6.S978 Graphs, Linear Algebra, and Optimization – Fall 2015 September 10, 2015

Lecture 1

Lecturer: Aleksander Mądry *Scribe: Dimitris Tsipras*

1 Introduction

Welcome to 6.S978! Please take a look at the course website http://courses.csail.mit.edu/6.S978 that contains all the administrative information. Also, don't forget to subscribe to the class mailing list http://lists.csail.mit.edu/mailman/listinfo/6s978 (this will be the primary medium of communication for this class) and sign up for scribing (see the course website for instructions).

2 What This Class Will Be About

Our main focus in this class will be on graph algorithms. However, our treatment of this topic will be very different to the one you might have seen in classic algorithms classes.

To explain this difference, recall first that traditional graph algorithms are purely combinatorial in spirit. That is, they operate on various combinatorial notions associated with a graph, such as paths, cuts, trees, and partitions, using sophisticated data structures to make these manipulations efficient. Resorting to these kind of approaches is fairly natural – after all, graphs are combinatorial objects – and, historically, it was very successful. The resulting techniques were shaping our understanding of algorithms, in general. In particular, it is not a coincidence that there is so much of graph algorithms coverage in undergraduate algorithms curriculum.

However, despite all these successes, over the last 60 years, it became evident that there are certain fundamental limitations that purely combinatorial techniques seem to be unable to overcome. Consequently, there is a need for development of broader perspective on graph algorithms. One of the first attempts in this direction dates back to 1980s, when the field of *spectral graph theory* was born. Very roughly speaking, the objective of spectral graph theory is to tie the combinatorial properties of the graph to certain linear-algebraic properties, such as eigenvalues and eigenvectors, of associated matrices, such as the Laplacian or adjacency matrix. This new, more continuous way of looking at graphs led to substantial progress on a number of longstanding open problems in algorithmic graph theory. However, the connections it established were mostly descriptive in nature, and thus were of somewhat limited use when one wanted to obtain faster algorithms for various fundamental graph problems.

The missing component here turns out to be the toolbox of *convex optimization*, a field that develops efficient algorithms for minimizing convex functions in different (purely continuous) scenarios. Over the last five years, this component enabled us to strengthen the connections unearthed by spectral graph theory and turn them into an approach for producing fast graph algorithms. In fact, this direction became a new frontier of algorithmic graph theory, leading to a number of exciting breakthroughs.

This class will study the new landscape that emerges from these developments. In particular, it will explore the "mental picture" of areas and connections that underlie them – see Figure 1. Along the way, we will not only try to cover all the key techniques in modern graph algorithms and spectral graph theory but also survey the core tools of convex optimization. In fact, we will see that algorithmic graph theory is not merely one more application area for convex optimization methods, but also a new lens on this field that enables us to approach – and even make progress on – some of its most fundamental questions.

3 The Maximum Flow Problem

As explained above, the set of topics that we want to cover throughout the semester is pretty broad. It is, therefore, somewhat surprising that there is a single problem that can be tied to all of them: the

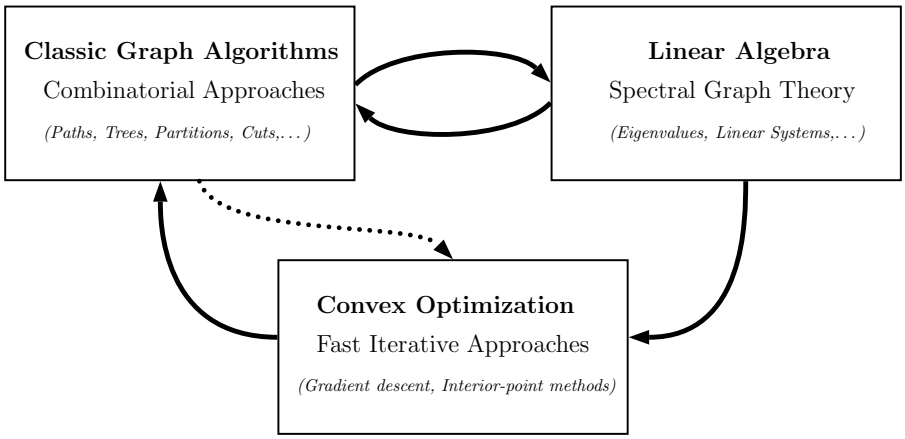

Figure 1: A mental picture of the areas and connections that we want to explore.

maximum flow problem. This problem is one of the most intensely studied problem in graph algorithms and, more broadly, combinatorial optimization. In fact, over the years, it was one of the key drivers of development of combinatorial algorithms. For instance, it was in the context of this problem that the idea of primal-dual algorithms was first conceived.

