CS 521: Linear Programming **Rutgers: Fall 2022**

Lecture 11

November 18, 2022

Instructor: Sepehr Assadi *Scribes: Hanna Komlós, Vishal Patel, Adarsh Srinivasan, Janani Sundaresan*

Contents

1 Convex Functions Recap

We begin with a summary of the basic definitions related to convexity covered in Lecture 7. Some of the statements and proofs here also appear in Lecture 8 but we repeat them here again for completeness.

Definition 1 (Convex function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is called convex if, for every $x, y \in \mathbb{R}^n, t \in [0, 1]$, $f(t \cdot x + (1 - t) \cdot y) \leqslant t \cdot f(x) + (1 - t) \cdot f(y)$.

Geometrically, this just means that the curve of any convex function lies below the line connecting two points on the curve (refer to Figure 1 for an example).

We also need the definition of a convex set.

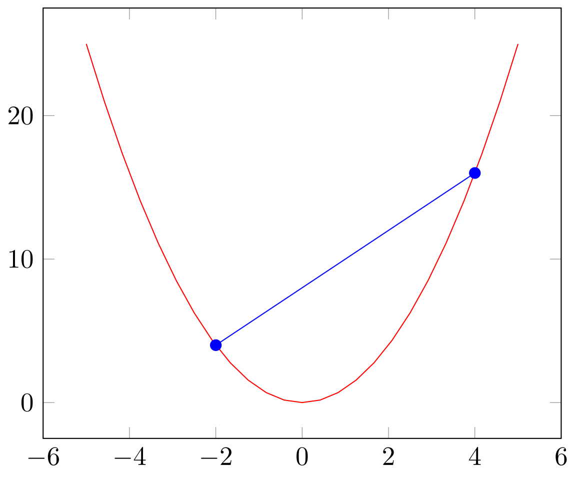

Figure 1: The convex function $f(x) = x^2$ and the line segment connecting $(-2, 4), (4, 16)$

Definition 2 (Convex Set)**.** A set $S \subseteq \mathbb{R}^n$ is called convex if for all $x, y \in S$, and for all $0 \leqslant t \leqslant 1$

$$(1-t) \cdot x + t \cdot y \in S$$

Geometrically, it means that the straight line joining any two points in the set lies entirely within the set.

Next we recall the definition of the gradient of a function.

Definition 3. The gradient of function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $x \in \mathbb{R}^n$, denoted by $\nabla f(x)$, is a vector in $\mathbb{R}^n$ with

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

That is, $\nabla f(x) \in \mathbb{R}^n$ is the vector of partial derivatives of each variable x_i for $i \in [n]$ at point $x \in \mathbb{R}^n$.

Gradients can be used to quantify the change in any specific direction $v \in \mathbb{R}^n$.

Claim 4. *For any $x, v \in \mathbb{R}^n$,*

$$\langle \nabla f(x), v \rangle = \lim_{t \to 0} \frac{f(x + t \cdot v) - f(x)}{t}.$$

Proof. This follows from the chain rule.

$$\lim_{t \to 0} \frac{f(x + t \cdot v) - f(x)}{t} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} v_i = \langle \nabla f(x), v \rangle.$$

□

We will prove a key property about convex functions that will be used throughout.

Proposition 5. *For any convex function $f : \mathbb{R}^n \to \mathbb{R}$, and for all $x, y \in \mathbb{R}^n$, the following condition holds:*

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle.$$

Proof. Let t be any real number in $(0, 1)$. We have,

$$f(x + t \cdot (y - x)) = f(t \cdot y + (1-t) \cdot x) \leqslant t \cdot f(y) + (1-t) \cdot f(x) = f(x) + t \cdot (f(y) - f(x)),$$

where in the inequality is by Definition 1. Thus, by taking $f(x)$ in the RHS to the LHS instead and dividing by t, we get,

$$\frac{f(x + t \cdot (y - x)) - f(x)}{t} \leqslant f(y) - f(x).$$

Using Claim 4 (for $v = y - x$) and by taking the limit of t to 0, we have,

$$\langle \nabla f(x), y - x \rangle = \lim_{t \to 0} \frac{f(x + t \cdot (y - x)) - f(x)}{t} \leqslant f(y) - f(x),$$

concluding the proof. □

Geometrically, Proposition 5 can be interpreted as the tangent line to any convex function lying below the function. That is, when $y = x + \varepsilon$ is for some very small ε, $f(x + \varepsilon) \geqslant f(x) + \varepsilon(y - x)$.

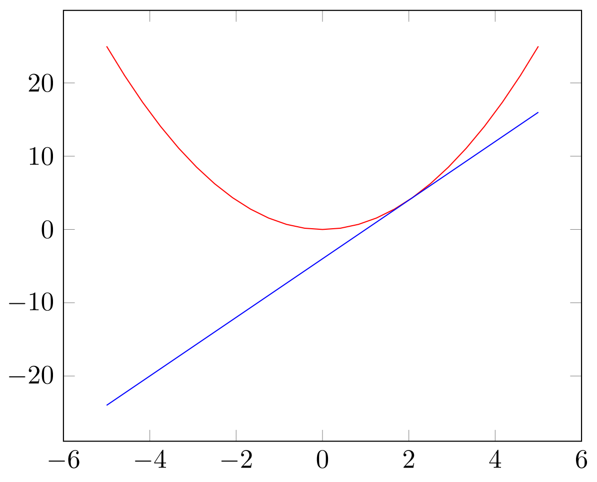

Figure 2: The convex function $f(x) = x^2$ and the tangent at $x = 2$

Intuitively, the gradient provides the direction of the *steepest ascent*. Consider any unit vector v. Since $\langle \nabla f(x), v \rangle$ is an inner product, we maximize this inner product when v points in the same direction as $\nabla f(x)$ and minimize the inner product when v points in the opposite direction as $\nabla f(x)$.

