}).$$

Observe that $f(\boldsymbol{\delta})$ is a convex quadratic function in $\boldsymbol{\delta}$. By minimizing it, or equivalently setting $\boldsymbol{\nabla}_{\boldsymbol{\delta}} f(\boldsymbol{\delta}^*) = \boldsymbol{0}$, we get

$$\boldsymbol{\delta}^* = -\frac{1}{(K+1)^2}\boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}) \qquad (16.9)$$

Here, the step size α is equal to $(K+1)^{-2}$. Then, plugging (16.9) into (16.8),

$$\begin{aligned}\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}^*) &\le \mathcal{E}(\boldsymbol{y}) - \frac{1}{2(K+1)^2}\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})^\top\boldsymbol{H}_{\mathcal{E}}^{-1}(\boldsymbol{y})\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}) \\ &\le \mathcal{E}(\boldsymbol{y}) - \frac{1}{2(K+1)^3}\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})^\top\boldsymbol{A}^{-1}\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y}) \\ &\qquad \text{(subtract } \mathcal{E}(\boldsymbol{y}^*) \text{ on both sides)} \\ \mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}^*) - \mathcal{E}(\boldsymbol{y}^*) &\le \mathcal{E}(\boldsymbol{y}) - \mathcal{E}(\boldsymbol{y}^*) - \frac{1}{2(K+1)^3}\underbrace{\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})^\top\boldsymbol{A}^{-1}\boldsymbol{\nabla}_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{y})}_{=:\sigma} \qquad (16.10)\end{aligned}$$

where the second inequality is due to K-stability of the inverse Hessian,

$$\frac{1}{1+K}\boldsymbol{A}^{-1} \preceq \boldsymbol{H}_{\mathcal{E}}(\boldsymbol{y})^{-1} \preceq (1+K)\boldsymbol{A}^{-1}.$$

Meanwhile, by convexity, we also have

$$\mathcal{E}(\boldsymbol{y}) - \mathcal{E}(\boldsymbol{y}^*) \leq \underbrace{(\boldsymbol{y}-\boldsymbol{y}^*)^\top \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})}_{=:\gamma}$$

Next, our task is reduced to comparing σ and γ.

We define $\boldsymbol{z}_s = \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*) + s(\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}) - \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y}^*))$ and then $\mathrm{d}\boldsymbol{z}_s = \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})\mathrm{d}s$. Using *Theorem 16.2.3*, we have

$$\boldsymbol{y} - \boldsymbol{y}^* = \int_0^1 \boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{z}_s)\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})\mathrm{d}s.$$

Then,

$$\begin{aligned}
\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top(\boldsymbol{y}-\boldsymbol{y}^*) &= \int_0^1 \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top \boldsymbol{H}_{\mathcal{E}^*}(\boldsymbol{z}_s)\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})\mathrm{d}s \\
&\leq (K+1)\int_0^1 \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top \boldsymbol{A}^{-1}\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})\mathrm{d}s \\
&\leq (K+1)\sigma
\end{aligned} \tag{16.11}$$

Combining (16.10) and (16.11) yields

$$\mathcal{E}(\boldsymbol{y}+\boldsymbol{\delta}^*) - \mathcal{E}(\boldsymbol{y}^*) \leq (\mathcal{E}(\boldsymbol{y}) - \mathcal{E}(\boldsymbol{y}^*))\left(1 - \frac{1}{2(K+1)^4}\right).$$

□

Remark 16.3.2. The basic idea of relating σ and γ in the above proof is writing the same quantity, $\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{y})^\top(\boldsymbol{y}-\boldsymbol{y}^*)$, as two integrations along different lines. $(K+1)^4$ can be reduced to $(K+1)^2$ and even to $(K+1)$ with more care. In some settings, Newton's method converges in $\log\log(1/\epsilon)$ steps.

16.3.3 Linearly Constrained Newton's Method

Let us apply Newton's method to convex optimization programs with only linear constraints,

$$\begin{aligned}
\min_{\boldsymbol{f}\in\mathbb{R}^m} \quad & \mathcal{E}(\boldsymbol{f}) \\
\text{s.t.} \quad & \boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}
\end{aligned}$$

where $\mathcal{E}: \mathbb{R}^m \to \mathbb{R}$ is a convex function and $\boldsymbol{B} \in \mathbb{R}^{n\times m}$. Wlog, let $\boldsymbol{d} = \boldsymbol{0}$, since otherwise we can equivalently deal the following program with $\boldsymbol{B}\boldsymbol{f}_0 = \boldsymbol{d}$,

$$\begin{aligned}
\min_{\boldsymbol{\rho}\in\mathbb{R}^m} \quad & \mathcal{E}(\boldsymbol{f}_0 + \boldsymbol{\rho}) \\
\text{s.t.} \quad & \boldsymbol{B}\boldsymbol{\rho} = \boldsymbol{0}
\end{aligned}$$

It is useful to think of the variable $\boldsymbol{f} \in \mathbb{R}^m$ as a flow in a graph. Define $C := \{\boldsymbol{f} : \boldsymbol{B}\boldsymbol{f} = \boldsymbol{0}\}$ which is the kernel space of $\boldsymbol{B}$. C is also called the "cycle space" as it is the set of cycle flows when treating $\boldsymbol{f}$ as flows.

Analyzing the convergence of Newton's method with linear constraints. It is not immediately obvious, but our analysis of Newton step's and their convergence on objectives with a K-stable Hessian carries over directly to linearly constrained convex optimization problems. We will only sketch a proof of this. Firstly, we should notice that instead of thinking of our objective function $\mathcal{E}$ as defined on $\mathbb{R}^m$ and then constrained to inputs $\boldsymbol{f} \in C$, we can think of a new function $\hat{\mathcal{E}} : C \to \mathbb{R}$ defined such that for $\boldsymbol{f} \in C$ we have $\hat{\mathcal{E}}(\boldsymbol{f}) = \mathcal{E}(\boldsymbol{f})$. But, C is a linear subspace and is isomorphic[1] to $\mathbb{R}^{\dim(C)}$. This means that our previous analysis can be directly applied, if we can compute the gradient and Hessian of the function viewed as an *unconstrained* function on C (or equivalently $\mathbb{R}^{\dim(C)}$). We now have two important questions to answer:

1. What does the gradient and Hessian, and hence Newton steps, of $\hat{\mathcal{E}}(\boldsymbol{f})$ look like?
2. Does the K-stability of the Hessian of $\mathcal{E}$ carry over to the function $\hat{\mathcal{E}}$?

The gradient and Hessian of $\hat{\mathcal{E}}$ should live in $\mathbb{R}^{\dim(C)}$ and $\mathbb{R}^{\dim(C)\times\dim(C)}$ respectively. Let Π_C be the orthogonal projection matrix onto $\boldsymbol{C}$, meaning Π_C is symmetric and $\Pi_C\boldsymbol{\delta} = \boldsymbol{\delta}$ for any $\boldsymbol{\delta} \in C$. Given any $\boldsymbol{f} \in C$, add to it an infinitesimal $\boldsymbol{\delta} \in C$, then

$$\begin{aligned}\hat{\mathcal{E}}(\boldsymbol{f}+\boldsymbol{\delta}) = \mathcal{E}(\boldsymbol{f}+\boldsymbol{\delta}) &\approx \mathcal{E}(\boldsymbol{f}) + \langle \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{f}), \boldsymbol{\delta}\rangle \\ &= \mathcal{E}(\boldsymbol{f}) + \langle \boldsymbol{\nabla}\mathcal{E}(\boldsymbol{f}), \Pi_C\boldsymbol{\delta}\rangle \\ &= \mathcal{E}(\boldsymbol{f}) + \langle \Pi_C\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{f}), \boldsymbol{\delta}\rangle\end{aligned}$$

From this, we can deduce that the gradient of $\hat{\mathcal{E}}$ at a point $\boldsymbol{f} \in C$ is essentially equal (up to a fixed linear transformation independent of $\boldsymbol{f}$) to the projection of gradient of $\boldsymbol{\nabla}\mathcal{E}$ at $\boldsymbol{f}$ onto the subspace C. Similarly,

$$\hat{\mathcal{E}}(\boldsymbol{f}+\boldsymbol{\delta}) = \mathcal{E}(\boldsymbol{f}+\boldsymbol{\delta}) \approx \mathcal{E}(\boldsymbol{f}) + \langle \Pi_C\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{f}), \boldsymbol{\delta}\rangle + \frac{1}{2}\langle \boldsymbol{\delta}, \Pi_C\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{f})\Pi_C\boldsymbol{\delta}\rangle$$

Again from this, we can deduce that the Hessian of $\hat{\mathcal{E}}$ at a point $\boldsymbol{f} \in C$ is essentially equal (again up to a fixed linear transformation independent of $\boldsymbol{f}$) to the matrix $\Pi_C\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{f})\Pi_C$. Note that $\boldsymbol{X} \preceq \boldsymbol{Y}$ implies $\Pi_C\boldsymbol{X}\Pi_C \preceq \Pi_C\boldsymbol{Y}\Pi_C$, and from this we can see that the Hessian of $\hat{\mathcal{E}}$ is K-stable if the Hessian of $\mathcal{E}$ is. Also note that we were not terribly formal in the discussion above. We can be more precise by replacing Π_C with a linear map from $\mathbb{R}^{\dim(C)}$ to $\mathbb{R}^m$ which maps any vector in $\mathbb{R}^{\dim(C)}$ to a vector in C and then going through a similar chain of reasoning.

[1] You don't need to know the formal definition of isomorphism on vector spaces. In this context, it means equivalent up to a transformation by an invertible matrix. In fact in our case, the isomorphism is given by an orthonormal matrix.

What is a Newton step $\boldsymbol{\delta}^*$ w.r.t. $\hat{\mathcal{E}}$? It turns out that for actually computing the Newton step, it is easier to think again of $\mathcal{E}$ with a constraint that the Newton step must lie in the subspace C. One can show that this is equivalent to the Newton step of $\hat{\mathcal{E}}$, but we omit this.

In the constrained view, $\boldsymbol{\delta}^*$ should be a minimizer of

$$\begin{aligned}
&\min_{\substack{\boldsymbol{\delta}\in\mathbb{R}^m \\ \boldsymbol{B}\boldsymbol{\delta}=\mathbf{0}}} \langle \underbrace{\boldsymbol{\nabla}\mathcal{E}(\boldsymbol{f})}_{=:\boldsymbol{g}}, \boldsymbol{\delta}\rangle + \frac{1}{2}\langle \boldsymbol{\delta}, \underbrace{\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{f})}_{=:\boldsymbol{H}}\boldsymbol{\delta}\rangle \\
&\quad \text{(Lagrange duality)} \\
\iff &\max_{\boldsymbol{x}\in\mathbb{R}^n}\min_{\boldsymbol{\delta}\in\mathbb{R}^m} \underbrace{\langle \boldsymbol{g}, \boldsymbol{\delta}\rangle + \frac{1}{2}\langle \boldsymbol{\delta}, \boldsymbol{H}\boldsymbol{\delta}\rangle - \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{\delta}}_{\text{Lagrangian } L(\boldsymbol{\delta},\boldsymbol{x})}
\end{aligned} \tag{16.12}$$

Applying the KKT optimality conditions, one has

$$\begin{aligned}
\boldsymbol{B}\boldsymbol{\delta} &= \mathbf{0}, \\
\boldsymbol{\nabla}_{\boldsymbol{\delta}} L(\boldsymbol{\delta}, \boldsymbol{x}) = \boldsymbol{g} + \boldsymbol{H}\boldsymbol{\delta} - \boldsymbol{B}^\top \boldsymbol{x} &= \mathbf{0},
\end{aligned}$$

from which we get

$$\begin{aligned}
\boldsymbol{\delta} + \boldsymbol{H}^{-1}\boldsymbol{g} &= \boldsymbol{H}^{-1}\boldsymbol{B}^\top \boldsymbol{x} \\
\underbrace{\boldsymbol{B}\boldsymbol{\delta}}_{=\mathbf{0}} + \boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{g} &= \boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{B}^\top \boldsymbol{x} \\
\boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{g} &= \underbrace{\boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{B}^\top}_{=:\boldsymbol{L}} \boldsymbol{x}
\end{aligned}$$

Finally, the solutions to (16.12) are

$$\begin{cases}
\boldsymbol{x}^* &= \boldsymbol{L}^{-1}\boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{g} \\
\boldsymbol{\delta}^* &= -\boldsymbol{H}^{-1}\boldsymbol{g} + \boldsymbol{H}^{-1}\boldsymbol{B}^\top \boldsymbol{x}^*
\end{cases}$$

It is easy to verify that $\boldsymbol{B}\boldsymbol{\delta}^* = \mathbf{0}$. Thus, our update rule is $\boldsymbol{f}_{i+1} = \boldsymbol{f}_i + \boldsymbol{\delta}^*$. And we have the following convergence result.

