CS 435, 2018
Lecture 3, Date: 8 March 2018
Instructor: Nisheeth Vishnoi

Gradient Descent

This lecture introduces *Gradient Descent* – a meta-algorithm for unconstrained minimization. Under convexity of the objective, we prove a convergence time bound when the gradient of the function is Lipschitz. Subsequently, it is shown how to use this continuous optimization method to come up with a fast algorithm for a discrete optimization problem: computing maximum flows in a graph.

Contents

1 The Setting

In this lecture we mostly focus on unconstrained convex optimization, i.e., on solving problems of them form

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1}$$

for a convex function f. As always, let us denote the optimal value $y^\star := \min_{x \in \mathbb{R}^n} f(x)$. Let us briefly recap the main points of the previous lecture, where computational aspects of convex programming were discussed.

1. **How is f given in** (1)**?** There are essentially two ways to give a computer access to a function f. The first one is to specify f explicitly by a short list of rational parameters. For instance if we talk about quadratic functions $f(x) = x^\top Ax + b^\top x$, then it is enough to provide $A \in \mathbb{Q}^{n \times n}$ and $b \in \mathbb{Q}^n$ as input.

   Alternatively one can consider *oracle access* to f. In such a case the program for solving (1) is given access to a primitive which can

   - given $x \in \mathbb{Q}^n$ output $f(x)$ – 0-order oracle,
   - given $x \in \mathbb{Q}^n$ output $\nabla f(x)$ – 1st-order oracle,
   - or more generally, given $x \in \mathbb{Q}^n$ output $D^k f(x)$ – kth-order oracle.

   Note that such a way of representing f is s weaker then by giving its compact description, as querying a finite (even a very large) number of points does not give us full information about f. However, at the same time, it is less restrictive.

2. **What is a solution of** (1)**?** Our notion of a solution to a program of the form (1) is an algorithm (can be randomized) which given access to f and given $\varepsilon > 0$ outputs a point $x \in \mathbb{R}^n$ such that
   $$f(x) \leq y^\star + \varepsilon.$$
   We measure the running time of such an algorithm as a function of $\frac{1}{\varepsilon}$ and either the bit complexity of the description of f, or the number of oracle calls to f or its higher order derivatives, depending on the model. We regard such an algorithm as a polynomial time algorithm if its running time is polynomial in the bit complexity of f and $\log \frac{1}{\varepsilon}$ and it outputs the right answer with high probability.

In this lecture we mostly work in the (1st-order) black-box (oracle) model and arrive at algorithms with running times proportional to $\frac{1}{\varepsilon}$. Although these are not polynomial time algorithms, we also prove that in the considered black-box setting one cannot in general obtain polynomial time algorithms.

2 Gradient Descent

In this lecture we consider the problem (1) when only a 1st-order oracle is given (i.e., we can query the gradient at any point). We introduce perhaps the simplest but extremely powerful *gradient descent* method to solve a convex program. Actually, it is not a single method, but a general framework with many possible realizations. In this lecture and the next, we describe some concrete variants and analyze their performance. The performance guarantees that we are going to obtain will depend on assumptions that we make about f. We describe the core ideas of the gradient descent methods in the *unconstrained setting*, i.e., $K = \mathbb{R}^n$. In Section 3.4 we discuss the constrained setting.

The general scheme of gradient descent can be summarized as follows

1. Choose a starting point $x_1 \in \mathbb{R}^n$.
2. Suppose $x_1, \ldots, x_t$ are computed.
3. Choose x_{t+1} as a linear combination of x_t and $\nabla f(x_t)$.
4. Stop once a certain stopping criterion is met and output the last iterate.

Let us define x_t to be the point chosen in Step 3 of this algorithm at the t-th iteration and T to be the total number of iterations performed. The running time for the algorithm is $O(T \cdot M(x))$ where $M(x)$ is the time of each update. Normally, the update time $M(x)$ is something which cannot be really optimized below a certain level and hence the main goal is to design methods with T as small as possible.