1.1 Convex Optimization

Recall the definition of a convex optimization problem.

Problem 1. A convex optimization problem is any minimization problem of the following form, where f is a convex function and K is a convex set.

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

We prove another key property obeyed only by convex functions now.

Lemma 6. *Any $x^* \in \mathbb{R}$ is a global minimum of a differentiable convex function f if and only if $\nabla f(x^*) = 0$.*

Proof. If $\nabla f(x) = 0$ for some $x \in \mathbb{R}^n$, then, for any $y \in \mathbb{R}^n$, by Proposition 5,

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle = f(x),$$

making x a global minimum.

If x^* is a global minimum, then we have that for all $v \in \mathbb{R}^n$, $f(x^* + t \cdot v) \geqslant f(x^*)$. In particular, for $v = -\nabla f(x^*)$, by Claim 4,

$$0 \leqslant \lim_{t \to 0} \frac{f(x^* + t \cdot v) - f(x^*)}{t} = \langle \nabla f(x^*)\,,\, v \rangle = -\|\nabla f(x^*)\|_2^2,$$

which forces $\nabla f(x^*) = 0$ as otherwise the RHS would be negative. □

This concludes our recap and properties of convex functions.

2 Gradient Descent

We have a natural progression to an algorithm to find the global minimum of a convex function (solving Problem 1 without any constraints) based on Lemma 6. If we choose a point x where $\nabla f(x) \neq 0$, then we know that the opposite direction of the gradient is the direction of steepest descent in our function. We can simply choose x' some distance away from x in this direction to decrease our objective function and proceed iteratively. This is in essence, what the *Gradient Descent* algorithm does. We now formalize this.

Algorithm 1 (Gradient Descent). For unconstrained convex optimization (Problem 1 with $K = \mathbb{R}^n$).

1. Let x_0 be an initial point and $\eta \in (0, 1)$ be a parameter to be defined later.
2. For $t = 1$ to $T - 1$ iterations, let
$$x_{t+1} = x_t - \eta \cdot \nabla f(x_t).$$
3. Return x_T.

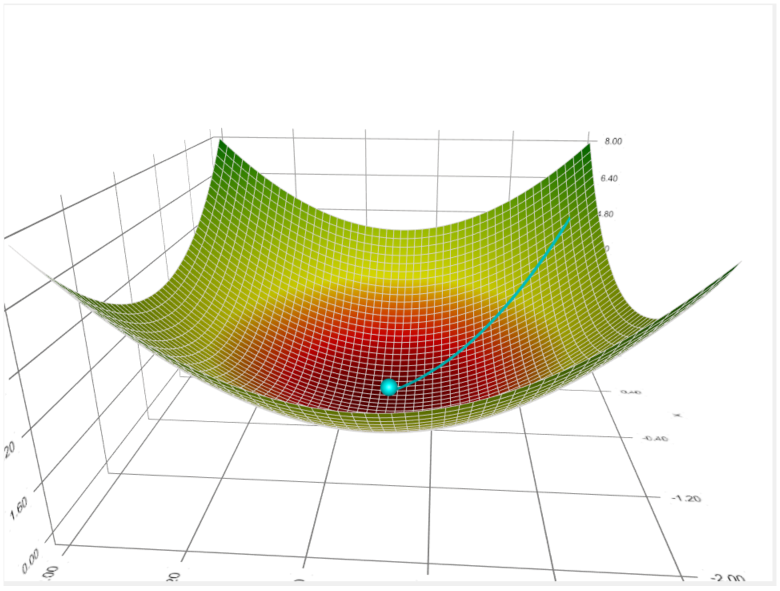

Figure 3: A visualization of gradient descent based on software by Lily Jiang in Towards Data Science[1].

The parameter η is referred to as the step size or "learning rate", which represents how quickly we move in the direction of the gradient from any point. We are interested in what theoretical guarantees Algorithm 1 provides based on the number of iterations T. We need a few more definitions to prove such guarantees.

[1]https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c.

2.1 Properties Needed for Gradient Descent

We want our convex function to not have arbitrary values for the gradient. If the convex function changes haphazardly, we may miss the optimal point x^* we are interested in while executing Algorithm 1.

Definition 7. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to have **L-Lipschitz gradient** if, for all $x, y \in \mathbb{R}^n$,

$$\|\nabla f(x) - \nabla f(y)\|_2 \leqslant L \cdot \|x - y\|_2.$$

The following lemma shows a somewhat of a converse to Proposition 5 by upper bounding how much gap can be in the inequality of Proposition 5 (as an aside, the term in the LHS of this lemma is referred to as the **Bregmann divergence** of f).