Theorem 16.3.3. $\hat{\mathcal{E}}(\boldsymbol{f}_k) - \hat{\mathcal{E}}(\boldsymbol{f}^*) \le \epsilon \cdot \left(\hat{\mathcal{E}}(\boldsymbol{f}_0) - \hat{\mathcal{E}}(\boldsymbol{f}^*)\right)$ *when* $k > 2(K+1)^4 \log(1/\epsilon)$.

Remark 16.3.4. Note if $\mathcal{E}(\boldsymbol{f}) = \sum_{i=1}^m \mathcal{E}_i(\boldsymbol{f}(i))$, then $\boldsymbol{H}_{\mathcal{E}}(\boldsymbol{f})$ is diagonal. Thus, $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{H}^{-1}\boldsymbol{B}^\top$ is indeed a Laplacian provided that $\boldsymbol{B}$ is an incidence matrix. Therefore, the linear equations we need to solve to apply Newton's method in a network flow setting are Laplacians, which means we can solve them very quickly.

Chapter 17

Interior Point Methods for Maximum Flow

Background and Notation

In this chapter, we'll learn about interior point methods for solving maximum flow, which is a rich and active area of research [DS08, Mad13, LS20b, LS20a].

We're going to frequently need to refer to vectors arising from elementwise operations combining other vectors.

To that end, given two vector $\boldsymbol{a} \in \mathbb{R}^m$, and $\boldsymbol{b} \in \mathbb{R}^m$, we will use $\overrightarrow{(\boldsymbol{a}(i)\boldsymbol{b}(i))}$ to denote the vector $\boldsymbol{z}$ with $\boldsymbol{z}(i) = \boldsymbol{a}(i)\boldsymbol{b}(i)$ and so on.

Throughout this chapter, when we are working in the context of some given graph G with vertices V and edges E, we will let $m = |E|$ and $n = |V|$.

The plots in this chapter were made using Mathematica, which is available to ETH students for download through the ETH IT Shop.

17.1 An Interior Point Method

The Maximum Flow problem in undirected graphs.

$$\begin{aligned} \max_{\boldsymbol{f} \in \mathbb{R}^E} \quad & F \\ \text{s.t. } & \boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st} \\ & -\boldsymbol{c} \leq \boldsymbol{f} \leq \boldsymbol{c} \end{aligned} \tag{17.1}$$

"The Undirected Maximum Flow Problem"

We use $\text{val}(\boldsymbol{f})$ to denote F when $B\boldsymbol{f} = F\boldsymbol{b}_{st}$.

As we develop algorithms for this problem, we will assume that we know the maximum flow value F^*. Let $\boldsymbol{f}^*$ denote some maximum flow, i.e. a flow with $-\boldsymbol{c} \leq \boldsymbol{f} \leq \boldsymbol{c}$ can $\text{val}(\boldsymbol{f}^*) = F^*$.

In general, an a lower bound $F \leq F^*$ will allow us to find a flow with value F, and because of this, we can use a binary search to approximate F^*.

17.1.1 A Barrier Function and an Algorithm

$$V(\boldsymbol{f}) = \sum_e -\log(\boldsymbol{c}(e) - \boldsymbol{f}(e)) - \log(\boldsymbol{c}(e) + \boldsymbol{f}(e))$$

We assume the optimal value of Program (17.1) is F^*. Then for a given $0 \leq \alpha < 1$ we define a program

$$\begin{aligned} \min_{\boldsymbol{f} \in \mathbb{R}^E} \quad & V(\boldsymbol{f}) \\ \text{s.t. } & \boldsymbol{B}\boldsymbol{f} = \alpha F^* \boldsymbol{b}_{st} \end{aligned} \tag{17.2}$$

"The Barrier Problem"

This problem makes sense for any $0 \leq \alpha < 1$. When $\alpha = 0$, we are not routing any flow yet. This will be our starting point. For any $0 \leq \alpha < 1$, the scaled-down maximum flow $\alpha \boldsymbol{f}^*$ strictly satisfies the capacities $-\boldsymbol{c} < \alpha \boldsymbol{f}^* < \boldsymbol{c}$, and $\boldsymbol{B}\alpha\boldsymbol{f}^* = \alpha F^* \boldsymbol{b}_{st}$. Hence $\alpha \boldsymbol{f}^*$ is a feasible flow for this value of α and hence $V(\alpha \boldsymbol{f}^*) < \infty$ and so the optimal flow for the Barrier Problem at this α must also have objective value strictly below ∞, and hence in turn strictly satisfy the capacity constraints. Thus, if we can find the optimal flow for Program (17.2) for $\alpha = 1 - \epsilon$, we will have a feasible flow with Program (17.1), the Undirected Maximum Flow Problem, routing $(1 - \epsilon)F^*$. This is how we will develop an algorithm for computing the maximum flow.

Program (17.2) has the Lagrangian

$$\mathcal{L}(\boldsymbol{f}, \boldsymbol{x}) = V(\boldsymbol{f}) + \boldsymbol{x}^\top (\alpha F^* \boldsymbol{b}_{st} - \boldsymbol{B}\boldsymbol{f})$$

And we have optimality when

$$\boldsymbol{B}\boldsymbol{f} = \alpha F^* \boldsymbol{b}_{st} \text{ and } -\boldsymbol{c} \leq \boldsymbol{f} \leq \boldsymbol{c} \tag{17.3}$$

"Barrier feasibility"

and $\boldsymbol{\nabla}_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f}, \boldsymbol{x}) = \boldsymbol{0}$, i.e.

$$\boldsymbol{\nabla} V(\boldsymbol{f}) = \boldsymbol{B}^\top \boldsymbol{x} \tag{17.4}$$

"Barrier Lagrangian gradient optimality"

Let $\boldsymbol{f}^*_\alpha$ denote the optimal solution to Problem 17.2 for a given $0 \leq \alpha < 1$, and let $\boldsymbol{x}^*_\alpha$ be optimal dual voltages such that $\boldsymbol{\nabla} V(\boldsymbol{f}^*_\alpha) = \boldsymbol{B}^\top \boldsymbol{x}^*_\alpha$.

It turns out that, if we have a solution $\boldsymbol{f}^*_\alpha$ to this problem for some $\alpha < 1$, then we can find a solution $\boldsymbol{f}_{\alpha+\alpha'}$ for some $\alpha' < 1 - \alpha$. And, we can compute $\boldsymbol{f}_{\alpha+\alpha'}$ using a small number of Newton steps, each of which will only require a Laplacian linear equation solve, and hence is computable in $\widetilde{O}(m)$ time. Concretely, for any $0 \leq \alpha < 1$, given the optimal flow at this α, we will be able to compute the optimal flow at $\alpha_{\text{new}} = \alpha + (1-\alpha)\frac{1}{150\sqrt{m}}$. This means that after $T = 150\sqrt{m}\log(1/\epsilon)$ updates, we have a solution for $\alpha \geq 1 - \epsilon$.

We can state the update problem as

$$\begin{aligned} \min_{\boldsymbol{\delta} \in \mathbb{R}^E} \quad & V(\boldsymbol{\delta} + \boldsymbol{f}) \\ \text{s.t. } \boldsymbol{B}\boldsymbol{\delta} &= \alpha' F^* \boldsymbol{b}_{st} \end{aligned} \tag{17.5}$$

"The Update Problem"

17.1.2 Updates using Divergence

It turns out that for the purposes of analysis, it will be useful to ensure that our "Update Problem" uses an objective function that is minimized at $\boldsymbol{\delta} = \mathbf{0}$.

This leads to a variant of the Update Problem, which we call the "Divergence Update Problem". We obtain our new problem by switching from $V(\boldsymbol{\delta} + \boldsymbol{f})$ as our objective to $V(\boldsymbol{\delta}+\boldsymbol{f}) - (V(\boldsymbol{f}) + \langle \boldsymbol{\nabla} V(\boldsymbol{f}), \boldsymbol{\delta} \rangle)$ as our objective, and this is called the *divergece* of V w.r.t. $\boldsymbol{\delta}$ *based* at $\boldsymbol{f}$.

$$\begin{aligned} \min_{\boldsymbol{\delta} \in \mathbb{R}^E} \quad & V(\boldsymbol{\delta} + \boldsymbol{f}) - (V(\boldsymbol{f}) + \langle \boldsymbol{\nabla} V(\boldsymbol{f}), \boldsymbol{\delta} \rangle) \\ \text{s.t. } \boldsymbol{B}\boldsymbol{\delta} &= \alpha' F^* \boldsymbol{b}_{st} \end{aligned} \tag{17.6}$$

"The Divergence Update Problem"

Now, for any flow $\boldsymbol{\delta}$ such that $\boldsymbol{B}\boldsymbol{\delta} = \alpha' F^* \boldsymbol{b}_{st}$, using the Lagrangian gradient condition (17.4), we have $\langle \boldsymbol{\nabla} V(\boldsymbol{f}^*_\alpha), \boldsymbol{\delta} \rangle = \langle \boldsymbol{x}^*_\alpha, \alpha' F^* \boldsymbol{b}_{st} \rangle$. Hence, for such $\boldsymbol{\delta}$, we have

$$V(\boldsymbol{\delta} + \boldsymbol{f}^*_\alpha) - (V(\boldsymbol{f}^*_\alpha) + \langle \boldsymbol{\nabla} V(\boldsymbol{f}^*_\alpha), \boldsymbol{\delta} \rangle) = V(\boldsymbol{\delta} + \boldsymbol{f}^*_\alpha) - (V(\boldsymbol{f}^*_\alpha) + \langle \boldsymbol{x}^*_\alpha, \alpha' F^* \boldsymbol{b}_{st} \rangle)$$

We conclude that the objectives of the Update Problem (17.5) and the Divergence Update Problem (17.6) have the same minimizer, which we denote $\boldsymbol{\delta}^*_{\alpha'}$, although, to be precise, it is also a function of α.