Now, it turns out that the maximum flow problem is also inspiring most of the modern graph algorithmic developments. The whole paradigm of using continuous approaches to graph algorithms stems from attempts to make progress on this problem, and all the current fastest maximum flow algorithms were obtained recently via such techniques. We will follow this "tale of one problem" narrative in our lectures. This way we will always have a common baseline problem to enable us to compare and tie together various tools and concepts that we will discuss during the semester.

3.1 A Formal Definition

Let us thus start by formally defining the maximum flow problem.

Maximum flow problem:

- *Input:* A (directed) graph $G = (V, E)$ with a *source* vertex $s \in V$ and a *sink* vertex $t \in V$, and integer arc capacity u_e associated with each arc $e \in E$.
- *Goal:* Compute a *feasible* s-t flow f of maximum *value* in G.

To make the above definition precise, we need to define the notion of flow feasibility and that of the value of a flow.

An $s-t$ flow f is *feasible* iff it satisfies two types of constraints:

- **(flow conservation constraints)**: there are no "leaks" at any other vertex than the source s and the sink t. That is,
$$\forall v \neq s, t, \quad \text{flow-in}_f(v) = \text{flow-out}_f(v)$$
- **(capacity constraints)**: the arc capacities are not violated. That is,
$$\forall e \in E, \quad 0 \leq f_e \leq u_e$$

The *value* of a flow f is the net amount of flow coming out of the source vertex s. That is,
$$\text{value}(f) := \text{flow-out}_f(s)$$

Also, for future reference, we will usually denote the number $|E|$ of arcs/edges of a graph $G = (V, E)$ as m, and its number $|V|$ of vertices as n.

3.2 Our Focus: Unit-capacity and Undirected Case

Throughout (almost) the whole class we will be focusing our attention on a special case of the maximum flow problem. Namely, we will assume that the instances we are dealing with are: *unit-capacity*, i.e., $u_e = 1$ for all arcs e; and *undirected*, i.e., for each arc $e = (u, v)$ there is an arc $e' = (v, u)$ that goes in opposite direction and has the same (unit) capacity.

Obviously, by making these two assumptions we are simplifying our problem quite a bit. However, as it turns out, these simplifications are not as significant as one might think. First of all, almost all the techniques that can tackle the unit-capacity case can also be easily extended to handle the general capacity case, while incurring only small running time overhead. Secondly, the maximum flow problem has a certain surprising feature: one can essentially reduce the general, directed maximum flow problem to its undirected variant. More precisely, in our first problem set we will prove the following statement.

Fact 1 *Given an algorithm that solves the (unit-capacity[1]) undirected maximum flow problem in time $O(T(m, n))$ we can transform it into an algorithm that solves the (unit-capacity) directed maximum flow problem in time $\widetilde{O}(T(m, n) + m)$.*

The notation $\widetilde{O}()$ used above is the same as the $O()$ notation, but it additionally suppresses logarithmic factors, that is $\widetilde{O}(g(n)) = O(g(n) \cdot \text{polylog}(g(n)))$.

So, if we are willing to give up logarithmic factor (which we are), solving the undirected case of the maximum flow problem is essentially equivalent to solving the directed variant.

In the light of the above, from now on, unless stated otherwise, we are always solving the unit-capacity and undirected variant of the maximum flow problem.

3.3 Previous Algorithms and Approach

How can we solve the maximum flow problem? For a long time the dominating theme in all algorithms was the *augmenting path framework* of Ford and Fulkerson. In this framework, we gradually try to push more flow through the network by repeatedly finding paths from s to t, in the *residual* graph. Each time we find such a path, we flip the direction of the arcs on it to encode adding the flow along this path. (Recall that all capacities are one here, and we view each undirected edge as two arcs going in opposite directions.) Once we are no longer able to find an s-t path, we can decode the maximal flow by looking at the flipped arcs. More precisely, the procedure is as follows.

Algorithm 1 Repeated Augmenting Path Finding

```
G' ← G                                          ▷ G' is the residual (directed) graph
while s is connected to t in G' do
    find a (directed) s-t path P in G'
    flip in G' the directions of all the arcs in P.
for e ∈ E do
    if e is flipped in G' then f*_e = 1
    else f*_e = 0
return f*
```

By appealing to the celebrated Max Flow-Min Cut theorem due to [Ford-Fulkerson '56] and, independently, [Elias-Feinstein-Shannon '56] (from MIT!), one can show that the above algorithm indeed computes the maximum flow in G. (Take a moment to make sure you know how to show that!)