Lemma 8. *For any $x, y \in \mathbb{R}^n$, and $f : \mathbb{R}^n \to \mathbb{R}$ with L-Lipschitz gradient,*

$$f(y) - f(x) - \langle \nabla f(x), y - x\rangle \leqslant \frac{L}{2} \cdot \|x - y\|_2^2.$$

Proof. Let $x, y \in \mathbb{R}^n$. Define a function $g : [0, 1] \to \mathbb{R}$ as,

$$g(t) = f((1 - t) \cdot x + t \cdot y) - f(x) - \langle \nabla f(x), t \cdot (y - x)\rangle.$$

Observe that $g(0) = 0$ and that $g(1)$ is the quantity we are interested in. The derivative of g gives,

$$g'(t) = \langle \nabla f((1 - t) \cdot x + t \cdot y), y - x\rangle - \langle \nabla f(x), y - x\rangle.$$

We can now apply the Fundamental Theorem of Calculus to have,

$$\begin{aligned}
g(1) = g(1) - g(0) &= \int_{t=0}^{1} g'(t) \cdot dt \\
&= \int_{t=0}^{1} \langle \nabla f((1 - t) \cdot x + t \cdot y) - \langle \nabla f(x)\,,\, y - x\rangle \cdot dt && \text{(by the equation above)} \\
&\leqslant \int_{t=0}^{1} \|\nabla f((1 - t)x + ty) - \nabla f(x)\|_2 \cdot \|y - x\|_2 \cdot dt && \text{(by Cauchy-Schwarz)} \\
&\leqslant \int_{t=0}^{1} L \cdot \|t \cdot (y - x)\|_2 \cdot \|y - x\|_2 \cdot dt && \text{(since } f \text{ has } L\text{-Lipschitz gradient)} \\
&= L \cdot \|y - x\|^2 \cdot \int_{t=0}^{1} t \cdot dt \\
&= \frac{L}{2} \cdot \|y - x\|^2. && \square
\end{aligned}$$

Another property we require is that the initial point not be too far off from the optimal point x^*.

Definition 9. Initial point x_0 is said to have **Bounded Distance** D if

$$\max_{x, f(x) \leqslant f(x_0)} \|x - x^*\| \leqslant D,$$

i.e., all the points with value at most $f(x_0)$ are not "too far" from x^* in ℓ_2-distance.

We note that this property can be replaced with the more natural property that $\|x_0 - x^*\| \leqslant D$ when analyzing gradient descent for functions with L-Lipschitz gradient (we will revisit this later).

2.2 Analysis of Gradient Descent

We are now ready state the main result of this section about Algorithm 1.

Theorem 10. *Let f be a convex function with L-Lipschitz gradient, and suppose initial point x_0 satisfies the bounded distance property with bound D. Then, for any $\varepsilon \in (0, 1/2)$, after $T = O\left(\frac{LD^2}{\varepsilon}\right)$ iterations, Algorithm 1 with parameter $\eta = \frac{1}{L}$ converges to a point x_T such that $f(x_T) \leqslant f(x^*) + \varepsilon$.*

Proof. We begin by bounding the change in the function between iterations. For any $t \leqslant T$,

$$\begin{aligned} f(x_{t+1}) - f(x_t) &\leqslant \langle \nabla f(x_t), x_{t+1} - x_t \rangle + \frac{L}{2} \cdot \|x_{t+1} - x_t\|_2^2 && \text{(by Lemma 8)} \\ &= \langle \nabla f(x_t), -\eta \nabla f(x_t) \rangle + \frac{L}{2} \cdot \|-\eta \nabla f(x_t)\|_2^2 && \text{(by definition of } x_{t+1}\text{)} \\ &= -\eta \cdot \|\nabla f(x_t)\|^2 + \frac{L}{2} \cdot \eta^2 \cdot \|\nabla f(x_t)\|_2^2 \\ &= -\frac{1}{2L} \|\nabla f(x_t)\|_2^2. && \text{(since } \eta = 1/L\text{)} \end{aligned}$$

Define the error of iteration t as $E_t = f(x_t) - f(x^*)$. Then, we have,

$$E_{t+1} = E_t + f(x_{t+1}) - f(x_t) \leqslant E_t - \frac{1}{2L} \cdot \|\nabla f(x_t)\|_2^2, \tag{1}$$

by the equation above. Recall that the optimum of f happen when the gradient is zero (by Lemma 6) and that the further we are from the optimum solution, the value of the gradient is larger; thus, the RHS above implies that we can reduce the error depending on how far we are currently from the optimum solution. We formalize this intuition in the following.

By Proposition 5 (by setting $x = x_t$ and $y = x^*$ an re-organizing and multiplying with minus one),

$$E_t = f(x_t) - f(x^*) \leqslant \langle \nabla f(x_t)\,,\, x_t - x^* \rangle \leqslant \|\nabla f(x_t)\|_2 \cdot \|x_t - x^*\|_2,$$

by Cauchy-Schwartz inequality. Moreover, since $f(x_t) \leqslant f(x_{t-1}) \leqslant f(x_0)$ (follows from Eq (1)), we can apply the bounded distance property of f and bound the second term of the RHS by D. This implies that

$$\|\nabla f(x_t)\|_2 \geqslant \frac{E_t}{D}.$$

Replacing this bound in Eq (1) gives us,

$$E_{t+1} \leqslant E_t - \frac{1}{2L} \cdot \frac{E_t^2}{D^2}.$$

Thus, we bounded the reduction in the error in terms of the current value of error itself. We can also bound the initial error as follows.

$$\begin{aligned} E_0 &= f(x_0) - f(x^*) \\ &\leqslant \langle \nabla f(x_0), x_0 - x^* \rangle && \text{(by convexity and applying Proposition 5 for } y = x^* \text{ and } x = x_0\text{)} \\ &\leqslant \|\nabla f(x_0)\| \cdot D && \text{(by Cauchy-Schwarz, and bounded distance property of } f\text{)} \\ &= \|\nabla f(x_0) - \nabla f(x^*)\| \cdot D && \text{(since } \nabla f(x^*) = 0\text{)} \\ &\leqslant L \cdot \|x_0 - x^*\| \cdot D && \text{(since } f \text{ has L-Lipschitz gradient)} \\ &\leqslant LD^2. && \text{(bounded distance property of } f\text{)} \end{aligned}$$

At this point, we are mostly done and the only remaining task is to solve the following recurrence: we want to show that $E_T \leqslant \varepsilon$ by solving the recurrence

$$\begin{aligned} E_{t+1} &\leqslant E_t - \frac{E_t^2}{2LD^2}, \\ E_0 &\leqslant LD^2. \end{aligned}$$

This is a slightly tricky recurrence to bound because of the type of dependence on E_t itself in the RHS. However, we can use a simple idea for approximately bounding this recurrence as follows.