Thus $\boldsymbol{f}^*_\alpha + \boldsymbol{\delta}^*_{\alpha'}$ is optimal for the optimization problem

$$\begin{aligned} \min_{\boldsymbol{f} \in \mathbb{R}^E} \quad & V(\boldsymbol{f}) \\ \text{s.t. } \boldsymbol{B}\boldsymbol{f} &= (\alpha + \alpha') F^* \boldsymbol{b}_{st} \end{aligned} \tag{17.7}$$

Lemma 17.1.1. *Suppose $\boldsymbol{f}^*_\alpha$ is the minimizer of Problem* (17.2) *(the Barrier Problem with parameter α) and $\boldsymbol{\delta}^*_{\alpha'}$ is the minimizer of Problem* (17.6) *(the Update Problem with parameters $\boldsymbol{f}^*_\alpha$ and α'), then $\boldsymbol{f}^*_\alpha + \boldsymbol{\delta}^*_{\alpha'}$ is optimal for Problem* (17.2) *with parameter $\alpha + \alpha'$ (i.e. a new instance of the Barrier problem).*

Algorithm 22: Interior Point Method

```
f ← 0;
α ← 0;
while α < 1 − ε do
    a′ ← (1−α)/(20√m);
    Compute δ, the minimizer of Problem (17.6);
    Let f ← f + δ and α ← α + α′;
end
return f
```

Pseudotheorem 17.1.1. *Let $\boldsymbol{f}$ be the minimizer of Problem* (17.2)*. Then, when $a' \leq \frac{1-\alpha}{20\sqrt{m}}$, the minimizer $\boldsymbol{\delta}$ of Problem* (17.7) *can be computed in $\widetilde{O}(m)$ time.*

The key insight in this type of interior point method is that when the update α' is small enough,

Theorem 17.1.2. *Algorithm 22 returns a flow $\boldsymbol{f}$ that is feasible for Problem* (17.1) *in time $\widetilde{O}(m^{1.5}\log(1/\epsilon))$.*

Proof Sketch. First note that for $\alpha = 0$, the minimizer of Problem (17.2) is $\boldsymbol{f} = \mathbf{0}$. The proof now essentially follows by Lemma (17.1.1), and Pseudotheorem 17.1.1. Note that $1-\alpha$ shrinks by a factor $(1 - \frac{1}{20\sqrt{m}})$ in each iteration of the while-loop, and so after $20\sqrt{m}\log(1/\epsilon)$ iterations, we have $1 - \alpha \leq \epsilon$, at which point the loop terminates. To turn this into a formal proof, we need to take care of the fact the proper theorem corresponding to Pseudotheorem 17.1.1 only gives a highly accurate but not exact solution δ to the "Update Problem". But it's possible to show that this is good enough (even though both $\boldsymbol{f}$ and $\boldsymbol{\delta}$ end up not being exactly optimal in each iteration). □

Remark 17.1.3. For the maximum flow problem, when capacities are integral and polynomially bounded, if we choose $\epsilon = m^{-c}$ for some large enough constant c, given a feasible flow with $\text{val}(\boldsymbol{f}) = 1 - \epsilon$, is it possible to compute an exact maximum flow in nearly linear time. Thus Theorem 17.1.2 can also be used to compute an exact maximum flow in $\widetilde{O}(m)$ time, but we omit the proof. The idea is to first round to an almost optimal, feasible integral flow (which requires a non-trivial combinatorial algorithm), and then to recover the exact flow using Ford-Fulkerson. See [Mad13] for details.

Remark 17.1.4. It is possible to reduce an instance of directed maximum flow to an instance of undirected maximum flow in nearly-linear time, in such a way that if we can *exactly* solve the undirected instance, then in nearly-linear time we can recover an exact solution to the directed maximum flow problem. Thus Theorem (17.1.2) can also be used to solve directed maximum flow. We will ask you to develop this reduction in Graded Homework 2.

Remark 17.1.5. For sparse graphs with $m = \widetilde{O}(n)$ and large capacities, this running time is the best known, and improving it is major open problem.

17.1.3 Understanding the Divergence Objective

Note that if $V(x) = -\log(1-x)$, then $D(x) = V(x) - (V(0) + V'(0)x)$.

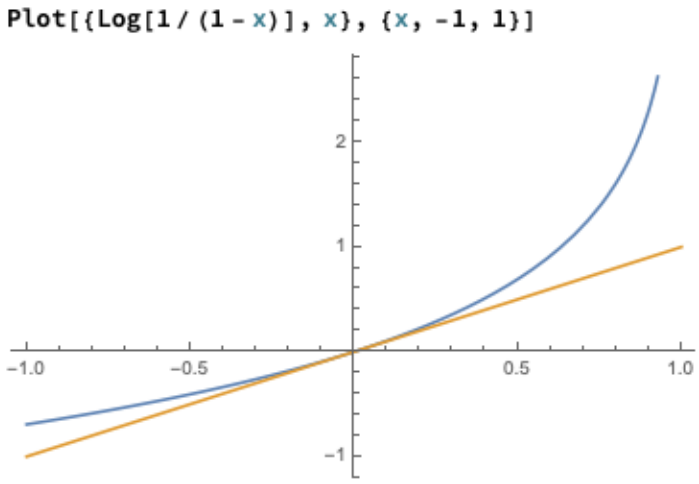


Figure 17.1: Plot showing $V(x) = -\log(1-x)$ and then linear approximation $V(0) + V'(0)x$.

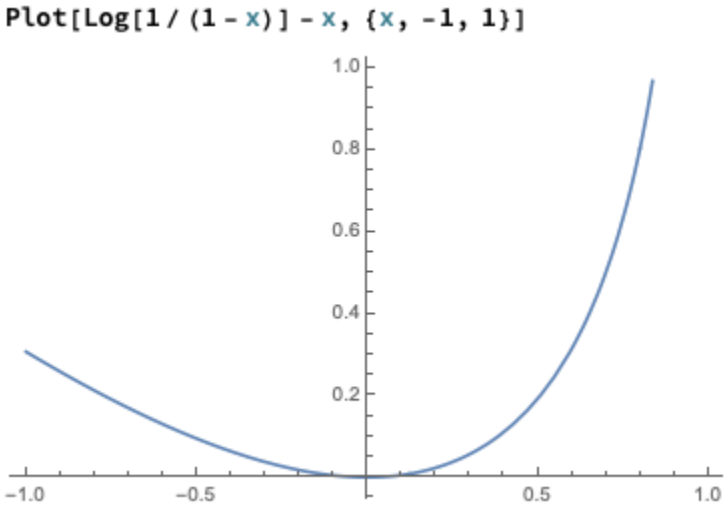


Figure 17.2: Plot showing $D(x) = V(x) - (V(0) + V'(0)x)$.

We let

$$\boldsymbol{c}_+(e) = \boldsymbol{c}(e) - \boldsymbol{f}(e) \text{ and } \boldsymbol{c}_-(e) = \boldsymbol{c}(e) + \boldsymbol{f}(e)$$

So then

$$\begin{aligned} D_V(\boldsymbol{\delta}) &= V(\boldsymbol{\delta}+\boldsymbol{f}) - (V(\boldsymbol{f}) + \langle \boldsymbol{\nabla} V(\boldsymbol{f}), \boldsymbol{\delta}\rangle) \\ &= \sum_e -\log\left(\frac{\boldsymbol{c}(e) - (\boldsymbol{\delta}(e)+\boldsymbol{f}(e))}{\boldsymbol{c}(e)-\boldsymbol{f}(e)}\right) - \frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}(e)-\boldsymbol{f}(e)} \\ &\qquad -\log\left(\frac{\boldsymbol{c}(e) + (\boldsymbol{\delta}(e)+\boldsymbol{f}(e))}{\boldsymbol{c}(e)+\boldsymbol{f}(e)}\right) + \frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}(e)+\boldsymbol{f}(e)} \\ &= \sum_e D\left(\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}(e)-\boldsymbol{f}(e)}\right) + D\left(-\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}(e)+\boldsymbol{f}(e)}\right) \\ &= \sum_e D\left(\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}_+(e)}\right) + D\left(-\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}_-(e)}\right) \end{aligned}$$

Note that we can express Problem (17.6) as

$$\begin{aligned} &\min_{\boldsymbol{\delta}\in\mathbb{R}^E} \quad D_V(\boldsymbol{\delta}) && (17.8) \\ &\text{s.t. } \boldsymbol{B}\boldsymbol{\delta} = \alpha' F^* \boldsymbol{b}_{st} && \text{The Update Problem, restated} \end{aligned}$$

Note that $D_V(\boldsymbol{\delta})$ is strictly convex of over the feasible set, so the argmin is unique.

17.1.4 Quadratically Smoothing Divergence and Local Agreement

$$\tilde{D}_\epsilon(x) = \begin{cases} -\log(1-x) - x & \text{if } |x| \le \epsilon \\ D(\epsilon) + D'(\epsilon)(x-\epsilon) + \frac{D''(\epsilon)}{2}(x-\epsilon)^2 & \text{if } x > \epsilon \\ D(-\epsilon) + D'(-\epsilon)(x+\epsilon) + \frac{D''(-\epsilon)}{2}(x+\epsilon)^2 & \text{if } x < -\epsilon \end{cases}$$

For brevity, we define

$$\tilde{D}(x) = \tilde{D}_{0.1}(x)$$

Lemma 17.1.6.

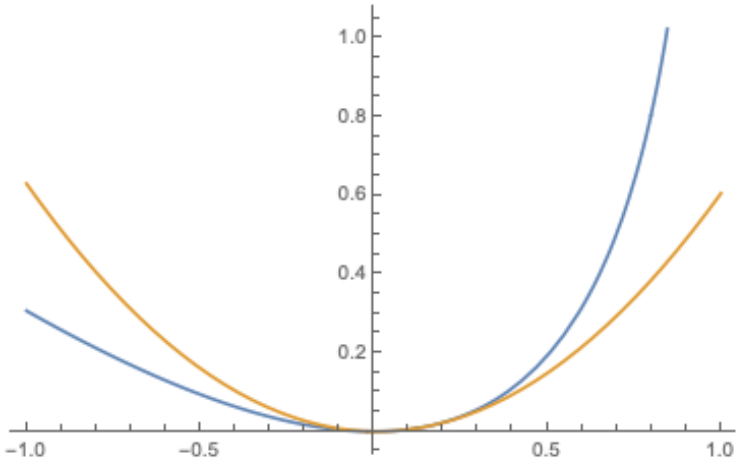
1. $1/2 \le \tilde{D}''(\boldsymbol{x}) \le 2$.
2. *For $x \ge 0$, we have $x/2 \le \tilde{D}'(\boldsymbol{x}) \le 2x$ and $-2x \le \tilde{D}'(-\boldsymbol{x}) \le -x/2$.*
3. $x^2/4 \le \tilde{D}(\boldsymbol{x}) \le x^2$.

What's happening here? We glue together $D(x)$ for small x with its quadratic approximation for $|x| > \epsilon$. For $x > \epsilon$, we "glue in" a Taylor series expansion based at $x = \epsilon$.

```
NumberForm[Series[Log[1 / (1 - x)] - x, {x, 0.1, 2}], 3]
```

```
nberForm=
0.00536 + 0.111 (x - 0.1) + 0.617 (x - 0.1)^2 + O[x - 0.1]^3
```

```
Plot[{Log[1 / (1 - x)] - x, 0.00536 + 0.111 (x - 0.1) + 0.617 (x - 0.1)^2}, {x, -1, 1}]
```




```
Plot[{Log[1 / (1 - x)] - x, 0.00536 + 0.111 (x - 0.1) + 0.617 (x - 0.1)^2}, {x, 0, 0.2}]
```

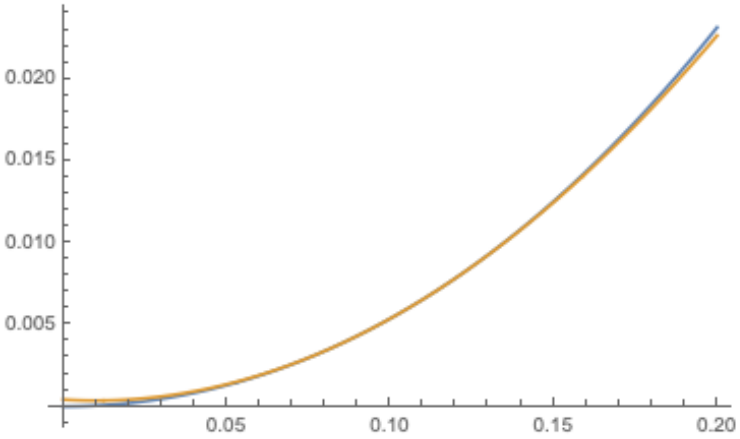


Figure 17.3: Plot showing $D(x) = -\log(1-x)$ and the quadratic approximation based at $x = 0.1$.