2.1 Why descend along the gradient?

In what follows we motivate one possible method for choosing the next iteration point x_{t+1} from x_t. Since the process we are about to describe can use only the local information about x, it makes sense to pick a direction which locally provides the largest drop in the function value. More formally, we would like to pick a unit vector u (i.e., a vector $u \in \mathbb{R}^n$ such that $\|u\| = 1$) for which the drop after an δ-step (for a tiny $\delta > 0$) in direction u is the largest, i.e., which maximizes

$$f(x) - f(x + \delta u),$$

over all $u \in \mathbb{R}^n$ with $\|u\| = 1$. In the above, we would like to work with δ infinitesimally small, and thus formally solve the following optimization problem

$$\max_{\|u\|=1} \left[\lim_{\delta \to 0^+} \frac{f(x) - f(x + \delta u)}{\delta} \right].$$

By the Taylor approximation of f at x, it can be seen that the expression inside is simply the directional derivative of f at x in direction u and thus we obtain

$$\max_{\|u\|=1} \left[- \langle \nabla f(x), u \rangle \right]. \tag{2}$$

We claim that the above is maximized at $u^\star := -\frac{\nabla f(x)}{\|\nabla f(x)\|}$. Indeed, consider any point u with norm 1. From the Cauchy-Schwarz inequality we have

$$\langle \nabla f(x), u \rangle \leq \|\nabla f(x)\| \, \|u\| = \|\nabla f(x)\| \,,$$

and the equality is attained at $u = u^\star$. Thus, moving in the direction of the negative gradient at the current point can be viewed as an instantaneously good greedy strategy – also called the gradient flow of f.

$$\frac{dx}{dt} = -\frac{\nabla f(x)}{\|\nabla f(x)\|}.$$

However, to implement the strategy above on a computer either we should consider discretizations of the differential equations above. Since we assume first order access to f, a natural discretization is the so-called forward-Euler:

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\|\nabla f(x_t)\|}. \tag{3}$$

where $\alpha > 0$ is the "step length", i.e., how far do we want to move along $u^\star$. Since $\frac{1}{\|\nabla f(x_t)\|}$ is only a normalization constant we can omit it and arrive at the following gradient descent update rule

$$x_{t+1} = x_t - \eta \nabla f(x_t), \tag{4}$$

where, again, $\eta > 0$ is a parameter – the step length which we might also make depend on the time t or the point x_t.

2.2 Assumptions on the function, gradient, starting point

While moving in the direction of the negative gradient is a good instantaneous strategy, it is far from clear how big a step to take. Ideally, we would like to take big steps hoping for a smaller number of iterations, but there is also a danger. The view we have of the function at the current point is only local and our best guess of the function is a linear function. Thus, the function can change and bend quite dramatically and taking a long step can lead us to a large error. This is one of the many issues that can arise.

To bypass this and related problems, it is customary to make assumptions on the function in terms of parameters such as the Lipschitz constant of the function f or its gradient. These are natural measures of how "complicated" f is and thus how hard to optimize f could be using a first order oracle. Therefore these "regularity parameters" will often show up in the running time guarantees for the optimization algorithms we develop. Moreover, when designing methods with oracle access to f only, it is natural to provide as input an additional point $x_0 \in \mathbb{R}^n$ which is not "too far" from an optimal solution, as otherwise it is not even clear where should such an algorithm start its search. More formally, below we list the kind of assumptions which show up is the theorems.

1. **Lipschitz Gradient.** For every $x, y \in \mathbb{R}^n$ we have
$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|,$$
where L is a possibly large but finite quantity. This is also sometimes referred to as L-smoothness of f. This condition ensures that around x, the gradient changes in a controlled manner and, thus, the gradient flow trajectories can only "bend" in a controlled manner. The smaller the L is, the larger the step size we can think of taking.

2. **Bounded Gradient.** For every[1] $x \in \mathbb{R}^n$ we have
$$\|\nabla f(x)\| \leq G,$$
where $G \in \mathbb{Q}$ is a possibly large but finite quantity. Note that this implies that f is a G-Lipschitz function. This quantity essentially controls how quickly the function can go towards infinity. The smaller G is, the slower this growth is.

3. **Good Initial Point.** A point $x_1 \in \mathbb{Q}^n$ is provided such that $\|x_1 - x^\star\| \leq D$, where $x^\star$ is some[2] optimal solution to (1).

We now state the main result that we prove in this lecture.

Theorem 1. *There is an algorithm, which given* 1*st-order oracle access to a convex function $f : \mathbb{R}^n \to \mathbb{R}$ along with a bound L on the Lipschitz constant of its gradient, an initial point $x_1 \in \mathbb{R}^n$ such that $\max\{\|x - x^\star\| : f(x) \leq f(x_1)\} \leq D$ (where $x^\star$ is an optimal solution to $\min_{x \in \mathbb{R}^n} f(x)$) and an $\varepsilon > 0$, outputs a point $x \in \mathbb{R}^n$ such that $f(x) \leq f(x^\star) + \varepsilon$.*

The algorithm makes $T = O\left(\frac{LD^2}{\varepsilon}\right)$ queries to the 1*st order oracle for f and performs $O(nT)$ arithmetic operations.*

We also note that while in this lecture we only prove the theorem in the above variant, one can alternatively a weaker statement: $\|x_1 - x^\star\| \leq D$ to arrive at the same conclusion.

