Lecture Notes CS:5360 Introduction to max flow problem
Lecture 1: Aug 27, 2017
Scribe: Yumna Anwar

1 Course Overview

The course assumes that students have basic knowledge of algorithms topics, such as:

- Dynamic programming
- Greedy algorithms
- Divide and conquer
- NP-completeness

These topics will not be covered in the class, so it might be a good idea to brush up on them. If you lack some of these concepts you might also consider taking the course design and implementation of algorithms. In this course we will cover four topics:

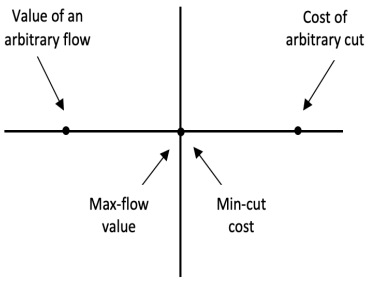
1. Combinatorial Optimization
2. Randomized Algorithms
3. Approximation Algorithms
4. Streaming and Parallel Algorithms

Throughout the course we will emphasize two themes:

- Interplay between a combinatorial view and a math programming view of problems
- Methods for probabilistic analysis.

1.1 Combinatorial Optimization

In this course, combinatorial optimization consists of three inter-related topics: Flows, Cuts, and Matching. Though all these are graph problems, they have interesting math programming models. *Linear program* is one example of a math programming problem. Another thing to note when using math programming approaches is the notion of *duality*. Notion of duality means that there are two paired problems, one maximization and one minimization problem. For instance, the max-flow problem and the min-cut problem are duals of each other. We will show that the value of max-flow is same as the cost of the min-cut.



1.2 Graph problems

Graph problems can be categorized based on the complexities. The "Easiest"problems includes the traversals, like *breath first search* and *depth first search* and their applications to connected components and strongly connected components, etc . All of these problems can be solved in linear time, that is $\mathcal{O}(m+n)$, where n is the number of vertices and m is the number of edges.

Slightly harder problems include *minimum spanning tree* (e.g; Kruskal's or Prim's algorithm), *single source shortest paths* (e.g; Dijkstra algorithm). These are super-linear time algorithms and are more sophisticated. The complexity of these graph problems is $\mathcal{O}((m+n)\log n)$, where n is the number of vertices and m is the number of edges.

Even harder problems are problems with flows, cuts, matchings. These can be solved in polynomial time, but the running time may be as high as $\mathcal{O}(n^3)$ where n is the number of vertices. And these algorithms require a lot more machinery.

1.2.1 Polynomial time reduction

Polynomial time reduction are helpful in two ways, building efficient algorithms for new problems and in showing that some problems can not be solved efficiently.

A problem is defined as NP-hard if it is at least as hard as any nondeterministic polynomial time problem (NP-problem). To show that some problem is NP-hard you take a known NP-hard problem and reduce it to our current problem in polynomial time. The direction of the reduction is very important in these reductions.

2 Max flow problem

Imagine you have a network of pipes of different capacities. In this network there is a source that produces some fluid (water, for instance) and there is a sink that processes the fluid it receives.

What is the maximum amount of flow we can set from source to sink? Keep in mind that some pipes are thinner and some are thicker so they they have different capacities. The input to this problem is the network and capacities of the pipe. The output is the maximum amount of fluid one can send through.

More formally,

- Let $G = (V, E)$ donate a directed graph
- Let s and t be two special vertices donating *source* and *target* (sink) respectively.
- Let $C : E \to \mathbb{R}_{>=0}$ donate the capacity function on edges of E

These are the pieces that constitutes the input to the Max flow problem.

A ***flow*** in a directed graph $G = (V, E)$ is a function $f : E \to \mathbb{R}$ satisfying the following flow conservation constraint for every vertex v except s and t :

$$\sum_{u \in OutNbr(v)} f(v \to u) = \sum_{w \in InNbr(v)} f(w \to v),\ for\ each\ v \in V \setminus \{s, t\}$$

Meaning, for every $v \in V$ that is not a source or target the outflow from v should be equal to the in flow into v. For directed graph we will use $u \to v$ to donate an edge from u to v. Without loss of generality (WLOG) $f(u \to v) = 0$, for $u \to v \notin E$. So the above constraint can be simplified to:

$$\sum_{u \in V} f(v \to u) = \sum_{w \in V} f(w \to v),\ for\ each\ v \in V \setminus \{s, t\}$$

Therefore, a flow is just a function that satisfies flow conservation.