We also define

$$\tilde{D}_V(\boldsymbol{\delta}) = \sum_e \tilde{D}\left(\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}_+(e)}\right) + \tilde{D}\left(-\frac{\boldsymbol{\delta}(e)}{\boldsymbol{c}_-(e)}\right)$$

We can now introduce the smoothed optimization problem

$$\min_{\boldsymbol{\delta}\in\mathbb{R}^E} \quad \tilde{D}_V(\boldsymbol{\delta}) \tag{17.9}$$

$$\text{s.t. } \boldsymbol{B}\boldsymbol{\delta} = \alpha' F^* \boldsymbol{b}_{st} \qquad \text{“The Smoothed Update Problem”}$$

Note that $\tilde{D}_V(\boldsymbol{\delta})$ is strictly convex of over the feasible set, so the argmin is unique.

Pseudoclaim 17.1.7. *We can compute the argmin $\boldsymbol{\delta}^*$ of Problem* (17.9)*, the Smoothed Update Problem, using the Newton-Steps for K-stable Hessian convex functions that we saw in the previous chapter, in $\widetilde{O}(m)$ time.*

Sketch of proof. Problem (17.9) fits the class of problems for which we showed in the previous chapter that (appropriately scaled) Newton steps converge. This is true because the Hessian is always a 2-spectral approximation of the Hessian at $\tilde{D}_V(\boldsymbol{\delta}^*)$, as can be shown from Lemma 17.1.6. Because the Hessian of $\tilde{D}_V(\boldsymbol{\delta})$ is diagonal, and the constraints are flow constraints, each Newton step boils down to solving a Laplacian linear system, which can be done to high accuracy $\widetilde{O}(m)$ time. □

Remark 17.1.8. There are three things we need to modify to turn the pseudoclaim into a true claim, addressing the errors arising from both Laplacian solvers and Newton steps:

1. We need to rephrase the claim to so that we only claim $\boldsymbol{\delta}^*$ has been computed to high accuracy, rather than exactly.

2. We need to show that we can construct an initial guess to start off Newton's method $\boldsymbol{\delta}_0$ for which the value $\tilde{D}_V(\boldsymbol{\delta}_0)$ is not too large. (This is easy).

3. We need show that Newton steps converge despite using a Laplacian solver that doesn't give exact solutions, only high accuracy solutions. (Takes a bit of work, but is ultimately not too difficult).

Importantly, to ensure our overall interior point method still works, we also need to show that it converges, even if we're using approximate solutions everywhere. This also takes some work to show, again is not too difficult.

Local Agreement Implies Same Optimum.

Lemma 17.1.9. *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 $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in S} g(\boldsymbol{x})$.*

Proof Sketch. We sketch the proof in the case when both f, g are differentiable: Observe that $\mathbf{0} = \boldsymbol{\nabla} f(\boldsymbol{x}^*) = \boldsymbol{\nabla} g(\boldsymbol{x}^*)$, and hence $g(\boldsymbol{x})$ is also minimized at $\boldsymbol{x}^*$. □

We define

$$\widehat{\boldsymbol{c}}(e) = \min(\boldsymbol{c}_+(e), \boldsymbol{c}_-(e)) \tag{17.10}$$

Lemma 17.1.10. *Suppose $\boldsymbol{\delta}^*$ is the argmin of Problem* (17.9)*, the Smoothed Update Problem, and $\left\| \overrightarrow{(\boldsymbol{\delta}^*(e)/\widehat{\boldsymbol{c}}(e))} \right\|_\infty < 0.1$. Then $\boldsymbol{\delta}^*$ is the argmin of Problem* (17.8)*.*

Proof. We observe that if $\left\|\overrightarrow{(\boldsymbol{\delta}^*(e)/\widehat{\boldsymbol{c}}(e))}\right\|_\infty < 0.1$, then $\tilde{D}_V(\boldsymbol{\delta}^*) = D_V(\boldsymbol{\delta}^*)$, and, for all $\boldsymbol{\tau} \in \mathbb{R}^m$ with norm

$$\left\|\overrightarrow{(\boldsymbol{\tau}(e)/\widehat{\boldsymbol{c}}(e))}\right\|_\infty < 0.1 - \left\|\overrightarrow{(\boldsymbol{\delta}^*(e)/\widehat{\boldsymbol{c}}(e))}\right\|_\infty$$

we have that $\tilde{D}_V(\boldsymbol{\delta}^* + \boldsymbol{\tau}) = D_V(\boldsymbol{\delta}^* + \boldsymbol{\tau})$. Thus $\tilde{D}_V$ and D_V agree on a neighborhood around $\boldsymbol{\delta}^*$ and hence by Lemma 17.1.9, we have that $\boldsymbol{\delta}^*$ is the argmin of Problem (17.8). □

17.1.5 Step size for divergence update

Definition 17.1.11 (*s-t well-conditioned* graph)**.** An undirected, capacitated multi-graph $G = (V, E, \boldsymbol{c})$ with source s and sink t is *s-t well-conditioned* if, letting U denote the maximum edge capacity $U = \|\boldsymbol{c}\|_\infty$, we have at least $\frac{2}{3}m$ multi-edges of capacity U going directly from s to t.

Remark 17.1.12. It is straightforward to make a graph *s-t* well-conditioned. We just add $2m$ new edges of capacity U directly between s and t. Given an exact maximum flow in the new graph, it is trivial to get one in the original graph: Just remove the flow on the new edges.

Definition 17.1.13. Given a *directed* graph $G = (V, E, \boldsymbol{c})$, the *symmetrization* of G is the undirected $\widehat{G} = (V, \widehat{E}, \widehat{\boldsymbol{c}})$ is the undirected graph given by

$$\{a, b\} \in \widehat{E} \text{ if } (a, b) \in E \text{ AND } (b, a) \in E$$

and

$$\widehat{\boldsymbol{c}}(\{a, b\}) = \min(\boldsymbol{c}(a, b), \boldsymbol{c}(b, a)).$$

Note that when $\widehat{G}_{\boldsymbol{f}}$ is the symmetrization of the residual graph $G_{\boldsymbol{f}}$ (which we defined in Chapter 10), then $\widehat{\boldsymbol{c}}$ matches exactly the definition of $\widehat{\boldsymbol{c}}$ in Equation (17.10).

Lemma 17.1.14. *Let G be an undirected, capacitated multi-graph $G = (V, E, \boldsymbol{c})$ which is s-t well-conditioned. Let $\boldsymbol{f}$ be the minimizer of Program* (17.2)*. Let $\widehat{G}_{\boldsymbol{f}}$ be the* symmetrization *of the residual graph $G_{\boldsymbol{f}}$ (in the sense of Lecture 10). Then there exists a flow $\hat{\boldsymbol{\delta}}$ which satisfies $\boldsymbol{B}\hat{\boldsymbol{\delta}} = \frac{1-\alpha}{5}F^*\boldsymbol{b}_{st}$ and is feasible in $\widehat{G}_{\boldsymbol{f}}$. Note that we can also state the feasibility in $\widehat{G}_{\boldsymbol{f}}$ as*

$$\left\|\overrightarrow{\left(\hat{\boldsymbol{\delta}}(e)/\widehat{\boldsymbol{c}}(e)\right)}\right\|_\infty \leq 1$$

Proof. We recall since $\boldsymbol{f}$ is the minimizer of Program (17.2), there exists dual-optimal voltages $\boldsymbol{x}$ such that

$$\boldsymbol{B}^\top \boldsymbol{x} = \boldsymbol{\nabla} V(\boldsymbol{f}) = \overrightarrow{\left(\frac{1}{\boldsymbol{c}(e) - \boldsymbol{f}(e)} - \frac{1}{\boldsymbol{c}(e) + \boldsymbol{f}(e)}\right)}$$

From Lecture 10, we know that there is flow $\bar{\boldsymbol{\delta}}$ that is feasible with respect to the residual graph capacities of the graph $G_{\boldsymbol{f}}$ such that $\boldsymbol{B}\bar{\boldsymbol{\delta}} = (1 - \alpha)F^*\boldsymbol{b}_{st}$. Note when treating $\bar{\boldsymbol{\delta}}$

as an undirected flow, feasibility in the residual graph means that $\overline{\boldsymbol{\delta}}(e) < \boldsymbol{c}(e) - \boldsymbol{f}(e)$ and $-\overline{\boldsymbol{\delta}}(e) < \boldsymbol{c}(e) + \boldsymbol{f}(e)$. Thus,

$$(1-\alpha)F^* \boldsymbol{b}_{st}^\top \boldsymbol{x} = \overline{\boldsymbol{\delta}} \boldsymbol{B}^\top \boldsymbol{x} = \sum_e \frac{\overline{\boldsymbol{\delta}}}{\boldsymbol{c}(e) - \boldsymbol{f}(e)} - \frac{\overline{\boldsymbol{\delta}}}{\boldsymbol{c}(e) + \boldsymbol{f}(e)} \leq m$$

Now, because the graph is s-t well-conditioned, there are at $\frac{2}{3}m$ edges directly from s to t with capacity U and each of these e satisfy by the Lagrangian gradient optimality condition (17.4)

$$\boldsymbol{b}_{st}^\top \boldsymbol{x} = \frac{1}{U - \boldsymbol{f}(e)} - \frac{1}{U + \boldsymbol{f}(e)}$$

Note that $\frac{2}{3}mU \leq F^* \leq mU$ because the graph is s-t well-conditioned. To complete the analysis, we consider three cases.

Case 1: $|\boldsymbol{f}(e)| \leq \frac{2}{3}U$. Then the capacity on each of these edges in the symmetrized residual graph $\widehat{G}_{\boldsymbol{f}}$ is at least $U/3$. As there are $\frac{2}{3}m$ of them, we get that there is a feasible flow in $\widehat{G}_{\boldsymbol{f}}$ of value at least $\frac{2}{9}mU \geq \frac{1}{10}F^*$.

Case 2: $\boldsymbol{f}(e) < -\frac{2}{3}U$. By the gradient condition, we have the same flow on all of the $\frac{2}{3}m$ s-t edges, adding up to at least $\frac{2}{3}mU$ going from t to s. This means that we must have at least $\frac{2}{3}mU$ flow going from s to t via the remaining edges. But, their combined capacity is at most $\frac{1}{3}mU$, so that cannot happen. Thus we can rule out this case entirely.