In the next lecture we will prove the following theorem and its generalizations to non-Euclidean norms.

Theorem 2. *There is an algorithm, which given* 1*st-order oracle access to a convex function $f : \mathbb{R}^n \to \mathbb{R}$ along with a bound G on its gradient, an initial point $x_1 \in \mathbb{R}^n$ with $\|x_0 - x^\star\| \leq D$ (where $x^\star$ is an optimal solution to $\min_{x \in \mathbb{R}^n} f(x)$) and an $\varepsilon > 0$, outputs a point $x \in \mathbb{R}^n$ such that $f(x) \leq f(x^\star) + \varepsilon$.*

The algorithm makes $T = O\left(\left(\frac{DG}{\varepsilon}\right)^2\right)$ queries to the 1*st order oracle for f and performs $O(nT)$ arithmetic operations.*

[1] We assume here that the bound on the gradient is valid for all $x \in \mathbb{R}^n$. One might often relax it to just x over a suitably large set $X \subseteq \mathbb{R}^n$ which contains x_0 and an optimal solution $x^\star$. This requires to prove that the algorithm "stays in X" for the whole computation.

[2] There might be more than just one optimal solution, here it is enough that the distance to any of them is small.

Before going any further, one might wonder if it is reasonable to assume knowledge of such parameters as G, L, D of the function f in Theorems 1 and 2, especially when the access to the function is black-box only. While for D, it is often possible to argue sensible bounds on the solution, finding G or L might be more difficult. However, one can often try to set these values adaptively. As an example, it is possible to start with a guess G_0 (or equivalently L_0) and update the guess given the outcome of the gradient descent. Should the algorithm fail for this value, we can try doubling the constant $G_1 = 2G_0$ and so on.

In practice, especially in machine-learning applications, these quantities are known up to a good precision and are often considered to be constants. Under such an assumption the number of oracle calls performed by the respective algorithms are $O\left(\frac{1}{\varepsilon^2}\right)$ and $O\left(\frac{1}{\varepsilon}\right)$ and do not depend on the dimension n. Therefore, such algorithms are often referred to as *dimension-free*, their convergence rate, as measure in the number of *iterations* (as clarified later) does not depend on n, only on certain regularity parameters of f. In TCS applications, an important aspect is to find formulations where these quantities are small and this is often the key challenge.

3 Analysis when the Gradient is Lipschitz

This section is devoted to proving Theorem 1. We start by stating an appropriate variant of the gradient descent algorithm and subsequently provide its analysis. In the pseudocode below the meaning of D and G is as in the statement of Theorem 1.

3.1 The algorithm

Gradient Descent Algorithm for Lipschitz Gradient:
Input: 1st-order oracle for f, bound on the Lipschitz constant of the gradient L, bound on the distance to optimal solution D, initial point $x_1 \in \mathbb{Q}^n$, and $\varepsilon > 0$.

Output: A point x such that $f(x) - f(x^\star) \leq \varepsilon$.

1. Let $T = O\left(\frac{LD^2}{\varepsilon}\right)$.
2. Let $\eta = \frac{1}{2L}$.
3. Repeat for $t \in \{1, \ldots, T-1\}$
   - $x_{t+1} = x_t - \eta \nabla f(x_t)$.
4. Output x_T.

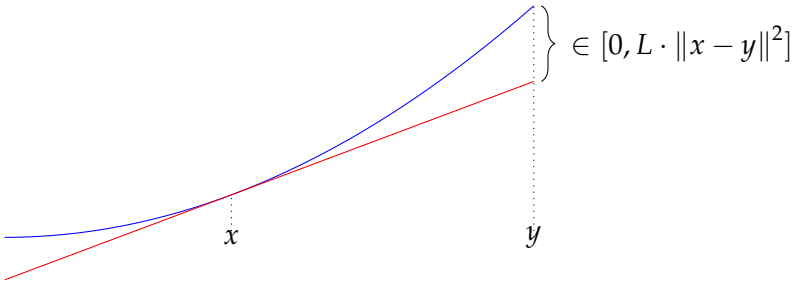

Figure 1: The gap between the function value $f(y)$ and the 1st order approximation at x is non-negative and bounded from above by a quadratic function $L\|x-y\|^2$ when f has L-Lipschitz gradient.

3.2 The analysis

Before we start the proof of Theorem 1 we need to establish an important lemma explaining the significance of our assumption.

Lemma 3. *For every differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ that satisfies $\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|$ for every $x, y \in \mathbb{R}^n$, it holds for any $x, y \in \mathbb{R}^n$*

$$f(y) - f(x) - \langle \nabla f(x), y - x \rangle \leq L \cdot \|x-y\|^2 .$$

This means in particular that if f is convex, then the distance from $f(y)$ to its first-order Taylor approximation at x is between 0 and $L \cdot \|x-y\|^2$, see Figure 1 for an illustration.