What is the running time of this procedure? In each step, finding an $s - t$ path takes time $O(m)$. Since in each step the flow is increased by one, and the value of any feasible flow cannot exceed n (which is the max value of any s-t cut), this algorithm terminates in time $O(m \cdot n)$.

But, can we do better? The fastest known algorithm in this framework (for unit-capacity case) is due to [Even-Tarjan '75]. Roughly speaking, it works by trying to find many augmenting paths

[1] The reduction also works for general capacities, we focused on unit-capacity case just for simplicity.

simultaneously and requires much more sophisticated analysis. Its running time is $O(m \cdot \min(m^{1/2}, n^{2/3}))$. This bound stood for almost 40 years, and this suggests that the limits of the whole framework may have been reached.

4 Maximum Flow as an Optimization Problem

What we will do in this class, is try a completely different approach. We will phrase the problem as an optimization problem and resort to known methods for solving it. However, this alone will be too crude of an approach. So we will examine what specific combinatorial structure we can exploit to push the running time of the optimization even further. As we will see later on, this approach will ultimately lead to improvement over the running time of Even-Tarjan algorithm.

4.1 Encoding the Graph and Flow as Linear-Algebraic Objects

In order to phrase the (undirected unit-capacity) maximum flow problem as a generic optimization problem, we will need to encode our graph and flows as vectors and matrices of some continuous vector space.

To this end, given our undirected graph G, we will orient its edges in an arbitrary way and for a given directed flow f, we will overload notation and encode it as a general vector in $f \in \mathbb{R}^m$, where each coordinate corresponds to one of the edges of G. (Thus, we will index its coordinates according to the edges). More precisely, for a given edge e, we will define the e-th coordinate f_e of the vector f as

$$f_e := \begin{cases} f_e & \text{if the flow is congruent with the direction of } e \\ -f_e & \text{otherwise} \end{cases}$$

Note that a negative flow in this notation is simply positive flow in the direction opposite to its orientation.

For the case of the input graph, there are many well-known ways to encode it as a matrix. One that we will find convenient is the $n \times m$ *edge-vertex incidence matrix* B indexed by vertices and edges, so that the column B_e corresponding to edge $e = (w, u)$, has values -1 at w, 1 at u and zero elsewhere.

4.2 The Problem Formulation

Now, one can convince oneself that the task of finding a feasible s-t flow of maximum value in unit-capacity undirected graph is equivalent to finding a unit value s-t flow in G that: (a) satisfies flow conservation constraints; and (b) minimizes the maximum flow on any of the arcs. Rescaling this flow to match the capacity of the most congested edge, will give us a maximum flow for the graph.

Consequently, the (unit-capacity undirected) maximum flow problem can be cast as the following optimization task.

$$\begin{aligned} \min \quad & \|f\|_\infty \\ \text{s.t.} \quad & f \in \mathcal{F}_{s,t} \end{aligned}$$

Above, $f \in \mathbb{R}^m$ is our vector encoding of a flow and $\|v\|_\infty := \max_i |v_i|$ is the ℓ_∞-norm. Also, $\mathcal{F}_{s,t}$ is defined to be the space of all encodings of s-t flows of value 1 that satisfy the flow conservation constraints. In order to encode the flow conservation constraints, we can utilize the matrix representation of the graph and phrase the problem as follows

$$\begin{aligned} \min \quad & \|f\|_\infty \\ \text{s.t.} \quad & Bf = \mathcal{X}_{s,t} \end{aligned}$$

where $\mathcal{X}_{s,t}$ is the vector with 1 at vertex t, -1 at vertex s and zero everywhere else.

To see why the two formulations are equivalent, note that for some vertex v, $(Bf)_v$ is the flow "deficit" at this node. This has to be -1 at s, 1 at t (to ensure unit flow) and zero at any other node.

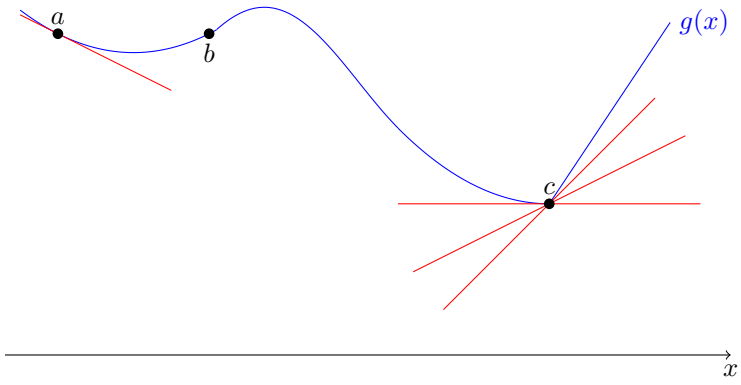

Figure 2: Subgradients at different points

4.3 Convexity and Subgradients

A notion the comes up a lot in continuous optimization, is that of convexity. Recall the definition of a convex set and a convex function.