Case 3: $\boldsymbol{f}(e) > \frac{2}{3}U$. Then

$$\frac{m}{(1-\alpha)F^*} \geq \boldsymbol{b}_{st}^\top \boldsymbol{x} \geq \frac{1}{U - \boldsymbol{f}(e)} - \frac{1}{U + \boldsymbol{f}(e)} \geq \frac{4/5}{U - \boldsymbol{f}(e)}$$

So

$$U - \boldsymbol{f}(e) \geq \frac{4}{5}\frac{(1-\alpha)F^*}{m} \geq \frac{1}{2}(1-\alpha)U$$

In this case, the capacity on each of the $\frac{2}{3}m$ s-t edges with capacity U in G will have capacity $(1-\alpha)U/2$ in $\widehat{G}_{\boldsymbol{f}}$. This guarantees that there is feasible flow in $\widehat{G}_{\boldsymbol{f}}$ of value at least $\frac{1}{3}(1-\alpha)mU \geq \frac{1}{3}(1-\alpha)F^*$. □

Lemma 17.1.15. *Let* $0 < \alpha' \leq \frac{1-\alpha}{150\sqrt{m}}$. *Then the minimizer* $\boldsymbol{\delta}^*$ *of Problem* (17.9) *satisfies* $\left\| \overrightarrow{(\boldsymbol{\delta}^*(e)/\widehat{\boldsymbol{c}}(e))} \right\|_\infty < 0.1$.

Proof. By Lemma 17.1.14, there exists a flow $\hat{\boldsymbol{\delta}}$ which satisfies $\boldsymbol{B}\hat{\boldsymbol{\delta}} = \frac{1-\alpha}{5}F^* \boldsymbol{b}_{st}$ and $\left\| \overrightarrow{\left(\hat{\boldsymbol{\delta}}(e)/\widehat{\boldsymbol{c}}(e)\right)} \right\|_\infty \leq 1$. Hence for any $0 < \alpha' \leq \frac{1-\alpha}{150\sqrt{m}}$, the flow $\tilde{\boldsymbol{\delta}} = \alpha' \frac{5}{1-\alpha}\hat{\boldsymbol{\delta}}$ satisfies

$\boldsymbol{B}\tilde{\boldsymbol{\delta}} = \alpha' F^* \boldsymbol{b}_{st}$ and $\left\| \overrightarrow{\left(\tilde{\boldsymbol{\delta}}(e)/\widehat{\boldsymbol{c}}(e) \right)} \right\|_\infty \leq \frac{1}{30\sqrt{m}}$. This means that

$$\begin{aligned}
\tilde{D}_V(\tilde{\boldsymbol{\delta}}) &= \sum_e \tilde{D}\left(\frac{\tilde{\boldsymbol{\delta}}(e)}{\boldsymbol{c}_+(e)} \right) + \tilde{D}\left(-\frac{\tilde{\boldsymbol{\delta}}(e)}{\boldsymbol{c}_-(e)} \right) \\
&\leq \sum_e 4\left(\frac{\tilde{\boldsymbol{\delta}}(e)}{\boldsymbol{c}_+(e)} \right)^2 + 4\left(-\frac{\tilde{\boldsymbol{\delta}}(e)}{\boldsymbol{c}_-(e)} \right)^2 \\
&\leq \sum_e 8\left(\frac{\tilde{\boldsymbol{\delta}}(e)}{\widehat{\boldsymbol{c}}(e)} \right)^2 \\
&\leq 8/900 < 1/100.
\end{aligned}$$

This then means that the minimizer $\boldsymbol{\delta}^*$ of Problem (17.9) also satisfies $\tilde{D}_V(\tilde{\boldsymbol{\delta}}) < 1/100$.

$$\begin{aligned}
\left\| \overrightarrow{(\boldsymbol{\delta}^*/\widehat{\boldsymbol{c}}(e))} \right\|_\infty^2 &\leq \sum_e \left(\frac{\boldsymbol{\delta}^*(e)}{\boldsymbol{c}_+(e)} \right)^2 + \left(-\frac{\boldsymbol{\delta}^*(e)}{\boldsymbol{c}_-(e)} \right)^2 \\
&\leq \sum_e \tilde{D}\left(\frac{\boldsymbol{\delta}^*(e)}{\boldsymbol{c}_+(e)} \right) + \tilde{D}\left(-\frac{\boldsymbol{\delta}^*(e)}{\boldsymbol{c}_-(e)} \right) && \text{By Lemma 17.1.6.} \\
&= \tilde{D}_V(\tilde{\boldsymbol{\delta}}) < 1/100.
\end{aligned}$$

Hence $\left\| \overrightarrow{(\boldsymbol{\delta}^*/\widehat{\boldsymbol{c}}(e))} \right\|_\infty < 0.1$. □

Part V

Further Topics in Combinatorial Graph Algorithms

Chapter 18

Low-Diameter Decompositions

In this chapter, we learn about an algorithm that clusters vertices in graphs by their distance. This clustering is often referred to as *low-diameter decomposition*. We define this concept below and give one of the most versatile and robust algorithms.

Definition 18.0.1. Given an undirected graph $G = (V, E, w)$ with non-negative weights $\boldsymbol{w} \in \mathbb{R}_{\geq 0}$. We say that a partition $X_1, X_2, \ldots, X_k$ of V and a set of edges $E_{del} = \{(u, v) \in E \mid u \in X_i, v \in X_j, i \neq j\}$ form a (D, α)-low-diameter decomposition of G if

1. for every $i \in [1, k]$ and $u, v \in X_i$, we have $\mathbf{dist}_{G[X_i]}(u, v) \leq \alpha D$, and
2. $\forall e \in E$, $\mathbb{P}[e \in E_{del}] \leq w(e)/D$.

Remark 18.0.2. The definition above is with respect to an implicit distribution $\mathcal{D}$ over partition sets $\mathcal{X}$. A more precise statement would be to say that for a distribution $\mathcal{D}$ over partition sets of V, we say that $\mathcal{D}$ is an (D, α)-LDD-distribution if sampling a partition from $\mathcal{D}$ yields the properties stated above. We will refrain from that here, however.

Remark 18.0.3. Instead of the second property, LDDs are often defined to have the weight of edges in E_{del} to be a small fraction of the edges. For unweighted graphs, the best one can hope for is essentially at most m/D intercluster edges E_{del} in total. The above definition allows us to obtain this property in expectation. Defined this way, we also have already seen a reasonable algorithm to obtain such a clustering: ϕ-expander decompositions have few intercluster edges and ϕ-expanders have diameter $O(\log(n)/\phi)$ as you have shown in an exercise.

However, for applications, the probabilistic guarantee is very useful and algorithms to find LDDs are generally much simpler than computing expander decompositions. LDDs have been used as a subroutine in many algorithms recently, we will see a truly exciting application of LDDs in the next chapter. In the first half of this lecture, we analyze a simple algorithm to obtain an LDD. In the second half of the lecture, we extend LDDs and our algorithm to directed graphs.

18.1 Low-Diameter Decomposition in Undirected Graphs

The main result of this half of the lecture is the following.

Theorem 18.1.1. *Given an undirected graph $G = (V, E, \boldsymbol{w})$ with $\boldsymbol{w} \in \mathbb{R}_{\geq 0}$ and $D > 0$, there is an algorithm* ComputeLDD(G, D) *that computes an $(D, 8\log(n))$-low-diameter-decomposition in time $O(m \log n)$ with probability at least $1 - n^{-3}$.*

Exponential Distribution. We denote by ExpDistr(D) a function that samples a real according to the exponential distribution with parameter D. Formally, we sample from a distribution with cumulative distribution function

$$F(x, D) = \begin{cases} 1 - e^{-x/D} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that we have the property that $F(D \cdot 4\log n, D) = 1 - e^{-4\log n} = 1 - 1/n^4$. A really cool property of the exponential distribution is further that it is memoryless which means that for any randomly sampled integer $X \sim$ ExpDistr(D) and fixed positive reals x, s, we have $\mathbb{P}[X > x + s | X > s] = \mathbb{P}[X > x]$.

A Simple LDD Algorithm. Below, we present a simple algorithm to compute a low-diameter decomposition $\mathcal{X}$. The algorithm simply selects an arbitrary vertex r in the graph, then chooses a random radius $\tilde{D}$ and adds the ball $B_G(r, \tilde{D})$ as a cluster to $\mathcal{X}$. The process is then repeated on the graph $G[V \setminus \mathcal{X}]$ (where we abuse notation and really mean $G[V \setminus (\cup_{X \in \mathcal{X}} X)]$), i.e. we repeat on the graph induced by all non-clustered vertices. Clearly, this outputs a partition of the vertex set V in the end.

Algorithm 23: ComputeLDD(G, D)

```
1 X = {}.
2 while X is not a full partition of V do
3     Let r be an arbitrary vertex in V \ X.
4     D̃ ← ExpDistr(D).
5     X ← X ∪ {B_G(r, D̃)}.
6 return (X, E_del = {(u, v) ∈ E | u ∈ X ∈ X, v ∈ Y ∈ X, X ≠ Y})
```

Analysis. We now analyze the algorithm.

Claim 18.1.2. *Given a clustering $\mathcal{X}$ returned by Algorithm 23. We have with probability $\geq 1 - n^{-3}$, that for any $u, v \in X \in \mathcal{X}$, $\mathbf{dist}_{G[X]}(u, v) \leq 8D\log(n)$.*

Proof. From our discussion of the exponential distribution, we have by a simple union bound over at most n iterations of the algorithm, that none of the sampled radii $\tilde{D}$ exceeds $4D\log(n)$ with probability $1 - n^{-3}$. Let us condition on this event.

Next, consider any iteration, where initially we have $\mathcal{X}$ and then we find a new vertex $r \in G[V \setminus \mathcal{X}]$ and a ball $X = B_G(r, \tilde{D})$ that is added to $\mathcal{X}$ at the end of the iteration. We clearly have that $G[X]$ includes a path from each vertex $u \in X$ to r of length at most $4D\log(n)$. But then by the triangle inequality, for every pair $u, v \in X$, $\texttt{dist}\, G[X](u, v) \leq \mathbf{dist}_{G[X]}(u, r) + \mathbf{dist}_{G[X]}(r, v) \leq 8D\log(n)$. □

Claim 18.1.3. *Given a clustering $\mathcal{X}$ returned by Algorithm 23. $\forall e = (u, v) \in E$, $\mathbb{P}[e \in E_{del}] \leq w(e)/D$.*

Detailed Proof Sketch. Consider the first iteration where either u or v is in a ball $B_G(r, \tilde{D})$. By assumption, we have that in this iteration $\tilde{D} \geq \min_{x\in\{u,v\}} \mathbf{dist}_G(r, x)$, i.e. we condition on the event $\mathcal{E}$ that $\tilde{D}$ satisfies this property.

Now, we have that e becomes intercluster and thus is in E_{del} if and only if one of the endpoints is not in the ball, i.e. that $\tilde{D} < \max_{x\in\{u,v\}} \mathbf{dist}_G(r, x)$. We obtain the following inequalities

$$\begin{aligned}
&\mathbb{P}[e \text{ is an intercluster edge} \mid \mathcal{E}] \\
&= \mathbb{P}[\tilde{D} < \max_{x\in\{u,v\}} \mathbf{dist}_G(r, x) \mid \mathcal{E}] \\
&\leq \mathbb{P}[\tilde{D} < \min_{x\in\{u,v\}} \mathbf{dist}_G(r, x) + w(e) \mid \tilde{D} \geq \min_{x\in\{u,v\}} \mathbf{dist}_G(r, x)] \\
&= \mathbb{P}[\tilde{D} < w(e)] \\
&= F(w(e), D) - F(0, D) = (1 - e^{-w(e)/D}) - (1 - e^0) = 1 - e^{-w(e)/D} \\
&\leq 1 - (1 - w(e)/D) = w(e)/D.
\end{aligned}$$

where we use the triangle inequality, that yields $\max_{x\in\{u,v\}} \mathbf{dist}_G(r, x) \leq \min_{x\in\{u,v\}} \mathbf{dist}_G(r, x) + w(e)$. Thus, the event that $\tilde{D} < \min_{x\in\{u,v\}} \mathbf{dist}_G(r, x) + w(e)$ contains the event that e becomes intercluster, and it suffices to upper bound its probability. We then exploit the memoryless property, the definition of the exponential distribution, and then use $1 + x \leq e^x$ to simplify.