Proof. Consider a univariate function $g(t) = f((1-t)x + ty)$ for $t \in [0,1]$. We have $g(0) = f(x)$ and $g(1) = f(y)$. Since g is differentiable, from the Fundamental Theorem of Calculus we have:

$$\int_0^1 g'(t)dt = f(y) - f(x).$$

Since $g'(t) = \langle \nabla f((1-t)x + ty), y - x \rangle$ we arrive at

$$\begin{aligned}
f(y) - f(x) &= \int_0^1 \langle \nabla f((1-t)x + ty), y - x \rangle \, dt \\
&= \int_0^1 \langle \nabla f(x), y - x \rangle \, dt + \int_0^1 \langle \nabla f((1-t)x + ty) - \nabla f(x), y - x \rangle \, dt \\
&\leq \langle \nabla f(x), y - x \rangle + \int_0^1 \|\nabla f((1-t)x + ty) - \nabla f(x)\| \, \|y - x\| \, dt \\
&\leq \langle \nabla f(x), y - x \rangle + \|x - y\| \int_0^1 L \|t(y - x)\| \, dt \\
&\leq \langle \nabla f(x), y - x \rangle + L \|x - y\|^2 .
\end{aligned}$$

Where we have used the Cauchy-Schwarz inequality and the L-Lipschitzness of the gradient of f. □

Proof of Theorem 1: Let us examine the evolution of the error as we iterate. Use Lemma 3 to get

$$\begin{aligned} f(x_{t+1}) - f(x_t) &\leq \langle \nabla f(x_t), x_{t+1} - x_t \rangle + L \cdot \|x_{t+1} - x_t\|^2 \\ &= -\eta \|\nabla f(x_t)\|^2 + L\eta^2 \|\nabla f(x_t)\|^2 . \end{aligned}$$

Just like before, we wish to obtain a tight inequality. To accomplish that we minimize over η for which we find a value of $\frac{1}{2L}$. Substituting it, we get

$$f(x_{t+1}) - f(x_t) \leq -\frac{1}{4L} \|\nabla f(x_t)\|^2 . \tag{5}$$

Intuitively, this means that, at every step, either the gradient is large and we are making good progress, or it is small, which means that we are already close to the optimum.

Let us now denote $R_t := f(x_t) - f(x^\star)$, we would like to bring R_t below ε. Start by noting that $R_1 \leq LD^2$, this follows from L-smoothness of f and the bound $\|x_1 - x^\star\| \leq D$. Further, from (5) we know that $R_t - R_{t+1} \geq \frac{1}{4L} \|\nabla f(x_t)\|^2$, moreover by convexity and the Cauchy-Schwarz inequality,

$$R_t \leq f(x_t) - f(x^\star) \leq \langle \nabla f(x_t), x_t - x^\star \rangle \leq \|\nabla f(x_t)\| \cdot \|x_t - x^\star\|$$

and if we bound $\|x_t - x^\star\|$ by D (this is valid since since $f(x_t) \leq f(x_1)$; the method produces smaller points as it goes), then we get

$$\|\nabla f(x_t)\| \geq \frac{R_t}{D},$$

and, by (5),

$$R_t - R_{t+1} \geq \frac{R_t^2}{4LD^2}. \tag{6}$$

Thus to bound the number of iterations to reach ε we need to solve the following calculus problem: given a sequence of numbers $R_1 \geq R_2 \geq R_3 \geq \cdots \geq 0$, with $R_1 \leq LD^2$ and satisfying the recursive bound (6), find a bound on T for which $R_T \leq \varepsilon$.

Before we give a formal proof that T can be bounded by $O\left(\frac{LD^2}{\varepsilon}\right)$ let us provide some intuitions by analyzing a continuous time analogue of recursion (6) – it gives rise to the following differential equation:

$$\frac{d}{dt} R(t) = -\alpha R(t)^2,$$

where $R : [0, \infty) \to \mathbb{R}$ with $R(0) = LD^2$ and $\alpha = \frac{1}{LD^2}$. To solve it, first rewrite the above as

$$\frac{d}{dt}\left[\frac{1}{R(t)}\right] = \alpha,$$

and hence we obtain

$$R(t) = \frac{1}{R(0)^{-1} + \alpha t},$$

from which we deduce that to reach $R(t) \leq \varepsilon$, we need to take

$$t \geq \frac{1 - \frac{\varepsilon}{R(0)}}{\varepsilon\alpha} \approx \Theta\left(\frac{LD^2}{\varepsilon}\right).$$

To formalize this argument in the discrete setting (where $t = 1, 2, \ldots$), the idea is to first estimate the number of steps for R_t to drop below $R_1/2$, then to go from $R_1/2$ to $R_1/4$, then from $R_1/4$ to $R_1/8$ and so on until ε.