Definition 2 *A set $S \subseteq \mathbb{R}^n$ is convex iff $\forall x, y \in S$*

$$\forall a \in [0,1], \quad ax + (1-a)y \in S$$

A function $g : \mathcal{D} \to \mathbb{R}$ is convex iff $\forall x, y \in \mathcal{D}$

$$\forall a \in [0,1], \quad ag(x) + (1-a)g(y) \geq g(ax + (1-a)y)$$

In other words, a set is convex iff for any two points in it, the straight line connecting them lies entirely in the set. A function is convex iff for any two points, the function plot between them lies below the straight line connecting their values. As it turns out, there are some great news about our optimization problem.

Fact 3 *The set $\mathcal{F}_{s,t}$ is convex.*

Fact 4 *The function $\|f\|_\infty$ is convex.*

The good thing about convexity is the it implies a certain local to global property. Any local minimum is a global minimum. For non-convex function this does not hold, there can be multiple local minima that are not global minima, thus making the task of minimizing the function extremely hard. We will now briefly discuss the concept of the subgradient.

Definition 5 *For a function $g : \mathcal{D} \to \mathbb{R}$, a vector $s \in \mathbb{R}^n$ is a* subgradient *of g at point $x \in \mathcal{D}$ iff*

$$\forall y \in \mathcal{D}, \quad g(x) - g(y) \leq s^T(x - y)$$

we denote this as $s \in \partial g(x)$.

Intuitively, $g(x)+s^T(y-x)$ for various $y \in \mathbb{R}^n$ defines a hyperplane that lies below $g(y)$ at all points. This essentially provides a lower bound for the value of the function as one goes away from x. For different points of a function, the set of subgradients can be quite different. Observe the function g of Figure 2. The function has a single subgradient at point a (the gradient), no subgradients at b ($\partial g(b) = \emptyset$) and multiple subgradients at point c, where it is not differentiable.

Theorem 6 *If $g : \mathcal{D} \to \mathbb{R}$ is a convex function, then*

$$\forall x \quad \partial g(x) \neq \emptyset$$

in addition, if g is differentiable at x,

$$\partial g(x) = \{\nabla g(x)\}$$

Proof (*Non-Emptiness*): Let g_{epi} be the epigraph of the function g, that is the set of points lying above the function plot in its domain.

$$g_{epi} = \{(a, b) \in \mathcal{D} \times \mathbb{R} : g(a) \leq b\}$$

Since g is a convex function, g_{epi} is a convex set. For any $x \in \mathcal{D}$, $(x, f(x))$ belongs to the boundary of this set. By the *supporting hyperplane* theorem, there exists a hyperplane that passes through $(x, f(x))$ and leaves the whole set g_{epi} on one side of it, that is there exist $s \in \mathbb{R}^n$, $w \in \mathbb{R}$ such that for all $(a, b) \in g_{epi}$

$$s^T a - wb \leq s^T x - wf(x)$$

Since g_{epi} is the epigraph of a convex function, one can show that such a hyperplane can be chosen such that $w = 1$. Since for any y, $(y, f(y)) \in g_{epi}$

$$s^T y - f(y) \leq s^T x - f(x)$$

and therefore $s \in \partial g(x)$.

(*Uniqueness for g differentiable:*) Let some $s \in \partial g(x)$. For some $i \in [n]$, let e_i be the vector with 1 at coordinate i and zero elsewhere. By the definition of the subgradient, for $y = x + \varepsilon e_i$, $y = x - \varepsilon e_i$,

$$\frac{f(x) - f(x - \varepsilon e_i)}{\varepsilon} \leq s^T e_i \leq \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon}$$

for any $\varepsilon > 0$. Taking the limit as $\varepsilon \to 0$, since g is differentiable, the bounds converge and

$$s_i = \frac{\partial f}{\partial x_i}(x)$$

This holds for every i, and therefore $s = \nabla f(x)$. ■

We can use subgradients to certify the optimality of a point in terms of minimizing the function, in the same way we use the gradient in differentiable functions.

Theorem 7 *Let $x \in \mathcal{D}$ and $g : \mathcal{D} \to \mathbb{R}$, then x is a global minimum iff $0 \in \partial g(x)$.*

Subgradients can substitute the use of the gradient in cases of non-differentiable functions. Moreover, they also work when the function is restricted in some subset of its domain $\mathcal{D} \subset \mathbb{R}^n$. The importance of this becomes apparent when one considers cases such as the one in Figure 3. In this case when one restricts $g(x)$ to the domain D, the point x^* is a global minimum. Even though its gradient is not zero, 0 is a subgradient of g at x^*.