Note that we only analyzed the probability conditioned on $\mathcal{E}$. A more rigorous proof has to argue that in the event $\neg\mathcal{E}$, the probability of e becoming intercluster is 0. This requires some more careful set-up. □

It remains to bound the runtime to obtain Theorem 18.1.1. To this end, observe that every iteration can be executed running Dijkstra's algorithm from the vertex r on the induced graph where vertices are only relaxed if their distance estimate drops below $\tilde{D}$. It is not hard to show that each iteration then takes time $O(|E(B_G(r, \tilde{D}))| \log(n))$ and since these balls are vertex disjoint, we have that the total runtime is at most $O(m \log n)$, as desired.

18.2 Generalizing LDDs to Directed Graphs

For our most exciting application, we need to generalize LDDs to directed graphs.We will start out with a new definition, and then argue about why this is a reasonable generalization (other generalizations exist but this one allows to obtain LDDs efficiently and turns out very useful in applications).

Definition 18.2.1. Given a directed graph $G = (V, E, w)$ with non-negative weights $\boldsymbol{w} \in \mathbb{R}_{\geq 0}$, we say that a partition $X_1, X_2, \ldots, X_k$ of V and a set of edges $E_{del} \subseteq E$ form a (D, α)-low-diameter decomposition of G if

1. for every $i \in [1, k]$ and $u, v \in X_i$, we have $\mathbf{dist}_G(u, v) \leq \alpha D$, and
2. $\forall e \in E$, $\mathbb{P}[e \in E_{del}] \leq O(w(e) \log n / D)$.
3. the graph $(G/\{X_1, X_2, \ldots, X_k\}) \setminus E_{del}$, that is the graph where each cluster X_i is contracted into a super-vertex, is a directed acyclic graph (DAG).

Note in particular the subtle difference in Property 1 that clusters have small distances in G and not in the graph induced by the cluster. In Property 2, we gave the probability an $O(\log n)$ slack. While this makes the definition slightly unclean, it makes it easier to state our result.

The third property might initially look a bit strange. Let us briefly argue that it is a reasonable property to have. Note first, that in directed graphs, we might have a highly asymmetric relationship in distances, i.e. $\mathbf{dist}_G(u, v) \ggg \mathbf{dist}_G(v, u)$ for some, or even all the vertices. One example that really stresses this is the cycle graph. While it is very easy to decompose the undirected cycle graph, the directed cycle graph only allows for decomposition into singletons if one insists on having clusters of small diameter D. What the third property expresses is that in this case, while our clusters are not very meaningful, we learn something about the structure of G: removing a few edges (in this case a single one) makes the directed graph acyclic.

In the next lecture, we will see an application of this subroutine in an algorithm that basically does a case split for these two extreme cases: it exploits the small distances in clusters to make progress on the clusters, and it exploits the structural closeness to a DAG on the remainder of the graph.

The rest of this lecture is concerned with proving the following theorem.

Theorem 18.2.2. *Given a directed graph $G = (V, E, \boldsymbol{w})$ with $\boldsymbol{w} \in \mathbb{R}_{\geq 0}$, there is an algorithm* ComputeDirectedLDD(G, D) *that computes an $(D, 8\log(n))$-low-diameter-decomposition in time $\tilde{O}(m)$ with probability at least $1 - n^{-3}$.*

We first give a slightly simpler algorithm that is not quite efficient but allows us to generalize the ideas from the undirected setting cleanly. We then show how to adapt the algorithm to make it efficient.

A Directed LDD Algorithm. Let us show how to implement the algorithm. Again, we are simply carving out balls, one after another, until we know that all remaining distances have small distance to each other. An important detail in this algorithm is that while the ball-carving is executed on the graph $G[V \setminus \mathcal{X}]$, which is the graph with all carved out vertices removed, the eligibility criterion of the while-loop is with respect to the initial input graph G. We also point out that $|V(G)|$ here might be different from n, as n denotes the size of the global input graph, while $|V(G)|$ denotes the size of the input graph to the current procedure. This is important for recursive calls.

Algorithm 24: ComputeDirectedLDD(G, D)

1 $\mathcal{X} = \{\}$; $E_{del} \leftarrow \emptyset$.
2 **while** *there is a vertex* $r \in V \setminus \mathcal{X}$ *with* $|B^{out}_G(r, 4D \log n)| \leq \frac{1}{2}|V(G)|$ ***or*** $|B^{in}_G(r, 4D \log n)| \leq \frac{1}{2}|V(G)|$ **do**
3 $\quad \tilde{D} \leftarrow$ ExpDistr(D).
4 $\quad \# \leftarrow argmin_{\# \in \{out, in\}} |B^{\#}_G(r, \tilde{D})|$.
5 $\quad X \leftarrow B^{\#}_{G[V \setminus \mathcal{X}]}(r, \tilde{D})$.
6 $\quad$ Add all #-edges of X in $G[V \setminus \mathcal{X}]$ to E_{del}.
7 $\quad (\mathcal{X}', E'_{del}) \leftarrow$ ComputeDirectedLDD$(G[X], D)$; `// Recurse.`
8 $\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{X\} \cup \mathcal{X}'$; $E_{del} \leftarrow E_{del} \cup E'_{del}$.
9 $\mathcal{X} \leftarrow \mathcal{X} \cup \{V \setminus \mathcal{X}\}$; `// Add new cluster.`
10 **return** $(\mathcal{X}, E_{del})$

Analysis. Here, we thoroughly discuss the above algorithm but only sketch how to implement it efficiently. A rigorous exposition of these arguments can be found at here[1].

Claim 18.2.3. *Given a clustering $\mathcal{X}$ returned by Algorithm 24. We have for any $u, v \in X \in \mathcal{X}$,* $\mathbf{dist}_G(u, v) \leq 8D \log(n)$.

Proof. Note that a new cluster is only added to $\mathcal{X}$ in Line 11 (all other clusters stem from recursive calls that themselves find clusters by adding them in this line). Consider any two vertices u, v in this new cluster, we have that u reaches more than half the vertices in G in its out-ball of radius $4D \log n$ and v reaches more than half the vertices of G in its in-ball of the same radius. Thus, these balls have an intersection and there is a path of weight at most $8D \log n$. □

Claim 18.2.4. *Given a clustering $\mathcal{X}$ returned by Algorithm 23. $\forall e = (u, v) \in E$, $\mathbb{P}[e \in E_{del}] = O(w(e) \log n / D)$.*

[1] Lecture Notes by Danupon Nanongkai: `https://hackmd.io/@UOnm1XUhREKPYLt1eUmE6g/Sycpovkiq`. The notes only present the arguments for obtaining a weak diameter guarantee for each cluster, however, this can be salvaged by slightly adapting the algorithm and analysis.

Proof Sketch. It is not hard to see that Algorithm 23 only calls itself recursively on disjoint vertex subsets of at most half the size of the original graph (technically, this is only true when conditioned on $\tilde{D}$ never exceeding $4D \log n$, but we will be imprecise here). It is therefore straightforward to see that each vertex u participates in at most $O(\log n)$ calls.

The remainder of the analysis is analogous to the undirected case. Here, we exploit that an out-ball only adds the boundary edges leaving the out-ball to E_{rem} and an in-ball only the boundary edges entering the in-ball. □

Claim 18.2.5. *Given a clustering $\mathcal{X}$ and edge set E_{del} returned by Algorithm 23. We have that $(G/\mathcal{X}) \setminus E_{del}$ is a DAG.*

Proof. Let us first prove that the algorithm maintains the invariant that there is no cycle in $G \setminus E_{del}$ that visits more than one cluster in $\mathcal{X}$.

We prove by induction on the size of $V(G)$. If $|V(G)| \leq 1$, the while loop is not entered and the algorithm returns $\mathcal{X} = \{V\}$.

For larger sizes of $V(G)$, whenever we remove a set X from $V \setminus \mathcal{X}$ to recurse on $G[X]$, we add either all in- or out-edges of X to E_{del}. By the induction hypothesis, the recursive call returns a partition $\mathcal{X}'$ of X such that no cycle in $G[X] \setminus E'_{del}$ visits any two clusters. But this means that any such cycle that does, must use an in- **and** an out-edge of X, but we remove either all in- or all out-edges of X. Similarly, no future subcluster of $V \setminus (\mathcal{X} \cup \{X\} \cup \mathcal{X}')$ can have a cycle with a cluster in $\{X\} \cup \mathcal{X}'$ as again it requires an in- and an out-edge from X.

Finally, each cluster in $\mathcal{X}$ has small diameter by Theorem 18.2.3 and thus, when we contract sets $\mathcal{X}$, we do not create new cycles. Thus $(G/\mathcal{X}) \setminus E_{del}$ is DAG. □

This completes the proof of correctness.

An Efficient Implementation. Finally, we want to argue about runtime. Here, the bottleneck of the algorithm is the computation of the information in the while-loop condition. Unfortunately, it is not clear how to obtain this information in reasonable time.

Instead, we resort to weakening the while-loop condition slightly to obtain an efficient algorithm: we initially sample a set S consisting of $\Theta(\log n)$ vertices uniformly at random. We then run Dijkstra from each vertex in S on the graph G and the reverse graph. This allows us to obtain for each vertex $v \in V$, the information how large the intersection of S is with its out- and in-ball of radius $4D \log n$. This now gives an estimator that w.h.p. can detect whether a vertex v has an out- and in-ball of size more than $\frac{1}{2}|V(G)|$ or otherwise guarantees that both balls are of size at most $\frac{3}{4}|V(G)|$. In our case, we could check whether a $\frac{2}{3}$-fraction of the vertices in S intersect which yields a good estimator on these two values.

Since balls in graph $G[V \setminus \mathcal{X}]$ are subsets of balls in graph G (for the same center and radius), we have that every recursive call is indeed on a graph that has the number of vertices decreased by a constant fraction compared to the original graph. For the remaining

Algorithm 25: ComputeDirectedLDD(G, D)

$\mathcal{X} = \{\}$; $E_{del} \leftarrow \emptyset$.
Sample each vertex $v \in V$ into set S with probability $\frac{100 \log(|V(G)|)}{|V(G)|}$.
Compute in- and out-SSSP for all $s \in S$.
while *there is a vertex* $r \in V \setminus \mathcal{X}$ *with* $|B^{out}_G(r, 4D \log n) \cap S| \leq \frac{2}{3}|S|$ ***or*** $|B^{in}_G(r, 4D \log n) \cap S| \leq \frac{2}{3}|S|$ **do**
    $\tilde{D} \leftarrow$ ExpDistr(D).
    $\# \leftarrow argmin_{\#\in\{out,in\}}|B^{\#}_G(r, 4D \log n) \cap S|$.
    $X \leftarrow B^{\#}_{G[V\setminus\mathcal{X}]}(r, \tilde{D})$.
    Add all #-edges of X in $G[V \setminus \mathcal{X}]$ to E_{del}.
    $(\mathcal{X}', E'_{del}) \leftarrow$ ComputeDirectedLDD$(G[X], D)$; `// Recurse.`
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{X\} \cup \mathcal{X}'$; $E_{del} \leftarrow E_{del} \cup E'_{del}$.
$\mathcal{X} \leftarrow \mathcal{X} \cup \{V \setminus \mathcal{X}\}$; `// Add new cluster.`
return $(\mathcal{X}, E_{del})$

vertices that are added as a cluster $V \setminus \mathcal{X}$ in the current call, we have that they reach half the vertices in G in in- and out-ball w.h.p. and thus, we can use an argument as in Theorem 18.2.3, we have that they are at distance at most $8D \log n$.