In general, given (6) it is easy to see that to go from R_t to $R_t/2$ one needs to make at least k steps, where

$$k \cdot \frac{(R_t/2)^2}{4LD^2} \geq R_t/2,$$

in other words, k needs to be at least $\left\lceil \frac{16LD^2}{R_t} \right\rceil$. Let $r := \left\lceil \log \frac{R_1}{\varepsilon} \right\rceil$. By repeated halving (need to repeat r times) to get from R_1 to ε the required number of steps is at most

$$\sum_{i=0}^{r} \left\lceil \frac{16LD^2}{R_1 \cdot 2^{-i}} \right\rceil \leq r + 1 + \sum_{i=0}^{r} 2^i \frac{16LD^2}{R_1} \leq (r+1) + 2^{r+1}\frac{16LD^2}{R_1} = O\left(\frac{LD^2}{\varepsilon}\right).$$

□

3.3 Lower bound and Nesterov acceleration

One can ask whether the proposed method, which performs $O(\varepsilon^{-1})$ iterations is optimal. Consider now a general model for 1st order black-box minimization which includes gradient descent and many related algorithms. The algorithm is given $x_1 \in \mathbb{R}^m$ and access to a gradient oracle for a convex function $f : \mathbb{R}^n \to \mathbb{R}$. It produces a sequence of points: $x_1, x_2, \ldots, x_T$ such that

$$x_t \in x_1 + \text{span}\{\nabla f(x_1), \ldots, \nabla f(x_{t-1})\} \tag{7}$$

i.e., the algorithm might move only in the subspace spanned by the gradients at previous iterations. Note that gradient descent clearly follows this scheme as

$$x_t = x_1 - \sum_{j=1}^{t-1} \eta \nabla f(x_j).$$

We do not restrict the running time of one iteration of such an algorithm, in fact we allow it to do an arbitrarily long calculation to compute x_t from $x_1, \ldots, x_{t-1}$ and the corresponding gradients. In this model we are interested only in the number of iterations. The lower bound, first established in [3] (see also [5] and [1]) is as follows.

Theorem 4 (Lower bound – Lipschitz Gradient [3])**.** *Consider any algorithm for solving the convex unconstrained minimization problem* $\min_{x\in\mathbb{R}^n} f(x)$ *in the model as in* (7)*, when* f *has Lipschitz gradient (with a constant* L*) and the initial point* $x_1 \in \mathbb{R}^n$ *satisfies* $\|x_1 - x^\star\| \leq D$*. There is a function* f *such that for any* $1 \leq T \leq \frac{n}{2}$ *it holds*

$$\min_{1\leq i\leq T} f(x_i) - \min_{x\in\mathbb{R}^n} f(x) \geq \frac{LD^2}{T^2}.$$

The above theorem translates to a lower bound of $\Omega\left(\frac{1}{\sqrt{\varepsilon}}\right)$ iterations to reach an ε-optimal solution. This lower bound does not quite match the upper bound of $\frac{1}{\varepsilon}$ established in Theorem 1. Therefore one can ask: *Is there a method which matches the above* $\frac{1}{\sqrt{\varepsilon}}$ *iterations bound?* And, surprisingly, the answer is YES! This can be achieved using the so called Accelerated Gradient Descent developed by Nesterov [4].

Theorem 5. *There is an algorithm, which given* 1*st-order oracle access to a convex function* $f : \mathbb{R}^n \to \mathbb{R}$ *along with a bound* L *on the Lipschitz constant of its gradient, an initial point* $x_1 \in \mathbb{R}^n$ *such that* $\|x_1 - x^\star\| \leq D$ *(where* $x^\star$ *is an optimal solution to* $\min_{x\in\mathbb{R}^n} f(x)$*) and an* $\varepsilon > 0$*, outputs a point* $x \in \mathbb{R}^n$ *such that* $f(x) \leq f(x^\star) + \varepsilon$*.*

The algorithm makes $T = O\left(\frac{\sqrt{L}D}{\sqrt{\varepsilon}}\right)$ *queries to the* 1*st order oracle for* f *and performs* $O(nT)$ *arithmetic operations.*

3.4 Constrained setting – projection

So far we have discussed the unconstrained optimization problem (1), however the same method can be also extended to the constrained setting, when we would like to solve

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

for a convex subset $K \subseteq \mathbb{R}^n$.