Let us check the number of iterations till the error reduces by half successively. We define t_i as the number of iterations till the error reduces from $\frac{E_0}{2^i}$ to $\frac{E_0}{2^{i+1}}$. That is, t_i is the number of iterations for which the error lies between $\frac{E_0}{2^{i+1}} \leqslant E_t < \frac{E_0}{2^i}$. For any $k < t_i$, starting with the iteration t that has error $E_0/2^i$,

$$E_{t+k} \leqslant E_{t+k-1} - \frac{E_0^2}{4^{i+1}} \cdot \frac{k}{2LD^2},$$

using the lower bound on the value of E_{t+k-1} for all these iterations.

Using this equation we get that in $t_i = \frac{4LD^2 \cdot 2^i}{E_0}$ iterations, the value of error reduces from $E_0/2^i$ to $E_0/2^{i+1}$. For the error to reduce to ε, we need $\log(E_0/\varepsilon)$ such successive reductions of the error by factor 1/2. The total number of iterations to reduce the error to ε is thus,

$$t_1 + t_2 \ldots + t_{\log(E_0/\varepsilon)} \leqslant \frac{8LD^2}{E_0} \cdot 2^{\log(E_0/\varepsilon)} = \frac{8LD^2}{\varepsilon},$$

where the first inequality is because these iterations form a geometric series.

Thus we have the desired convergence in $T = O\left(\frac{LD^2}{\varepsilon}\right)$ iterations to reduce the error to ε. □

2.3 Projected Gradient Descent (PGD)

We have discussed unconstrained convex optimization, but what if we want to optimize a convex function over a convex set as we started with in Problem 1? Specifically, we want to solve

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

where K is a convex set, instead of

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

If we run gradient descent exactly as in Algorithm 1, we may end up at some iterate x_t such that $x_t \notin K$. To solve the constrained optimization problem, we need to ensure that $x_t \in K$ for all iterations. We can do that by just *projecting* any iterate back into K, formally defined as follows.

Definition 11. We define the closest point in a set K to some point $x \in \mathbb{R}^n$ as the **projection** of x onto K denoted by:

$$\mathsf{proj}_K(x) := \arg\min_{y \in K} \|x - y\|_2.$$

Note that finding the closest point in K to x is itself a convex optimization problem, and in general can be hard to solve. For projected gradient descent, we will assume we have some *oracle* that will give us the projection. These oracles can then be implemented based on the application.

Algorithm 2 (Projected Gradient Descent). For constrained convex optimization (Problem 1).

1. Let x_0 be an initial point in K and $\eta = \frac{1}{L}$ where L is the Lipschitz constant of ∇f.
2. For $t = 1$ to $T - 1$ iterations,
$$x_{t+1} = \mathsf{proj}_K(x_t - \eta \cdot \nabla f(x_t)).$$
3. Return x_T.

The following theorem states the same guarantee for projected gradient descent as that of Theorem 10 for gradient descent, while also replacing the bounded distance property to the more natural variant that only bound the distance for the original point.

Theorem 12. *Let f be a convex function over a convex set K, with L-Lipschitz gradient, and suppose initial point x_0 satisfies $\|x_0 - x^*\|_2 \leqslant D$. Then after $T = O\left(\frac{LD^2}{\varepsilon}\right)$ iterations, Algorithm 2 converges to x_T such that $f(x_T) \leqslant f(x^*) + \varepsilon$ with $\eta = \frac{1}{L}$.*

We will not prove this theorem for now and instead refer the interested reader to [Bub15, Section 3.2] for this proof.

Remark. We emphasize that to be able to use projected gradient descent for solving a constrained optimization problem, we need to have access to (*i*) a gradient oracle for the function f, and (*ii*) a projection oracle for the convex set K. Depending on the application at hand, such oracles may or may not be readily available to us – in fact, in some cases, the projection oracle might be simply as hard as the problem we would like to solve! For instance, consider applying PGD for linear programming directly, i.e., when $f(x) = \langle c\,, x\rangle$ for some $c \in \mathbb{R}^n$ and $K = \{x \mid Ax \leqslant b\}$ for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. In this case, projecting any arbitrary $y \in \mathbb{R}^m$ to the polyhedra K at the very least requires checking whether or not K is feasible, which as we saw is as hard as solving LPs in general.

3 An Application of PGD for Solving Maximum Flows

To conclude this lecture, we present an application of the projected gradient descent method for finding maximum flows in undirected and uncapacitated graphs, based on the paper by Lee, Rao and Srivastava [LRS13]. Recall that in this problem, we have an undirected graph $G = (V, E)$ with n vertices and m edges, and two vertices s, t, and capacity one over each edge; the goal is to find the maximum amount of flow possible in this graph G from s to t. In particular, any feasible flow $x \in \mathbb{R}^m$ must obey the two following constraints:

1. **Preservation of Flow Constraint:** For every vertex $v \in V \setminus \{s, t\}$,
$$\sum_{(u,v)\in E} x(u, v) = \sum_{(v,w)\in E} x(v, w).$$
2. **Edge Capacity Constraints:** For every $(u, v) \in E$,
$$x(u, v) \leqslant 1.$$

The value of the flow is then
$$\sum_{v} x(s, v).$$
We consider the feasibility version of problem again: For a given integer $F \geqslant 1$, does *there exists* a feasible flow of value at least F? A solution to this problem then easily generalizes to the original maximum flow problem using the standard binary search trick that we have covered multiple times in this course.

