Lecture 15: Minimum Cost Flows and Intro to Linear Programming

Objectives of this lecture

- Introduce the minimum-cost flow problem and "demands".
- Equivalence between different version of minimum-cost flow (e.g., mincost-maxflow).
- Algorithm statement for mincost flow (no proof).
- Definition of a linear program.
- Standard form of linear programs.

1 Philosophy

In this lecture we introduce the problem of minimum cost flow, which asks to route flow satisfying a particular "demand" while minimizing the cost (now edges have capacities and costs). We will give a brief argument that it can be solved in polynomial time (and state some stronger versions that are known).

Minimum cost flows naturally lead into linear programs, which are a very broad class of optimization problems which are known to admit polynomial time solutions. This gives you the ability to notice when a very large number of problems are polynomial time solvable (it is easy to recognize linear programs).

2 Assumed Knowledge

1. Graphs: vertices, edges, weights
2. Paths in graphs, finding paths.
3. Matrix-vector products, linearity.

3 Minimum Cost Flows

3.1 Defining Minimum Cost Flow and Demands

Throughout, we consider directed graphs that have edges with capacities, but also now with costs. At a high level, the minimum cost flow problem asks to route flow while minimizing cost. We will make this more precise in this lecture. Here is the way I prefer to think about minimum cost flows. For this we need to introduce the concept of a "demand vector".

Demand vectors. A demand vector is a vector $d \in \mathbb{R}^V$ (i.e., it assigns a real number to each vertex of the graph). The demand at a vertex v is supposed to represent the total "net flow" at the vertex, in other words incoming flow minus outgoing flow.

Precisely, we say that a flow $f \in \mathbb{R}^E$ (an assignment of real number of each edge of the graph) routes demand $d \in \mathbb{R}^V$ if for all $v \in V$:

$$\underbrace{\sum_{e=(u\to v)} f_e}_{\text{incoming}} - \underbrace{\sum_{e=(v\to u)} f_e}_{\text{outgoing}} = d_v.$$

Feasibility. We say that a demand d is *feasible* for the graph G if there is a flow that routes demand d. Note that for a feasible demand, it holds that $\sum_{v\in V} d_v = 0$, because every unit of flow on an edge adds demand to one vertex but removes the same from another.

Definition of minimum-cost flow. One version of the minimum-cost flow problem asks: given G (which has edges with given costs and capacities), and demand d, to output a flow routing d which minimizes the cost, defined as

$$c^\top f = \sum_{e\in E} c_e f_e.$$

Written even more compactly, the minimum-cost flow problem is:

$$\min_{\substack{f\in\mathbb{R}^E \\ B^\top f = d \\ 0\le f_e \le \text{cap}_e \,\forall e\in E}} c^\top f. \tag{1}$$

What each pieces represents:

- $f \in \mathbb{R}^E$: the flow assigns a real number to each edge.
- $B^\top f = d$: the flow routes the demand d.
- $0 \le f_e \le \text{cap}_e$: the flow is nonnegative on the directed edge and at most the edge capacity.
- $c^\top f$: the goal is to minimize the cost of the flow.

Known runtimes. It is known how to solve minimum-cost flow in polynomial time (something like $O(n^3)$). By "solve" we mean decide if a feasible flow exists, and if so, outputs the one with minimum cost. If you assume the costs and capacities to be integers, polynomially bounded in n, then faster runtimes are known.

3.2 Reductions between Different Versions

In this section we describe special cases and equivalent versions of mincost flow. The point of this is to demonstrate that mincost flow is a quite general problem that captures all flow problems we've considered thus far in the course.

Maxflow as a special case. $s \to t$ maxflow is a special case of mincost flow. Add a $t \to s$ edge with cost -1 and infinite capacity. Give every other edge cost 0. Then the minimum-cost flow which routes demand $d = 0$ is the $s \to t$ maxflow.

Minimum-cost maximum-flow. In some textbooks, people instead ask: what is the minimum-cost way to route the maximum flow (for example, $s \to t$)? To do this, first just run $s \to t$ maxflow to figure out the target demand, and then run mincost flow with that demand.

3.3 Integrality

THis is important for the next section. If you look at (1), there is no contraint that the optimal flow should assign integer values to all edges. It turns out that when all capacities and demands are integers, then this is true: the minimum cost flow assigns integer flow to all edges. We will see an example proof later, after discussing an algorithm for mincost flow.

3.4 Algorithm Sketch

The algorithm for minimum-cost flow is based on *cycle cancelling*. In other words, find a cycle whose total cost is negative and update my flow by adding it.

Let f be a flow routing demand d. Let f^* be the optimal flow. Then we know that:

- $f^* - f$ is feasible in the residual graph, and
- $f^* - f$ routes 0 net demand, and thus can be split into cycles.

In other words, if f is not the minimum cost flow, there must be a negative cycle. So all the classical combinatorial mincost flow algorithms will find a cycle with negative weight and delete it. Finding the negative weight cycle generally can be done with something like the Bellman-Ford algorithm (to find negative cycles in a graph if they exist). As described the runtime depends on the maximum cost/capacity, but you can use capacity scaling methods to speed this up further (this will not be discussed in class further, or tested on).

Why does this imply integrality? The initial flow is integral. Consider adjusting a negative weight cycle. This doesn't cause any flow values to become non-integer. Because the algorithm terminates, the final result is a flow that is integral.

It should be noted that not all optimal mincost flows have to be integral, it's just that one exists.

4 Linear Programming

A linear program (LP) has the following form. The input consists of linear equations and inequalities that constrain a vector $x \in \mathbb{R}^n$. It asks to minimize (or maximize) $c^\top x = \sum_{i=1}^n c_i x_i$ for $c \in \mathbb{R}^n$. The "standard form" of linear programs is:

$$\min_{Ax=b\, x_i \geq 0 \forall i} c^\top x, \tag{2}$$

where

- $A \in \mathbb{R}^{m \times n}$ for $m < n$ (normally) contains the linear equalities that are enforced on x,
- $x \geq 0$ is the "inequality" constraints. The "standard form" of an LP only has simple $x_i \geq 0$ constraints, but more generally you can have any linear inequalities that you want,
- $c^\top x$ is the cost of x.

As you can see, mincost flow is more or less a special case of LPs (the capacities are an inequality constraint, just not in standard form). Below we will see how to convert any LP to standard primal form without increasing its size by too much.

Putting an LP in standard form. A general LP is allowed to have inequality constraints, for example $x_1 + 2x_2 + 3x_3 \geq 7$. How do we convert this to standard form? First all an equality constraint $t = x_1 + 2x_2 + 3x_3 - 7$, for a new variable t (so a new variable is introduced for every equality constraint). Then add the "standard form" constraint $t \geq 0$.

As you can see, the new LP is in standard primal form, and the number of variables increased by the total number of equations.

Feasibility. We say an LP is feasible if there is an x satisfying all the constraints $Ax = b$ and $x \geq 0$ (this doesn't always have to exist). The analogy is that we called a mincost flow instance feasible if there was a flow that routed the demands within the capacity budgets (ignoring the costs).

Integrality. There is no useful notion of integrality for most LPs. You should expect that the optimal solution is fractional except in special cases.

Runtime of linear programming. While this is much more difficult (will probably not do in this course, maybe at the end if time permits), it is known how to solve LPs to high-accuracy in polynomial time. By high-accuracy, this means that if you want error ϵ on the answer to (2), the runtime is $n^{O(1)} \log(1/\epsilon)$. In other words, polynomial time, with a logarithmic dependence on the accuracy. There are a number of ways to do this, but are all "geometric" in nature and beyond the scope of this course.