4.4 First Step: Fast Approximation of the Maximum Flow

Looking back at our optimization problem for the maximum flow, since the feasible set is convex (indeed affine) and the objective function convex, it can be solved by the ellipsoid algorithm in polynomial time. This running time however is far from efficient for the specific problem at hand. In order to come up with faster algorithms following this approach we will first relax our goal a little bit, and aim for an approximate solution. For now our goal will be to find for every a $(1 - \varepsilon)$-approximate solution – for every $\varepsilon > 0$ – in running time that depends polynomially on $\frac{1}{\varepsilon}$. Note that the ellipsoid algorithm finds for every $\varepsilon > 0$ a $(1 - \varepsilon)$-approximation within time depending on $\log(\frac{1}{\varepsilon})$, so we consider this solution to be exact.

5 A first view of our basic tool: Gradient Descent

We will now briefly discuss the intuition behind the gradient descent method, a tool that we will use extensively. While the intuition is very simple, it is so powerful that it appears in almost every algorithm (we will make the term "every algorithm" more concrete). Suppose that we want to compute

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

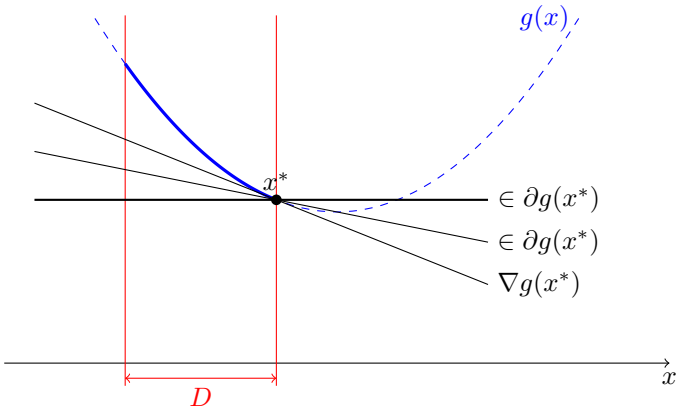

Figure 3: A global minimum with non-zero gradient

for some convex and differentiable function $g : \mathbb{R}^n \to \mathbb{R}$. The way we will try to do this is start at some point $x_1 \in \mathbb{R}^n$ and use it to compute a point x_2 closer to the optimal solution. We will then use x_2 to compute a closer point x_3 and so on, until we get sufficiently close to an optimal point.

Since our objective function is convex and differentiable, its plot will look like a bowl. We can therefore try to go down the "steepest" slope from our point to get fast to the bottom of the bowl, our global minimum. Our update step is thus

$$x_{i+1} = x_i - \eta \nabla g(x_i)$$

where η is some normalizing constant. This is the greedy approach to minimization. Remember that $\nabla g(x_i)$ is direction in which g increases with the highest rate. We will therefore move in the opposite direction. The factor η prevents us from "overshooting" and going too far in one step. This will lead us down a path of x_i with strictly decreasing g, that is $g(x_i) > g(x_{i+1})$, as is depicted in Figure 4.

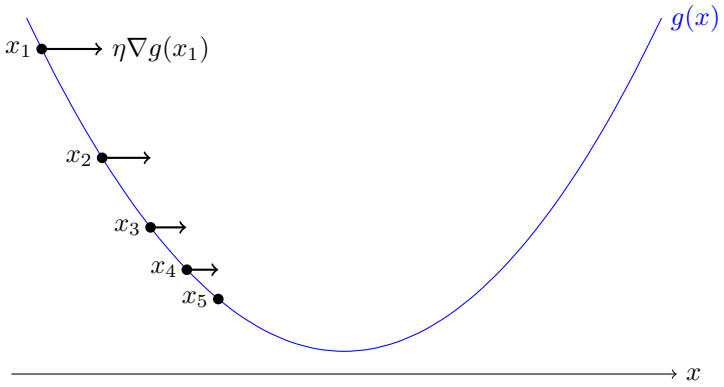

Figure 4: An example of gradient descent

In order to better understand this procedure, lets look at the Taylor expansion of g at some point y

$$g(y) = g(x_i) + \nabla g(x_i)^T (y - x_i) + O(\|y - x\|_2^2)$$

This means that as long as we don't go too far from x_i, our function can be well approximated by a hyperplane and we are indeed moving in the right direction. In the next lectures, we will revisit the method in a more rigorous way.