It is not hard to bound via the above guarantees the number of calls that every vertex participates in by $O(\log n)$ which yields the two other properties of LDDs. And further, since sampling is efficient, and SSSP computations take time near-linear in the number of edges of the input graph, we can bound the runtime now by $\tilde{O}(m)$. We leave it to the reader to verify this bound.

Chapter 19

Negative Single-Source Shortest Paths

In this lecture, we are interested in the following result. It says that even in graphs with negative edge weights, we can find the single-source shortest paths in near-linear time! This result has been obtained extremely recently and the algorithm is a beautiful combination of some of the most useful modern graph algorithmic techniques.

Theorem 19.0.1. *Given a directed graph $G = (V, E, \boldsymbol{w})$ with $\boldsymbol{w} \in \mathbb{R}^V$ such that there exists no negative cycle in G, and a dedicated source vertex $s \in V$, the distances $\mathbf{dist}_G(s, t)$ can be computed in time $\tilde{O}(m \log(W))$ where $W = \|\boldsymbol{w}\|_\infty$, i.e. W is the largest absolute weight (and where we assume that the smallest absolute non-zero weight is at least 1).*

Henceforth, we assume that any graph G under consideration does not contain a negative weight cycle.

19.1 A Not-So-Brief Recap: Classic Algorithms for Handling Negative Weights

Bellman-Ford. The classic algorithm that should come to mind when computing single-source shortest paths with negative weights is Bellman-Ford's algorithm. The algorithm works by initializing distance estimates $\hat{d}(s, u)$ with infinity and $\hat{d}(s, s)$ with 0. It then proceeds for n rounds by relaxing the edge set in every round.

The following theorem summarizes the guarantees of the algorithm.

Theorem 19.1.1. *Algorithm 26 takes as input a directed, weighted graph $G = (V, E, \boldsymbol{w})$ where weights are allowed to be negative, but that contains no negative cycle; and a source vertex $s \in V$. It runs in time $O(mn)$ and outputs the correct distances from s to every vertex $v \in V$.*

Proof. The algorithm runs for n iterations and relaxes m edges in each iteration, each in $O(1)$ time. This establishes the runtime.

Algorithm 26: BellmanFord(G, s)

```
1 ∀u ∈ V \ {s}, d̂(s,u) ← ∞; d̂(s,s) ← 0.
2 for i = 1, 2, ..., n do
3     foreach e = (u,v) ∈ E do
4         d̂(s,v) ← min{d̂(s,v), d̂(s,u) + w(e)}
5 return {d̂(s,v)}_v
```

We prove correctness by induction. We prove that after the h-th iteration of the algorithm, for all vertices v that have a shortest path consisting of at most h edges have $\hat{d}(s,v)$ is equal to the real s to v distance.

The base case $h = 0$ is trivial because $\hat{d}(s,s)$ is initialized to 0. For the inductive step $h \mapsto h+1$, we use the subpath property of shortest paths, that is, for any v that has an shortest s to v shortest path consisting of exactly $h+1$ edges, we have that for the vertex u preceding v on this path, we have that the subpath from s to u is a shortest path itself and has clearly $< h+1$ edges (it is easy to establish the subpath property via a proof by contradiction). Thus in the $h+1$-th iteration, when the edge $e = (u,v)$ is relaxed, we have $\hat{d}(s,v) \leftarrow \hat{d}(s,u) + w(e) = \mathbf{dist}_G(s,u) + w(e) = \mathbf{dist}_G(s,v)$. □

A Heuristic for Speeding-Up Bellman-Ford. If you think a bit longer about Bellman-Ford and why you cannot use Dijkstra, you might actually stumble upon the following idea that might improve runtime in practice: consider that there is a shortest path from s to v that consists of at most h negative edges. Then, we could try to relax negative edges by Bellman-Ford, and relax all non-negative edges between two consecutive edges on the shortest path by using Dijlstra's algorithm. In fact, we know that after h rounds of this algorithm v's distance estimate just becomes the distance and we therefore never need to relax after round $h+1$ from v.

Let us make that intuition precise.

Definition 19.1.2. Given a directed, weighted graph $G = (V, E, \boldsymbol{w})$ with negative weights but no negative cycle and a source $s \in V$. For any vertex $v \in V$, we let $\eta(s,v)$ be the minimum number of negative edges on any shortest s to v path.

The pseudo-code below now formalizes our intuitive approach. It reduces the runtime to $O((\sum_v \eta(s,v) \cdot \mathbf{deg^{out}}(v) + m) \log n)$ which is always upper bounded by $O(nm \log n)$.

We can now prove the following theorem which prove that the algorithm above is faster at least for the special case when $\eta(G)$ is much smaller than n.

Theorem 19.1.3. *Algorithm* HeuristicBellmanFord(G, s) *runs in time* $O((\sum_v \eta(s,v) \cdot \mathbf{deg^{out}}(v) + m) \log n)$ *time and outputs the correct* s *to* v *distances.*

Algorithm 27: HeuristicBellmanFord(G, s)

```
1  ∀u ∈ V \ {s}, d̂(s,u) ← ∞; d̂(s,s) ← 0.
2  for i = 1, 2, ..., n do
       /* Relaxation via Bellman-Ford.                                  */
3      foreach e = (u,v) ∈ E where d̂(u) was decreased last iteration do
4          d̂(s,v) ← min{d̂(s,v), d̂(s,u) + w(e)}.
       /* Relaxation via Dijkstra's Algorithm.                          */
5      Q ← V.
6      while Q ≠ ∅ do
7          Let u be the vertex in Q that has smallest d̂(s,u).
8          foreach e = (u,v) ∈ E, w(e) ≥ 0, where d̂(u) was decreased in the current
             iteration do
9              d̂(s,v) ← min{d̂(s,v), d̂(s,u) + w(e)}.
10 return {d̂(s,v)}_v
```

Proof Sketch. The runtime is trivially analyzed. To avoid a tedious case analysis, let us assume that all out-going edges of s are negative (this can be enforced w.l.o.g.).

Then correctness follows by induction on $\eta(s, v)$, i.e. after the h-th iteration of the algorithm, we have for all vertices $v \in V$ with $\eta(s, v) \leq h$, that $\hat{d}(s, v) = \mathbf{dist}_G(s, v)$. The base case is again trivial. For the inductive step $h \mapsto h + 1$, we have that for any v with $\eta(s, v) = h + 1$, by the subpath property, we have that the vertex u just before the last negative edge on the shortest s to v path of $h + 1$ edges, that $\hat{d}(u)$ is equal to the real distance after iteration h. But then u is relaxed in the round after learning the real distance by Bellman-Ford which then decreases the distance of the next vertex on the path. Dijkstra's algorithm then relaxes all non-negative edges in the correct order on the path which leaves v's estimate decreased to the real distance. □

All-Pairs Shortest Paths in Negative-Weight Graphs via Price Function. While we are at reviewing classic algorithm, let us also briefly review Johnson's algorithms. This algorithm allows to compute APSP in directed graphs even when weights are negative (but again no negative cycles).

The key idea used in Johnson's algorithm is the concept of a price function.

Definition 19.1.4 (Price Function)**.** For any vector $\phi \in \mathbb{R}$, we say ϕ is a *price function* if $\boldsymbol{w}_\phi = \boldsymbol{w} + \boldsymbol{B}^\top \phi \geq \mathbf{0}$, i.e. where for each edge $e = (u, v) \in E$ directed from u to v, we have that $\boldsymbol{w}(e) + \phi(u) - \phi(v) > 0$. We write G_ϕ to mean the graph $(V, E, \boldsymbol{w}_\phi)$. We further restrict price functions such that the smallest entry of ϕ is equal to 0. We implicitly enforce this property throughout but we can also assume it wlog since $\boldsymbol{w}_\phi$ is invariant to ϕ being shifted along the all-ones vector $\mathbf{1}$.

A price function is a nice tool since once one obtains a price function, one can run Dijkstra's algorithm on the graph G_ϕ and extract the real distances efficiently. Let us briefly prove that.

Claim 19.1.5. *Given a directed graph $G = (V, E, \boldsymbol{w})$ and any function ϕ over the vertices. Then we have for any vertices $s, t \in V$ that $\mathbf{dist}_G(s,t) = \mathbf{dist}_{G_\phi}(s,t) + h(s) - h(t)$.*

Proof. For any s to t path π in G, we have that $\boldsymbol{w}_\phi(\pi) = \sum_{e=(u,v)\in\pi} \boldsymbol{w}_\phi(e) = \sum_{e=(u,v)\in\pi} \boldsymbol{w}(e) + \phi(u) - \phi(v) = \boldsymbol{w}(\pi) + \sum_{e=(u,v)\in\pi} \phi(u) - \phi(v)\phi$. And we have a telescoping sum, and therefore this simplifies to $\mathbf{dist}_{\boldsymbol{w}}(\pi) = \mathbf{dist}_{\boldsymbol{w}_\phi}(\pi) + h(s) - h(t)$. But this means that all s to t paths are increased by the same constant additive factor and thus the shortest path in G w.r.t. $\boldsymbol{w}$ is still a shortest path in G_ϕ. □

To see that price functions even exist consider the following algorithm which computes it via Bellman-Ford.

Algorithm 28: ComputePriceFunction(G)

1 Let G' be the graph G with an additional dummy source d, and edges (d, v) of weight 0 for every $v \in V$.
2 Run HeuristicBellmanFord(G', d).
3 **return** $\phi = \{\mathbf{dist}_{G'}(d, v)\}_{v\in V}$

Claim 19.1.6. *Algorithm 28 returns a price function ϕ for G. It runs in time $O((\sum_v \eta(d,v) \cdot \mathbf{deg}^{\mathbf{out}}(v) + m) \log n)$.*

Proof. In fact, we can prove that ϕ extended by the value $\phi(d) = 0$ for the dummy source d is a price function for G'.

To see this, observe that the distance from d to any vertex v in G weighted by $\boldsymbol{w}_\phi$ is 0 by definition. Thus, the shortest path from d to any vertex v is of weight exactly 0 w.r.t. ϕ. But assume that there is an edge $e = (x, v)$ with $\boldsymbol{w}_\phi(e) < 0$. Then, the path $(d, x)(x, v)$ is of negative weight. But this contradicts that v is at distance 0 from d. □

Once a price function is found, one can obtain All-Pairs Shortest Paths by simply running Dijkstra's algorithm from each vertex on the graph G_ϕ. The runtime of this algorithm becomes $\tilde{O}(mn)$.

19.2 Scaling for the Negative-Weight SSSP Problem

Next, we use a technique that is often referred to as *scaling*. It appeared first in the context of obtaining max-flow algorithms. The idea is that we can pre-condition/ normalize a problem and then only try to improve the situation slightly instead of completely solving the problem. To be precise, we want to prove the following statement.