Theorem 13 (Weak version of [LRS13])**.** *There is an algorithms that given an undirected graph $G = (V, E)$ with two vertices s, t and unit capacity on each edge, and parameters $\delta \in (0, 1)$ and $F \geqslant 1$, either outputs a feasible flow of value at least $(1 - \delta) \cdot F$ in G or output that maximum flow in G is strictly less than F. The algorithm is based on projected gradient descent in $\widetilde{O}(\frac{m}{\delta^2 F})$ iterations and total time $\widetilde{O}\left(\frac{m^2}{\delta^2\cdot F}\right)$.*

The main result of [LRS13] improves this bound to an $\widetilde{O}(\frac{1}{\delta} \cdot \sqrt{\frac{m}{F}} \cdot m)$ time algorithm by replacing projected gradient descent with *accelerated (projected) gradient descent* which reduces the number of iterations quadratically to $\widetilde{O}(\frac{1}{\delta} \cdot \sqrt{\frac{m}{F}})$ iterations. By balancing the runtime of this algorithm against classical combinatorial approaches for maximum flow, one can obtain an algorithm with $\widetilde{O}(mn^{1/3} \cdot \delta^{-2/3})$ which is better than the MWU-based approach we discussed in Lecture 6 by some poly$(1/\delta)$ factors.

3.1 Flow Conservation and Capacity Polyhedra

We start by setting up the preliminaries for proving Theorem 13. We start by defining the polyhedron capturing the preservation of flow.

Definition 14. The **signed edge-incidence** matrix B of a undirected graph $G = (V, E)$ is an $n \times m$ matrix where columns represent edges and rows represent vertices, defined as follows: Arbitrarily direct the edges of G, then, for any (directed) edge $e = (v_i, v_j)$, set $B_{(i,e)} = -1$ and $B_{(j,e)} = 1$.

The reason we have defined matrix B as above becomes clear with the following lemma.

Lemma 15. *Let $b \in \mathbb{R}^n$ be a vector indexed by vertices of G with value $-F$ for vertex s, F for vertex t and 0 everywhere else. Then, the equation $Bx = b$ captures all the preservation of flow constraints.*

Proof. If $x \in \mathbb{R}^m$ is a (potentially feasible) flow over graph G, then the vector $Bx \in \mathbb{R}^n$ gives us the total flow at each vertex. Specifically, for any vertex $v \in V$, the index corresponding to v of vector Bx is just

$$\sum_{e=(u,v)} x_e - \sum_{e=(v,u)} x_e$$

which is exactly the incoming flow minus the outgoing flow for vertex v.

The flow preservation constraint tells us that the flow is preserved at each vertex but s, t. At vertex s, there is an extra F flow going out of s, and there is an extra F flow coming into t. Hence, any valid s-t flow must have incoming flow same as outgoing flow at all vertices in $V \setminus \{s, t\}$, the outgoing flow exceeding the incoming by F at vertex s and vice-versa at vertex t. The vector b captures all these constraints. □

$$\underbrace{\begin{array}{c|ccccc|} & e_1 & \cdots & e_j & \cdots & e_m \\ v_1 & 1 & 0 & \cdots & \cdots & \cdots \\ \vdots & 0 & -1 & \cdots & \cdots & \cdots \\ s & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ t & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ v_1 & \cdots & \cdots & \cdots & \cdots & \cdots \end{array}}_{B} \underbrace{\begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_m \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 0 \\ \vdots \\ -F \\ \vdots \\ F \\ \vdots \\ 0 \end{pmatrix}}_{b}$$

Figure 4: An illustration of Lemma 15.

Preservation of flow. This gives us our first polyhedron for the flow feasibility problem for the preservation of flow constraints. We use P to denote this polyhedron.

$$\text{Preservation of flow polyhedron } \mathbf{(P)} \text{ in } \mathbb{R}^m : \quad Bx = b. \tag{2}$$

Capacity constraint. The polyhedron (in fact polytope) for capacity constraints is simply the m-dimensional hypercube because we just want all the flow values over each edge to lie in the range $[-1, 1]$ (where negative flow means going in the opposite of (arbitrary) direction we imposed on edges in B). We use C to denote this polytope:

$$\text{Capacity constraint polytope } \mathbf{(C)} \text{ in } \mathbb{R}^m : -1 \leqslant x_e \leqslant 1 \quad \forall e \in E. \tag{3}$$

We are now ready to describe the steps of applying projected gradient descent for max flow with these constraints.

3.2 How to Apply PGD?

We are given two polytopes P (Eq (2)) and C (Eq (3)). We want to find some $x \in P \cap C$ if it exists, or declare that it does not exist otherwise. If we find such an x, then this means we found a flow assignment x of value F that satisfies the preservation constraints and the capacity constraints. So, this would be a solution to the feasibility problem.

In order to apply projected gradient descent, we need to answer the following questions:

1. We want a convex set to optimize a function f over a convex set K (as defined in Problem 1). How do we pick f and the convex set?
2. Can we compute $f(x)$ and $\nabla f(x)$ efficiently to be able to run Algorithm 2?
3. Does the function f have L-Lipschitz gradient for some $L \in \mathbb{R}$?
4. How do we pick the initial point x_0 so that it has bounded distance D for some $D \in \mathbb{R}$?
5. Given x_t, how should we compute $\mathsf{proj}_K(x_t - \frac{1}{L}\nabla f(x_t))$?
6. We want to know if $P \cap C$ is empty or not exactly. What should we do with the final error?