The ***value of a flow*** f, denoted as $|f|$, is the outflow from s minus the inflow into s:

$$\sum_{u \in V} f(s \to u) - \sum_{w \in V} f(w \to s).$$

The value of the flow is the net outflow from the source. The net outflow from any vertices that is not the source is 0. Another important things to note is that we are assuming that "pipes"have an upper bound on the flow.

A flow f is ***feasible*** if it is non negative and is bounded by the capacities:

$$0 \leq f(u \to v) \leq c(u \to v), \text{ for } u \to v \in E$$

The ***input*** consists of a directed graph $G = (V, E)$, a capacity function $C : E \to \mathbb{R}$ and special vertices $s, t \in V$

The ***output*** is a feasible flow with maximum value.

2.0.1 Example of max flow problem

Consider the following network of pipes represented as a directed graph in figure 1 below:

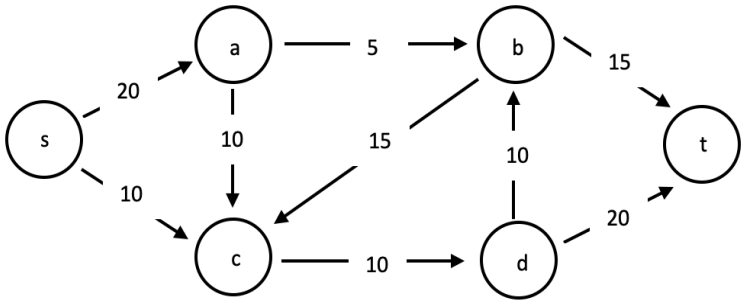


Figure 1: Network of pipes represented as a directed graph

One possible flow from source to target is: $s \to c$ (10), $c \to d$ (10), $d \to t$ (10). This has a value of 10. Another flow that can be considered is: $s \to a$ (15), $a \to b$ (5), $a \to c$ (10), $c \to d$ (10), $b \to t$ (5), $d \to t$ (10). This has a total value of 15. Which is also the maximum flow possible through our network. Figure 2 below shows the min capacity cut in reference to our example:

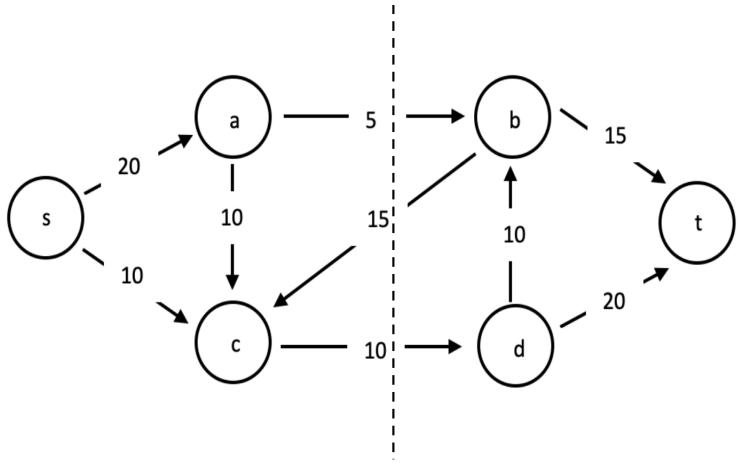


Figure 2: Network of pipes and its min capacity cut

This is the cut that is saturated so we know we can not push anything more than 15. Hence our maxflow value in this case is 15. The capacity of a cut is the sum of the capacities of edges from the s-side to the t-side, which in our case is $5 + 10 = 15$.

Observations

From this example we can make an observation that net outflow from s equals net inflow into t, represented by the equation below:

$$\sum_{u \in V} f(s \to u) - \sum_{w \in V} f(w \to s) = \sum_{u \in V} f(u \to t) - \sum_{w \in V} f(t \to w)$$

$$\text{net outflow from } s \qquad = \qquad \text{net inflow into } t$$