When we apply the gradient descent method, the next iterate x_{t+1} might fall outside of the convex body K. In this case we need to project it back onto K: to find the point in K with the minimum distance to x, and take it to be our new iterate instead:

$$x_{t+1} = \mathrm{proj}_K\left(x_t - \eta_t \cdot \nabla f(x_t)\right).$$

The convergence rates remain the same (the proof just carries over to this new setting). However, depending on K, the projection may or may not be difficult (or computationally expensive) to perform. More precisely, as long as the algorithm has access to an oracle which, given a query point x, returns the projection $\mathrm{proj}_K(x)$ of x onto K, we have the following analogue of Theorem 1:

Theorem 6. *There is an algorithm, which given* 1*st-order oracle access to a convex function* $f : \mathbb{R}^n \to \mathbb{R}$ *along with a bound* L *on the Lipschitz constant of its gradient, oracle access to a projection operator* proj_K *onto a convex set* $K \subseteq \mathbb{R}^n$*, an initial point* $x_1 \in \mathbb{R}^n$ *such that* $\max\{\|x - x^\star\| : x \in K, f(x) \leq f(x_1)\} \leq D$ *(where* $x^\star$ *is an optimal solution to* $\min_{x \in K} f(x)$*) and an* $\varepsilon > 0$*, outputs a point* $x \in K$ *such that* $f(x) \leq f(x^\star) + \varepsilon$*.*

The algorithm makes $T = O\left(\frac{LD^2}{\varepsilon}\right)$ *queries to the* 1*st order oracle for* f *and the projection operator* proj_K *and performs* $O(nT)$ *arithmetic operations.*

4 Application: Computing Maximum Flows in Graphs

As an application of the results we developed in the previous section, we present an algorithm to solve the problem of computing the maximum $s - t$ flow in an undirected graph. This is our first non-trivial example of a discrete optimization problem for which we use continuous optimization methods to solve. This general viewpoint has resulted to a number of breakthroughs for fundamental discrete optimization problems recently and, today, the fasted known methods for several problems are through continuous optimization formulations. The trick is often in coming up with the right formulation.

4.1 The $s - t$-maximum flow problem

Given an undirected, unweighted graph $G = (V, E)$ with (for simplicity) unit capacities on all edges and two vertices $s, t \in V$ (the source and the sink) we would like to know what is the maximum amount of flow $F^\star$ one can send from s to t without violating the capacity constraints on edges. Intuitively one can think of edges in G as pipes through which at most one unit of flow is allowed to be transported per second (in either direction) and the question is how much flow can be transported (per second) from s to t. At all vertices other than s and t, the incoming flow should be equal to the outgoing flow.

More formally, let us introduce the incidence matrix $B \in \mathbb{R}^{n \times m}$ of the graph, where $n = |V|$ and $m = |E|$. For that assume that every vertex is assigned a label between 1 and n and similarly edges are indexed by numbers from 1 to m. For every edge $i \in \{1, 2, \ldots, m\}$ between vertices $u, v \in \{1, 2, \ldots, n\}$, B contains a column b_i corresponding to $uv \in E$ of the form

$$b_i = e_u - e_v \in \mathbb{R}^n,$$

where $\{e_w\}_{w \in [n]}$ are standard basis vectors for $\mathbb{R}^n$. Note that the edge $uv \in E$ is undirected but b_i depends on the order of u, v – this choice is arbitrary and it does not matter whether we use b_{uv} or $-b_{uv}$ in the matrix B.

Let $b = e_s - e_t$ (where $s, t \in [n]$ are the source and sink respectively), then the maximum flow problem can be then formulated formally as

$$\begin{aligned} \max_{x \in \mathbb{R}^E,\, F \geq 0} \quad & F \\ \text{s.t.} \quad & Bx = Fb \\ & \|x\|_\infty \leq 1. \end{aligned} \tag{8}$$

Above we are maximizing the flow value F over all flow vectors $x \in \mathbb{R}^E$, which satisfy flow conservation: $Bx = Fb$, and satisfy capacity constraints: $\|x\|_\infty \leq 1$.

4.2 The main result

Theorem 7. *There is an algorithm, which given an undirected graph G with unit capacities, two vertices s, t and an $\varepsilon > 0$, finds an $s-t$ flow of value at least $(1-\varepsilon)F^\star$ in time*[3] $\widetilde{O}\left(\varepsilon^{-1}\frac{m^{5/2}}{F^\star}\right)$.

We do not give a full proof of the above, only sketch the overall idea and leave certain steps as exercises.