We get through these questions in the next steps to be able to use projected gradient descent for max flow.

3.3 Questions $1, 2$: The Function and the Convex Set

We have two polytopes P and C. Hence we have two choices for f and the convex set K to solve:

$$\min_{x \in P} \|\mathsf{proj}_C(x) - x\|_2^2 \qquad \text{or} \qquad \min_{x \in C} \|\mathsf{proj}_P(x) - x\|_2^2$$

For the first problem, $\|\mathsf{proj}_C(x) - x\|_2$ is just the distance of x from C for some point $x \in P$. We try to find the closest point in polytope P to polytope C. Similarly, in the second problem, $\|\mathsf{proj}_P(x) - x\|_2$ is just the distance of $x \in C$ from polytope P. We optimize for the closest point in polytope C to polytope P. If any of these are zero, then we have found a point that is in both polytopes P and C. Otherwise, we know that $P \cap C$ is empty, and our LP is not feasible.

We will choose to solve

$$\min_{x \in P} \|\mathsf{proj}_C(x) - x\|_2^2.$$

That is, we optimize over polytope P for function $f(x) := \|\mathsf{proj}_C(x) - x\|^2$. This is because it is quite easy to find the projection of any point x to polytope C, as we will show shortly.

Let us prove that our function $f : \mathbb{R}^m \to \mathbb{R}$ is indeed convex. For polytope C, due to its simple structure, the projection can be found easily.

Definition 16. Define the overflow function $\mathsf{overflow} : \mathbb{R} \to \mathbb{R}$ and the projection function $h : \mathbb{R} \to [0, 1]$ for each edge e and an assignment x_e to this edge in $x \in \mathbb{R}^m$ as:

$$\underset{\mathbb{R}\to\mathbb{R}}{\mathsf{overflow}(x_e)} := \begin{cases} 0 & -1 \leqslant x_e \leqslant 1, \\ x_e - 1 & x_e > 1, \\ -x_e - 1 & x_e < -1. \end{cases} \qquad \text{and} \qquad \underset{\mathbb{R}\to\mathbb{R}}{h(x_e)} := \begin{cases} x_e & -1 \leqslant x_e \leqslant 1, \\ 1 & x_e > 1, \\ -1 & x_e < -1. \end{cases}$$

Using these definitions, it is easy to see that $\mathsf{proj}_C(x)$ for each $x \in \mathbb{R}^m$ is simply:

$$\mathsf{proj}_C(x) = (h(x_{e_1}), h(x_{e_2}), \cdots, h(x_{e_m})).$$

We just pick the closest value for each dimension for any point outside the m-dimensional hypercube C. Then, it follows that for all cases of $x_e, |h(x_e) - x_e| = |\mathsf{overflow}(x_e)|$. Applying this to each dimension of $x \in \mathbb{R}^m$, we have,

$$f(x) = \|\mathsf{proj}_C(x) - x\|_2^2 = \sum_e \mathsf{overflow}(x_e)^2.$$

We can see that f is a convex function because it is the squared sum of multiple convex functions (each overflow function $\mathsf{overflow}$ for each dimension).

We can also compute the gradient of f for each dimension based on the above equation:

$$\nabla f(x)_e = \begin{cases} 0 & -1 \leqslant x_e \leqslant 1, \\ 2 \cdot (x_e - 1) & x_e > 1, \\ 2 \cdot (x_e + 1) & x_e < -1. \end{cases}$$

Thus, so far we identified the convex optimization problem we would like to solve, as well as how to evaluate the function and its gradient at any point.

3.4 Questions $3, 4$: Lipschitz Gradient and Bounded Distance Conditions

To be able to use Theorem 12 to analyze the use of gradient descent on function f over convex set P, we need to show that f has L-Lipschitz gradient for some $L \in \mathbb{R}$, and we also need to pick an initial point x_0 so that $\|x - x^*\|_2 \leqslant D$. This will be covered in this subsection.

We begin by showing that f does have L-Lipschitz gradient for $L = 4$.

Claim 17. $\nabla f(x)$ *is* 4*-Lipschitz.*

Proof. Fix any two $x, y \in \mathbb{R}^m$ and some edge $e \in E$. Suppose we have $x_e < -1$ and $y_e > 1$. So, $\nabla f(x)_e = 2x_e + 2$ and $\nabla f(y)_e = 2y_e - 2$. Then, we have

$$\begin{aligned} |\nabla f(x)_e - \nabla f(y)_e| &= |2x_e - 2y_e + 4| \\ &\leqslant 4 \cdot |x_e - y_e|. && \text{(as } |x_e - y_e| \geqslant 2\text{)} \end{aligned}$$

This bound is true even for other values of x_e and y_e because the function ∇f can only become closer. From this, for any $x, y \in \mathbb{R}^m$

$$\begin{aligned} \|\nabla f(x)_e - \nabla f(y)_e\|_2 &= \sqrt{\sum_e (\nabla f(x)_e - \nabla f(y)_e)^2} \leqslant \sqrt{\sum_e 16(x_e - y_e)^2} \\ &= 4\sqrt{\sum_e (x_e - y_e)^2} = 4 \cdot \|x - y\|_2, \end{aligned}$$

proving the claim. □

We will show that we can use **electrical flows**, introduced in Lecture 6, to find an initial point with good bounded distance D next, as well as in the later stages of the algorithm. In particular, we use the following somewhat *inaccurate* result.