Theorem 19.2.1. *Given an algorithm* PriceFunctionOnNormalizedGraph(G) *that may assume that the input graph* $G = (V, E, \boldsymbol{w})$ *has* $\boldsymbol{w} \geq -2 \cdot \mathbf{1}$*, and that then computes a function* ϕ *such that* G_ϕ *has its most negative weight being at least* -1 *and runs in time* $\mathcal{T}(m, n)$*.*

Then, there exists an algorithm NegativeSSSP(G, s) *for any graph* $G = (V, E, \boldsymbol{w})$ *that solves the negative SSSP problem in time* $O(\mathcal{T}(m, n) \log(W_{max}/W_{min}))$ *where* W_{max} *is the largest absolute weight in* G *and* W_{min} *is the smallest absolute non-zero weight.*

Using scaling initially, we can and will assume henceforth that initially $W_{min} \geq 1$ (one simply needs to scale by $1/W_{min}$ to make this true).

A Scaling Algorithm. Let us now present such an algorithm. The idea is that one can initially scale the graph down to satisfy the condition that the smallest weight is at least -2, and then using the produced "price" function, scale up by a factor 2 in the next round when inputting the graph reweighted by this first price function. The algorithm below proposes to keep going with this process until one is at a precision where one can round the "price function" into a real price function ϕ for G. It then remains to run Dijkstra's algorithm.

Algorithm 29: NegativeSSSP(G, s)

$W_{max} \leftarrow \|\boldsymbol{w}\|_\infty$.
$\phi_0 \leftarrow \mathbf{0}$.
for $i = 1, 2, \ldots, \eta = \lceil \log_2(W_{max} n^2) \rceil$ **do**
  $\phi_i \leftarrow$ PriceFunctionOnNormalizedGraph $\left((G \cdot 2^i / \lceil \log_2(W_{max}) \rceil)_{\phi_{i-1}} \right)$.
$\phi \leftarrow \frac{\lceil \log_2(W_{max}) \rceil}{\lceil \log_2(W_{max} n^2) \rceil} \cdot \min\{\mathbf{0}, \phi_\eta\}$.
return Dijkstra(G, s, ϕ).

Here, we will not give proof that this algorithm indeed yields Theorem 19.2.1, but simply use its result. For a proof, we refer you to the original paper.

19.3 The (Normalized) Negative-Weight SSSP Problem

Recall, we can now assume that the most negative weight in G is -2, and we want to produce a "price" function ϕ such that the most negative weight in G_ϕ is -1. For the rest of this section, we focus on giving an implementation of PriceFunctionOnNormalizedGraph() with runtime $\tilde{O}(m\sqrt{m})$.

Constant-Degree Assumption. It will be convenient to assume that G has constant-degree. This is w.l.o.g. since we can take an arbitrary graph G and transform it into a

metric-preserving graph G' with $O(m)$ vertices and edges by replacing each vertex v with d edges incident, by a zero-weight cycle with d vertices where each edge on the cycle receives one of the edges originally incident to v.

Notation. We introduce the notation G^{+i} to denote the graph $(V, E, \boldsymbol{w} + i \cdot \mathbf{1})$, i.e. the graph G after adding i to each edge weight. We will use

- G^{+0} which is the initial graph;
- G^{+1} which is the graph for which ϕ will be a real price function; and
- G^{+2} which has the property that every edge weight is non-negative.

The Algorithm. To obtain Single-Source Shortest Paths from a dedicated source s, we use the following algorithm. The algorithm first decomposes G into LDDs where it ignore edge weights and uses weigths 1 instead. It then computes a price function for each cluster X_i in the LDD. Finally, combining these price functions in the most straight-forward into a global function ϕ (that is not a price function but as we will see is close to a price function), we can invoke our heuristic for Bellman-Ford on the graph G_ϕ, and then return the computed distances after adjusting them for the price function.

Algorithm 30: PriceFunctionOnNormalizedGraph($G = (V, E, \boldsymbol{w})$)

1 $(\mathcal{X} = \{X_1, X_2, \ldots, X_k\}, E_{del}) \leftarrow$ ComputeLDD($G^{+2}, \sqrt{n}$).
`// Find "price" function` ϕ_1 `on clusters.`
2 **for** $i = 1, 2, \ldots, k$ **do** $\phi_1^i \leftarrow$ ComputePriceFunction($G^{+1}[X_i]$). ;
3 $\phi_1 \leftarrow \left[\phi_1^1, \phi_1^2, \ldots, \phi_1^k\right]^\top$.
`// Adapt "price" function to` ϕ_2 `which also works for DAG edges.`
4 Let π be a topological order of $X_1, X_2, \ldots, X_k$ in $(G/\mathcal{X}) \setminus E_{del}$.
5 **for** $i = 1, 2, \ldots, k$ **do** $\phi_2^i \leftarrow \phi_1^i + (2\|\phi_1\|_\infty + 1) \cdot \pi(X_1) \cdot \mathbf{1}_{X_1}$. ;
6 $\phi_2 \leftarrow \left[\phi_2^1, \phi_2^2, \ldots, \phi_2^k\right]^\top$.
`// Obtain "price" function` ϕ_3 `that works for all of` G^{+1}.
7 $\phi_3 \leftarrow$ ComputePriceFunction($G_{\phi_2}^{+1}$).
8 **return** ϕ_3.

Analysis. From the algorithm, it is immediately clear that ϕ_3 is a price function for G^{+1} and thus satisfies our claim. It remains to bound the runtime.

Claim 19.3.1. *For each cluster $X_i \in \mathcal{X}$, we have that* ComputePriceFunction($G^{+1}[X_i]$) *runs in time $\tilde{O}(n\sqrt{n})$.*

Proof. Recall that ComputePriceFunction($G^{+1}[X_i]$) internally adds a dummy source d and edges of weight 0 from d to everyone in $G^{+1}[X_i]$, and then computes the price function in time $\tilde{O}(\sum_v \eta(d,v))$ (where we use that G is assumed constant-degree).

Let us now show that $\eta(d,v) \leq \sqrt{n}$ for all v which then establishes the claim. We note that $\eta(\cdot,\cdot)$ is here defined w.r.t. $G^{+1}[X]$ (plus the vertex d). Assume for the sake of contradiction that $\eta(d,v) \geq 8\sqrt{n}\log n + 1$. Since there is an edge (d,v) of weight 0 in this graph, we have that the weight of the shortest d to v path P must be of non-positive weight in G^{+1}. Let u be the first vertex on this shortest path after d. Then, the shortest u to v path P' is exactly the path P omitting the first vertex, and also be of weight at most 0. Note that P' is contained in G (without d) and consists of at least $8\sqrt{n}\log n$ edges.

Finally, note that in G^{+0} each edge carries weight one less than in G^{+1}. So the path P' is of weight at most $-8\sqrt{n}\log n$ in $G = G^{+0}$, thus $\mathbf{dist}_G(u,v) \leq -8\sqrt{n}\log n$. But at the same time, we know from the LDD decomposition (see Theorem 18.2.2) that since u and v are in the same cluster w.r.t. G^{+2} that $\mathbf{dist}_G(v,u) < \mathbf{dist}_{G^{+2}}(v,u) \leq 8\sqrt{n}\log n$.

But this implies that there is a negative cycle in G which yield a contradiction. □

To analyze the runtime of the call to ComputePriceFunction($G^{+1}_{\phi_2}$) in Line 7, we establish that ϕ_2 indeed is a price function for $G^{+1} \setminus E_{del}$.

Claim 19.3.2. *ϕ_2 is a price function for $G^{+1} \setminus E_{del}$.*

Proof. Since price functions are invariant to shifts by $\mathbf{1}$, it is not hard to verify that for every edge e in $G[X_i]$, we have that $\boldsymbol{w}_{\phi_2}(e) = \boldsymbol{w}_{\phi_1}(e) \geq -1$ and thus in G^{+1}, the function is a price function restricted to edges in clusters.

Let us now consider any edge $e = (u,v)$ in $G \setminus E_{del}$ that is not in a cluster. Let $u \in X_i$ and $v \in X_j$ for $i \neq j$. We have by definition, that $\pi(X_i) + 1 \leq \pi(X_j)$. But $|\phi_1(v) - \phi_1(u)| \leq 2\|\phi_1\|_\infty$ and in G the smallest possible edge weight is -2 by assumption. Thus,

$$\begin{aligned}
\boldsymbol{w}_{\phi_2}(e) &= \boldsymbol{w}(e) + \phi_1(u) - \phi_1(v) + (2\|\phi_1\|_\infty + 1)(\pi(X_j) - \pi(X_i)) \\
&\geq \boldsymbol{w}(e) + \phi_1(u) - \phi_1(v) + 2\|\phi_1\|_\infty + 1 \\
&\geq \boldsymbol{w}(e) + 1 \\
&\geq -1.
\end{aligned}$$

Again, this confirms that ϕ_2 is also a price function on the edges that are between clusters (and not in E_{del}) with respect to G^{+1}. □

Claim 19.3.3. *The runtime of the call to* ComputePriceFunction($G^{+1}_{\phi_2}$) *in Line 7 is $\tilde{O}(n\sqrt{n})$ in expectation.*

Proof. Again recall that ComputePriceFunction($G^{+1}_{\phi_2}$) internally adds a dummy source d and edges of weight 0 from d to everyone in $G^{+1}_{\phi_2}$, and then computes the price function in time $\tilde{O}(\sum_v \eta(d,v))$ (where we use that G is assumed constant-degree).

To finish the proof it remains to show that for every vertex $v \in V$, $\mathbb{E}[\eta(d, v)] = O(\sqrt{n} \log n)$. To see this, fix any shortest path P from d to v in the graph G^{+1} (plus the vertex d). Let P' be again the shortest u to v path where u is the vertex that appears on P after d. Again, we have that P' has weight at most 0 in G^{+1}, since P is of weight at most 0 since it is a shortest d to v path and a zero-weight edge (d, v) exists.

Since further P' consists of at most $n-1$ edges, we have that the weight of P' in G^{+2} is at most $n-1$. But by Theorem 18.2.2, this implies that in expectation, at most $\tilde{O}(\frac{n-1}{\sqrt{n}})$ edges from P' are added to E_{del}. But in $G^{+1}_{\phi_2}$, we have that only edges in E_{del} are negative. The claim follows. □

19.4 A Near-Linear-Time Algorithm

Using the same framework as above one can in fact obtain runtime $\tilde{O}(m)$ for implementing PriceFunctionOnNormalizedGraph(). The only missing trick is that one can also scale of the maximum number of negative edges on any shortest path. To get an idea, observe that if we would know beforehand that this number is at most $\sqrt{n}$, then invoking the LDD algorithm with parameter $n^{1/4}$ in-place of $\sqrt{n}$ will yield an $\tilde{O}(m^{5/4})$ algorithm. The key claim to look at is Theorem 19.3.3 where we then can use that any shortest path has at most $\sqrt{n}$ negative edges and so its total weight in G^{+2} is at most $\sqrt{n}$, and therefore, we now add at most $n^{1/4}$ edges to E_{del} in expectation.

Bibliography

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[Che14] Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 654–663, 2014.

[Che15] Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 1–10, 2015.

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 451–460, New York, NY, USA, May 2008. Association for Computing Machinery.

[KS16] R. Kyng and S. Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, 2016.

[LS20a] Yang P. Liu and Aaron Sidford. Faster Divergence Maximization for Faster Maximum Flow. *arXiv:2003.08929 [cs, math]*, April 2020.

[LS20b] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

[Spi19] Daniel A Spielman. Spectral and Algebraic Graph Theory, 2019.

[SS11] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 81–90, New York, NY, USA, 2004. Association for Computing Machinery.

[T+15] Joel A Tropp et al. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

[Tar83] Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[Tro19] Joel A Tropp. Matrix concentration & computational linear algebra. 2019.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.