4.3 A formulation as an unconstrained convex program

We start by reformulating (8) as an unconstrained convex optimization problem. Note that it is already convex, however we would like to avoid complicated constraints such as "$\|x\|_\infty \leq 1$". To this end, note that instead of maximizing F we might just make a guess for F and ask if there is a flow x with value F obeying the capacity constraints. If we could solve such a decision problem efficiently, then we could solve (8) up to a good precision by performing a binary search over F. Thus it is enough, for a given $F \in \mathbb{Q}_{\geq 0}$ to solve the decision problem

$$\begin{aligned} \text{find} \quad & x \in \mathbb{R}^E \\ \text{s.t.} \quad & Bx = Fb, \\ & \|x\|_\infty \leq 1. \end{aligned} \tag{9}$$

If we denote

$$H_F := \{x \in \mathbb{R}^m : Bx = Fb\}$$

and

$$B_{m,\infty} := \{x \in \mathbb{R}^m : \|x\|_\infty \leq 1\}$$

then the set of solutions above is the intersection of these two convex sets and hence the problem really asks a question of the form: "is $K \neq \emptyset$?" for a certain convex set K. As we would like to apply the gradient descent algorithm we need to state it as an minimization problem. It turns out that the most convenient way to do that is to think of (9) as minimizing the distance of x to $B_{m,\infty}$ over $x \in H_F$. For that, let P be the projection operator $P : \mathbb{R}^m \to B_{m,\infty}$ given as

$$P(x) := \operatorname{argmin}\{\|x - y\| : y \in B_{m,\infty}\}.$$

The final form of (8) we would like to consider is

$$\begin{aligned} \min \quad & \|x - P(x)\|^2 \\ \text{s.t.} \quad & Bx = Fb. \end{aligned} \tag{10}$$

[3]The notation $\widetilde{O}(f(n))$ means $O(f(n) \cdot \log^c f(n))$ for some constant $c > 0$.

One might be concerned about the constraint $Bx = Fb$, as so far we have mostly talked about unconstrained optimization over $\mathbb{R}^m$ – however, as it is a linear subspace of $\mathbb{R}^m$ one can easily prove that as long as we, in every step, project the gradient onto $\{x : Bx = 0\}$ then the conclusion of Theorem 1 still[4] holds.

Further, since we expect to solve (10) approximately (not exactly), we need to show how to deal with errors which might occur in this step and how do they affect solving (10) given a solution to (10), we refer to Section 4.6 for a discussion on this issue.

4.4 Bounding parameters for gradient descent.

To apply the gradient descent algorithm to problem (10) and estimate its running time we need to perform the following steps

1. Prove that the objective $f(x) = \|x - P(x)\|^2$ is convex on $\mathbb{R}^m$.
2. Prove that f has Lipschitz-continuous gradient and find the Lipschitz constant.
3. Find a "good starting point" $x_1 \in \mathbb{R}^m$.
4. Estimate the running time of a single iteration, i.e., how quickly can we compute the gradient.

We now discuss these steps one by one.

Step 1. Convexity. In general, for every set $S \subseteq \mathbb{R}^n$ the function $x \mapsto \text{dist}^2(x, S)$ is convex (exercise).

Step 2. Lipschitz gradient. One has to compute the gradient of f first to analyze its properties. For that, we first observe that the projection operator on the hypercube is described rather simply by the following

$$P(x)_i = \begin{cases} x_i & \text{if } x_i \in [-1, 1], \\ -1 & \text{if } x_i < -1, \\ 1 & \text{if } x_i > 1. \end{cases}$$

and hence

$$f(x) = \sum_{i=1}^{n} h(x_i),$$

where $h : \mathbb{R} \to \mathbb{R}$ is the function $\text{dist}^2(x, [-1, 1])$, i.e.,

$$h(x) = \begin{cases} 0 & \text{if } x \in [-1, 1], \\ (x+1)^2 & \text{if } x < -1, \\ (x-1)^2 & \text{if } x > 1. \end{cases}$$

[4]For that one can either consider a more general variant of gradient descent called *projected gradient descent* – see Theorem 6, or just repeat the proof of Theorem 1 over a linear subspace.

Thus, in particular

$$[\nabla f(x)]_i = \begin{cases} 0 & \text{if } x \in [-1,1], \\ 2(x+1) & \text{if } x < -1, \\ 2(x-1) & \text{if } x > 1. \end{cases}$$

Such a function ($\nabla f(x)$) is easily seen to be Lipschitz, with a Lipschitz constant $L = 2$.