Proposition 18 (cf. [ST14] – inaccurate version)**.** *There exists an $\widetilde{O}(m)$-time algorithm that given any undirected graph $G = (V, E)$ and demand vector $b \in \mathbb{R}^n$ such that $\sum_v b_v = 0$, solves the following problem:*

(i) Energy minimizing flow: find a flow $y \in \mathbb{R}^m$ as a solution to the following optimization problem:

$$\begin{aligned} \min_{y \in \mathbb{R}^m} \quad & \|y\|_2^2 \\ \text{subject to} \quad & By = b. \end{aligned}$$

(ii) Laplacian solver: find a vector of potentials $z \in \mathbb{R}^m$ as a solution to the following linear system:

$$(BB^\top) \cdot z = b.$$

(The accurate version solves both problems <u>*approximately*</u> *and not exactly – in particular constant approximation for the first one, and up to additive* poly$(1/n)$ *error in ℓ_2-norm for the second. But, for simplicity of exposition in this lecture, we assume both problems can be solved exactly; working with approximate solutions does not change the proof that much; see [LRS13] for more details).*

Using this proposition, we can show how to find a starting point x_0 as well.

Claim 19. *There exists a flow $x_0 \in \mathbb{R}^m$ such that $\|x_0 - x^*\|_2 \leqslant 2\sqrt{m}$ which can be found in $\widetilde{O}(m)$ time.*

Proof. We can write $\|x_0 - x^*\|_2 \leqslant \|x_0\|_2 + \|x^*\|_2$ by triangle inequality. Since x^* can at most assign 1 to all the edges due to the capacity constraint, i.e., $\|x^*\|_\infty \leqslant 1$, we have $\|x^*\|_2 \leqslant \sqrt{m}$.

Let x_0 to be electrical flow y obtained from G which we can compute in $\widetilde{O}(m)$ by using part (i) of Proposition 18. By the definition of x_0 having minimum ℓ_2-norm among all flows that satisfy preservation of flow (and not necessarily capacity constraint), we have, $\|x_0\|_2 \leqslant \|x^*\|_2 \leqslant \sqrt{m}$. Thus, we also have $\|x_0 - x^*\| \leqslant 2\sqrt{m}$, concluding the proof. □

This shows that we have all the conditions necessary for Theorem 12.

3.5 Question 5: Projection Step and Executing PGD

To be able to execute Algorithm 2, we need to find the projection in P so we optimize over polytope P. That is, given $y \in \mathbb{R}^m$ such that $y = x_t - \frac{1}{L}\nabla f(x_t)$, we want to find $\text{proj}_P(y)$.

We know that Bx represents a hyperplane. We will use the following fact about projections of points onto hyperplanes without proving it.

Fact 20. *Fix any affine subspace $Q := \{x \in \mathbb{R}^n \mid Ax = q\}$ for $A \in \mathbb{R}^{m\times n}$ and $q \in \mathbb{R}^m$. For any $y \in \mathbb{R}^n$,*

$$\mathsf{proj}_Q(y) = y - A^\top \cdot (AA^\top)^\dagger \cdot (Ay - q),$$

where for any matrix M, $M^\dagger$ denotes the **pseudo-inverse** *of M.*

Thus, for us, we are interested in projecting on the affine subspace $P = \{x \in \mathbb{R}^n \mid Bx = b\}$. We show that this can be done efficiently also

Claim 21. *For any point $y \in \mathbb{R}^m$, $\mathsf{proj}_P(y) = y - B^\top \cdot (BB^\top)^\dagger \cdot (By - b)$ can be found in $\widetilde{O}(m)$ time.*

Proof. First, it is easy to see that $z := (By - b) \in \mathbb{R}^n$ can be computed in $\widetilde{O}(m)$ time as matrix B only has $O(m)$ non-zero entries. The matrix $L := (BB^\top)$ is the Laplacian of graph G. To find $(BB^T)^+ z$ given $z \in \mathbb{R}^n$, we solve for the equation $Lw = z$, as $w = L^+ z$. This can be done in $\widetilde{O}(m)$ time with the Laplacian solver of Proposition 18 because the vector z satisfies the condition $\sum_v z_v = 0$ (i.e., belongs to the column space of L). This is because

$$\sum_v z_v = \left(\sum_v \left(\sum_{e=(u,v)} y_e - \sum_{e=(v,w)} y_e\right)\right) - \sum_v b_v = \sum_v \left(\sum_{e=(u,v)} y_e - \sum_{e=(v,w)} y_e\right)$$

(by the definition of matrix B and because $\sum_v b_v = 0$ by the definition of b)

$$= \sum_{e=(u,v)} (y_e - y_e) = 0$$

(as each edge appears once with a positive coefficient and once with a negative coefficient in B)

Finally, we can compute $B^T w$ in $O(m)$ time again based on the sparsity of B, concluding the proof. □

This gives us everything we require to run Algorithm 2 for maximum flows. In particular, we have that by Theorem 12 for any $\varepsilon > 0$, after running PGD for

$$O(\frac{LD^2}{\varepsilon}) = O(\frac{m}{\varepsilon})$$

iterations, we obtain a flow $\tilde{x} \in P$ such that

$$\sum_e \mathsf{overflow}(e)^2 = f(\tilde{x}) \leqslant f(x^*) + \varepsilon \leqslant \varepsilon, \tag{4}$$

where we used the fact that $f(x^*) = 0$ assuming there is a flow of value F in G. Moreover, each iteration takes $\widetilde{O}(m)$ time. We are still not done entirely because it is not clear how to "round" x into an actual flow.

