Homework Assignment 1

October 5, 2023

Please write your homework exericse using Latex. Please send me your answers by the 19th of October.

You are welcome to discuss among yourselves. But everyone must write down his/her own answers.

Question 1

We do not have time to talk about the blocking flow method in class, a milestone in the max-flow algorithms. Let us do it here.

Given a directed graph $H = (V, F)$ with capacity $u : F \to \mathbb{R}_{\geq 0}$ and terminals s and t, a flow f is *blocking* if in the *subgraph* $\tilde{H} = (V, \tilde{F})$, where $e \in F$ is part of $\tilde{F}$ if and only if $f(e) < u(e)$, there is no s-t path. Notice that a blocking flow can have value a lot less than the maximum flow.

Next we define the *level* graph. Given $H = (V, F)$, we create a subgraph $L(H) = (V, F')$, where a directed edge $(u, v) \in F$ is part of F' if and only if $dist(s, v) = dist(s, u) + 1$, where $dist(s, x)$ denotes the shortest distance from s to the vertex x in H. Convince yourself that given H, the level graph $L(H)$ can be built in $O(|F|)$ time. We will also let $L(H)$ inherit the capacity from H.

Now let $G = (V, E)$ be the original network with capacity $c : E \to \mathbb{R}_{\geq 0}$ and terminals s and t. Dinic's algorithm builds a maximum flow f in G as follows.

1. Initilisation: $f = 0$.
2. Construct a blocking flow g in the level graph $L(G(f))$ of the residual network $G(f)$.
3. Augment f by g.
4. If there is no more s-t path in the residual network $G(f)$. Stop; otherwise, go back to Step 2.

The correctness of the algorithm is easy. The tricky thing is how to analyse its running time. In particular, how many times Step 2 is performed and how to implement this step efficiently.

Question 1(a)

Prove that the distance between s and t in the residual graph $G(f)$ increases each time Step 3 is performed. This automatically implies that Step 2 is performed at most $n-2$ times.

Hint: All comes from the first principle: check the definition of the residual network and the level graph carefully.

Question 1(b)

The next question is how to implement Step 2. Consider the special case that in G, all edges, except those incident on s or t, have capacity 1. Design an algorithm so that Step 2 takes $O(m)$ time. In other words, in this case, Dinic's algorithm takes $O(nm)$ time.

Remark: this may appear as an odd special case. But in fact there is an easy application: think about the bipartite b-matching. Here we are given a graph $G = (V, E)$ and a quota $b : V \to \mathbb{Z}_{>0}$ on the vertices. We want a maximum subset of edges $E' \subseteq E$ so that every vertex $u \in V$ is incident to at most $b(u)$ edges in E'.

Question 1(c)

This time assume that all edges have capacity 1. Furthermore, assume that every node, other than s and t, has at most k-outgoing edges or at most k in-coming edges. Prove that Dinic's algorithm takes $O(\sqrt{|V|k}|E|)$ time.

Hint: as a starter, assume that $k = 1$ and consider what happens after we have performed Step 2 $\sqrt{|V|}$ times.

Question 2

Given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}_{\geq 0}$, we want a maximum weight matching. Edmond has given a polynomial time algorithm to solve the problem exactly. The rest is history.

Nonetheless, here is another approach, called local search, to tackle the problem. The idea is simple. We start with an arbitrary matching M. We then check whether it is possible to remove up to p edges from M and add up to p edges from $E \backslash M$ into M so that the outcome is still a matching and the weight strictly increases. If so, we make the exchange and repeat the process. We stop the moment when no more improvement is possible; and we say we have reached a *local optimum*.

The limitation of exchanges up to p edges is apparently a device to control the running time, as the possible number of checks is then bounded by $O(|E|^p)$. Of course, in actual

implementation, there are smarter ways of keep the complexity down. (But there is another issue here. How do we know that the number of improvements is bounded by a polynomial? This is in fact not a serious issue. But in this exercise, for simplicity, let us assume the algorithm stops in polynomial time.)

Prove that the matching M obtained at a local optimum is a $(1 - \frac{1}{p})$-approximation.

Question 3

In class we presented an algorithm based on maximum flow to find the densest subgraph. Let us consider the following generalisation: given $G = (V, E)$, a 3-uniform hypergraph (that is, every edge $e \in E$ contains 3 vertices in V), give an algorithm to compute the densest sub-hypergraph of G.