Step 3. Good Starting Point. We need to provide a flow $g \in H_F$ which is as close as possible to the "cube" $B_{m,\infty}$. For that we can just find the flow $g \in H_F$ with smallest Euclidean norm. In other words, we can project the origin onto the affine subspace H_F to obtain a flow g (we will discuss the time complexity of this task in Step 4). Note that if $\|f\|^2 > m$ then the optimal value of (10) is non-zero, which is enough to conclude that $F^\star < F$. Indeed, if there was a flow $x \in B_{m,\infty}$ such that $Bx = Fb$ then x would be a point in H_F of Euclidean norm

$$\|x\|_2^2 \leq m \|x\|_\infty^2 \leq m,$$

and hence we would arrive at a contradiction with the choice of g. Such a choice of $x_1 = g$ in particular implies that $\|x_1 - x^\star\| \leq 2\sqrt{m}$, thus we have $D = O(\sqrt{m})$.

Step 4. Complexity of computing gradients. In Step 2 we have already derived a formula for the gradient of f. However, since we are working in a constrained setting, such a vector $\nabla f(x)$ needs to be projected onto the linear subspace $H := \{x \in \mathbb{R}^m : Bx = 0\}$. In other words, let $\Pi : \mathbb{R}^m \to \mathbb{R}^m$ be the orthogonal projection onto H. In every step t of the gradient descent algorithm, we need to compute

$$\Pi \nabla f(x_t).$$

While $\nabla f(x_t)$ is easy to compute in linear time (given the formulas derived previously) computing its orthogonal projection might be quite expensive. In fact, even if we precomputed Π beforehand (which would take roughly $O(m^3)$ time, it would still take $O(m^2)$ time to apply it to the vector $\nabla f(x_t)$. To the rescue here comes an important result by Spielman and Teng [6] that such a projection can be computed in time $\widetilde{O}(m)$. This is achieved by noting that such a projection is nothing but computing a certain electrical flow on the graph G and thus can be found using a Laplacian solver (see for instance [7]).

4.5 Running time analysis.

Given the above discussion we can bound the running time of the gradient descent based algorithm to find an ε-approximate max-flow. By Theorem 1 the number of iterations to solve (10) up to an error of ε is

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

the cost of every iteration (and of finding x_1) is $\widetilde{O}(m)$. Thus in total, this requires $\widetilde{O}\left(\frac{m^2}{\varepsilon}\right)$ time.

To recover a flow of value $F^\star(1-\varepsilon)$ from a solution to (10) we need to solve (10) up to precision $\frac{F^\star\varepsilon}{\sqrt{m}}$ (see Section 4.6). Thus finally we obtain a running time bound of $\widetilde{O}\left(\varepsilon^{-1}\frac{m^{2.5}}{F^\star}\right)$. The binary search over F to find $F^\star$ up to a good precision incurs only a logarithmic factor in $\frac{m}{\varepsilon}$ and hence does not significantly affect the running time.

4.6 Final remarks

Dealing with approximate solutions. A concern which arises when dealing with the formulation (10) is that in order to solve (9) we need to solve (10) exactly – i.e., with error $\varepsilon = 0$ (we need to know whether the optimal solution is zero or non-zero), which is not really possible using gradient descent. This is dealt with using the following lemma (we refer to [2] for a proof).

Lemma 8. *Let $g \in \mathbb{R}^m$ be an s-t flow of value F in the graph G, i.e., $Bg = Fb$. Suppose that f overflows a total of F_{ov} units of flow (formally $F_{ov} := \|g - P(g)\|_1$) on all edges. There is an algorithm which given g, finds a flow g' that does not overflow any edge (i.e., $g' \in B_{m,\infty}$) of value $F' \geq F - F_{ov}$, in time $O(m \log n)$.*

It says essentially that any error we make in solving (10) can be efficiently turned into an error in the original objective of (8). More precisely: if we solve (10) up to an error of $\varepsilon > 0$ then we can efficiently recover a flow of value at least $F - \sqrt{m}\varepsilon$.

Faster Algorithm. We refer to [2] for a stronger version of Theorem 7 achieving running time $\widetilde{O}\left(\varepsilon^{-1}\frac{m^{3/2}}{F^\star}\right)$ by applying Nesterov's accelerated gradient descent instead of the vanilla version we use here.

References

[1] Sebastien Bubeck. Convex Optimization: Algorithms and Complexity. *ArXiv e-prints*, May 2014.

[2] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 755–764, New York, NY, USA, 2013. ACM.

[3] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley Interscience, 1983.

[4] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(\frac{1}{k^2})$. In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

[5] Yurii Nesterov. *Introductory lectures on convex optimization*, volume 87. Springer Science & Business Media, 2004.

[6] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, pages 81–90, 2004.

[7] Nisheeth K. Vishnoi. $Lx = b$. *Foundations and Trends in Theoretical Computer Science*, 8(1-2):1–141, 2012.