3.6 Question 6: Fixing the Final Error

It remains to see how we will take care of the final error to check if $P \cap C$ is empty or not. To do so, we need the following combinatorial lemma about removing overflows in capacity constraints over edges.

Lemma 22. *Suppose f is an s-t flow with value F that obeys the flow preservation constraint but not necessarily the capacity constraints. Then, there exists an s-t feasible flow $\tilde{f}$ with value $F - \sum_{x_e} \mathsf{overflow}(x_e)$. Furthermore, this flow can be found in $O(m \log n)$ time.*

Lemma 22 is provided in [LRS13] as the Overflow Drainage Lemma using dynamic trees of Sleator and Tarjan [ST81]. The proof, while not complicated, is beyond the scope of this course. But, to give the intuition behind this lemma, here we give its proof for the case when f is *integral*.

Proof of an easy version of Lemma 22 for integral flows. Consider the DAG D corresponding to the flow f in G. Add a new vertex z to D and for any directed edge $(u, v) \in D$, connect u to z with flow of $\mathsf{overflow}(e)$ and connect z to v again with flow of $\mathsf{overflow}(e)$; then, subtract $\mathsf{overflow}(e)$ from the flow f over edge e. Let f' be this new flow and note that f' still satisfies preservation of flow and also capacity constraint on all edges except for the ones incident on z. We are now going to remove all the flow incident on z.

Firstly, run DFS from z to s on backward edges and whenever you find a path, remove all edges of the path from D (and reduce the flow on the edge coming to z by one). Continue as long as all incoming edges of z are removed from the graph. This step requires $O(m)$ time as we never need to visit an edge more than once. Do the same from z to t on forward edges to remove all outgoing edges of z as well. Throughout this process, we remove exactly $\sum_e \mathsf{overflow}(e)$ paths while keeping the preservation of flow and thus the remaining graph corresponds to a flow $\tilde{f}$ with value $F - \sum_{x_e} \mathsf{overflow}(x_e)$ as desired. □

To conclude the proof of Theorem 13, we would like to apply Lemma 22 to the flow $\tilde{x}$ we obtained in Eq (4). The only problem is that Eq (4) bounds the ℓ_2-norm of the vector $\mathsf{overflow} \in \mathbb{R}^m$, while we need to bound its ℓ_1-norm to invoke Lemma 22. This is taken care of by the following lemma.

Lemma 23. *Let $\delta \in (0, 1)$ and $\varepsilon := \delta^2 \cdot F$. Suppose we have a vector $\tilde{x} \in P$ (i.e., in the preservation of flow polyhedron) satisfying Eq (4) for the parameter ε. Then, we can round $\tilde{x}$ in $\widetilde{O}(m)$ time to a feasible unit-capacity flow $\tilde{f} \in \mathbb{R}^m$ with value at least $(1 - 4\delta) \cdot F$.*

Proof. By Eq (4) and the choice of ε, we have that

$$\sum_e \mathsf{overflow}(x_e)^2 \leqslant \delta^2 \cdot F.$$

We classify the edges into two types and remove the overflow in different ways for each of them.

(i) Severely congested edges $H := \{e \mid \mathsf{overflow}(x_e) > \delta\}$ and, (*ii*) Remaining edges: $R := \{e \mid \mathsf{overflow}(x_e) \leqslant \delta\}$.

We can handle the second type of edges to satisfy the capacity constraints by just rescaling the entire flow on every edge by a factor of $\frac{1}{1+\delta}$ to get a new flow x'. In other words, let $x' = \frac{x}{1+\delta}$ and thus we have any edge $e \in R$ now satisfy the unit capacity, and the value of the flow is

$$\sum_{e \ni s} x'_e = \sum_{e \ni s} \frac{x_e}{1+\delta} = \frac{F}{1+\delta}.$$

For the first type of edges, note that, since for all $e \in H$, $\mathsf{overflow}(e) > \delta$, we have,

$$\delta \cdot \sum_{e \in H} \mathsf{overflow}(x_e) \leqslant \sum_{e \in H} \mathsf{overflow}(x_e)^2 \leqslant \sum_{e} \mathsf{overflow}(x_e)^2 \leqslant \delta^2 \cdot F.$$

Hence, we must have that,

$$\sum_{e \in H} \mathsf{overflow}(x_e) \leqslant \delta F.$$

This inequality also holds for the rescaled flow x', since we do not make our overflow larger on any edge $e \in E$. This implies that the flow x' has value $F/(1+\delta)$ and has total overflow at most $\delta \cdot F$. As such, by Lemma 22, we can round this flow in $\widetilde{O}(m)$ time to a flow $\tilde{f}$ satisfying preservation of flow and unit-capacity constraint with value at least

$$\underbrace{\frac{F}{1+\delta}}_{\text{rescaled flow}} - \delta \cdot F \geqslant (1-4\delta) \cdot F,$$

concluding the proof. □

Theorem 13 now follows from Eq (4) and Lemma 23 as we can set $\varepsilon = \delta^2 \cdot F$ to obtain the desired flow in $O(\frac{m}{\delta^2 \cdot F})$ iterations each taking $\widetilde{O}(m)$ time (if the value of Eq (4) is larger than ε at the end of running PGD, we can safely return G does not have a flow of value at least F). Rescaling $\delta \leftarrow \delta/4$ also implies that the final answer is a $(1-\delta)$-approximation.

References

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 8(3-4):231–357, 2015. 8

[LRS13] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *STOC '13*, 2013. 8, 12, 13

[ST81] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Symposium on the Theory of Computing*, 1981. 13

[ST14] Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.*, 35(3):835–885, 2014. 11