Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 06

Lecture 29

Welcome to the 29-th lecture of second level algorithms course in the last couple of lectures we have seen various algorithms for finding a maximum st flow in a graph in this lecture we will see various generalizations of the basic model okay so let's begin The first generalization is multiple source and multiple sink. In the basic max flow problem, we had only one source and one sink. But in a real world scenario, we can have multiple sources and multiple sinks. So let's see how we can solve the max flow problem from multiple source to multiple sink using the basic framework.

So, in this problem we have a set of source nodes Let's call the set S and a set of, let's call the set T of sink nodes. S and T are disjoint. we maintain flow conservation property we maintain flow conservation property at every node other than the nodes in $S \cup T$. So, flow conservation property is maintained at every node other than source and sink nodes. Flow conservation property means let us recall at every vertex other than source and sink nodes, total flow into the vertex should be the same as the total flow going out of the vertex and of course, we maintain capacity constraint at every age. So, here is a graph given and there are suppose T source nodes and L sink nodes. So, what we do?

We reduce this instance. To an S-T instance where I introduce a new vertex S and a new vertex T. S is the new source node, and the original k source nodes act as normal vertices. From the new source node, I add a directed edge to all S1 to SK, and their capacity is infinite. And here are the sink nodes. From all the sink nodes $T_1, \ldots, T_L$, I add an edge to T, the newly introduced sink node, with capacity infinity.

So, this source problem is multisource multisink max flow, whereas the reduced instance is S-T max flow. So, suppose I am given an instance of multi-source and multi-sink max flow. I need to compute a maximum S-T flow where S is the set of source nodes and T is the set of sink nodes. The value of the flow is the total outgoing flow from S, which is the same as the total net incoming flow to T. Now, to solve that problem, I create the instance of S-T flow like this, and then here is the claim. So, let us call the initial instance I_1.

For which I need to compute a maximum S-T flow, and the reduced instance is I_2, for which we have many algorithms to compute an S-T flow. So, here is a theorem. We can compute a maximum capital S capital T flow in I_1 from a maximum smallest multi-flow in I_2 in polynomial time. Proof.

Let f_2 from the H set of the instance I_2 to real numbers be a maximum S-T flow in I_2. So, in the reduced instance, I am given a maximum S-T flow, which I can compute using any of the algorithms that we have discussed. For example, Edmund Karp, Dinic's or push-reliable algorithm. define a flow f_1 from the edge set to real numbers as follows.

f_1 of e that means flow in any edge of instance I_1 is the same as f_2 of E for all edges E in edge set of I_1. So, f2 is a valid flow, sorry f_1 is a valid flow of I_1 since f_2 is a valid flow in I_2. f1 is a valid flow in I_1. since f_2 is a valid flow in I_2. We need to show that f1 is a maximum capital S to capital T flow in I_1.

We will prove this by contradiction so for contradiction suppose there exists a flow say $f_1^{'}$ in I_1 such that. value of $f_1^{'}$ is strictly more than value of f_1 okay then let us consider the flow $f_2^{'}$ defined as follows. $f_2^{'}$ of e is $f_1^{'}$ of e for all edges e in I_1, but I_2 has some more edges, namely from S to $S_1, \dots, S_k$. So, for those edges, we define the flow as the net flow out at S_i in $f_1^{'}$. This is for all $i \in [k]$. Similarly, we define the flow values of all edges from t_i to t for all $i \in [l]$. This is the net flow in at t_i in $f_1^{'}$ for all $i \in [L]$. So, now since $f_1^{'}$ is a valid flow in I_1, $f_2^{'}$ is a valid flow in I_2. Moreover, the value of $f_2^{'}$ is the same as the value of $f_1^{'}$, which is more than the value of f_1, which is the same as the value of f_2.

However, this contradicts our assumption that f_2 is a maximum st flow in I_2. So, f_1 is a maximum capital S to capital T flow in I_1.

Moreover, f_1 can be computed from f_2. in time b go of number of edges m. So, this concludes the proof of theorem. using the basic S-T max flow problem and algorithm for it we can solve more general problem where instead of giving single source and single sink we are given multiple sources and multiple sink nodes. Next let us look at another generalization where vertices also has capacity.

So, vertices also has capacity, edges also still have capacity, but what does vertex capacity mean? The total flow into a vertex which is or the total flow into a vertex which is the same as the total flow going out of the vertex other than s and t should be at most the capacity of the vertex. So, that is what the vertex capacity mean and from this problem onwards we will only do the reduced instance and the equivalence that the from a maximum flow of the reduced instance we can compute a maximum flow of the original instance. That proof is similar to the proof of the multi source multi sink version. So we will skip the proof and that proof proving those results I will keep it as a nice homework. So here is a standard instance of max flow S-T but every vertex v also has a capacity c_v.

So what do we do? We reduce this instance. So this is an instance I_1. So this v has some incoming edges and some outgoing edges. So I replace this vertex v by two vertices, which I am calling v_i and v_o, and I put an edge from v_i to v_o with capacity value c_v, the capacity of the vertices. v_i has only the incoming edges of v, and v_o has only the outgoing edges of v. So, this is the reduced instance I_2, and again, given a maximum s-t flow in G', we can compute in polynomial time a maximum s-t flow of G. which satisfies all the flow constraints and the vertex capacity constraint. The third one vertices can have demand. Every vertex v other than s and t has a demand d_v, and we need to ensure that flow into v minus flow out of v should be equal to demand d_v. This also This instance can also be reduced to an equivalent instance of maximum s-t flow, and that part I give as homework.

Arbitrary instance of max flow with vertex demands. To an equivalent instance of classical max flow. In polynomial time. So let us stop here. In the next lecture, we will see a couple more extensions, some of which are polynomial-time solvable, some of which are hard, and then we will discuss them.

So let us stop here. Thank you.