Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 23

Welcome to the 23rd lecture of the second-level algorithms course. In this lecture, we will see another algorithm for computing a maximum ST flow, which is the push-relabel algorithm. So, let us begin. So, till now, we have seen three algorithms for computing maximum ST flows. Let us review their performance.

The first one was the Ford-Fulkerson method. So, this works. Only if, or this works only for rational capacity values. The number of iterations is at most the sum of the capacities. This is for integral capacities, and in each iteration, I need to find an s to t path, which takes m plus m time.

Then we have seen a particular algorithm based on the Ford-Fulkerson method, which is the Edmonds-Karp algorithm. This works for arbitrary capacity values, and runtime is big O of m square n. The third algorithm that we have seen is the Dinic's algorithm. This also works for arbitrary capacity values.

Runtime is $O(mn^2)$ and in this lecture we will see even faster algorithm which is called push reliable algorithm. This also works for arbitrary capacity values. we go of n cube. So, you see that the runtime of Edmund Karp, Dinic's and push relabel does not depend on the value of the capacities assuming or in the word RAM model where accessing each word and operating on each word addition subtraction comparison takes bigger of one time. These sort of algorithms are called strongly polynomial time algorithm.

Observe that the runtime of Edmund Karp, Dinic's and push-relabel does not depend or do not depend on the input capacity values in the word RAM model of computation. Such algorithms are called strongly polynomial time algorithm. So, now let us begin the push-reliable algorithm. Here the algorithm design principle is fundamentally different from Edmund Scarp or Ford Fulkerson or Dinic's algorithm. So, those algorithms are called flow augmentation based algorithms. So, first let us review what was the underlying principle of flow augmentation based algorithms.

The underlying principle of flow augmentation-based algorithms. For example, Ford-Fulkerson, Edmund-Karp, Dinic's, etc. Is the following. They maintain a feasible solution in this case flow and then work towards optimality.

That is, making T disconnected from S in the residual graph. So, this is the high-level principle of flow augmentation-based algorithms. The high-level principle of push-relabel algorithm is the following. It maintains the optimality condition and we work towards feasibility.

So, the optimality condition here also means that T is disconnected from S in the residual graph. And we work towards feasibility, which means in every iteration of the push-relabel algorithm, the so-called solution is not a feasible solution—that means it is not a valid flow. But it is what is called a preflow. So, what is a preflow? Let us define it.

Preflow is an assignment from edges to non-negative real numbers, like flows, satisfying the following conditions. The first one is the same as flow, which is the capacity constraint. It means for all edges E, the flow value is less than or equal to the capacity value. So, this capacity constraint is satisfied by both flow and preflow.

It is the second condition where they differ. Preflow satisfies what is called the relaxed conservation property, which says that for all vertices V other than s and t, the total flow into V is greater than or equal to the total flow leaving V. For flow, these two must be the same. For preflow, the total flow going into V should be at least as much as the total flow leaving V. More formally, if I sum all the edges going into V that should be greater than or equal to the total flow of the edges going from V. In particular, the total flow going into V can be more than the total flow going out of V, and the difference we call excess at V. This is denoted by $\alpha\_f(V)$. which is the difference between the total flow going into V and the total flow leaving V, which is formally this quantity. So, in every iteration of the push-relabel algorithm, we will maintain a preflow and ensure that the optimality condition is satisfied in every iteration, namely, there is no s to t path in the corresponding residual graph. In every iteration, we will try to make this preflow into a flow, and when we are done, because we have been maintaining the optimality condition, the resulting flow must be an optimal flow. So, in every iteration of the push-relabel algorithm. we ensure optimality, that is, no s to t path in the residual graph. and we maintain a preflow. The only difference between a preflow and a flow is that in a preflow, there could be some vertices which have excess flow.

The high-level idea of this push-relabel algorithm is to push the excess flow at those vertices which have that excess flow into some neighboring vertices. Informally, in the push-relabel, we push excess flow from any vertex having an excess flow to its neighbor, ok? But as is obvious, if we do not do it systematically, then this flow—the excess flow—may circulate within the network.

However, trying to push the excess flow unsystematically may result in the circulation of the excess flow within the network. So, to systematize the pushing of excess flow, we introduce a concept of height of the vertices to make the push of excess flow systematic, we introduce the concept of

The height of the vertices. So, in the next lecture, we will see how using the concept of height in the vertices we will ensure that there is no flow circulation, and we are making progress toward turning the preflow into a flow, thereby terminating the algorithm and computing a maximum ST flow. So, let us stop here. Thank you.