Homework Assignment

November 30, 2021

Please type down your answer using Latex.

Question 1

We do not have time to talk about the blocking flow method, 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 residual network $H(f)$ 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 $\text{dist}(s, v) = \text{dist}(s, u) + 1$, where $\text{dist}(s, x)$ 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. Initialisation: $f = 0$.

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.

**Answer:** Consider an edge $e = (u, v)$ in $G(f)$ which is not part of $L(G(f))$. Then

$$\text{dist}(s, v) \leq \text{dist}(s, u).$$  \hspace{1cm} (1)

Notice that such an edge $e$ can be the reverse edge of another edge in $L(G(f))$. On the other hand, every edge $e = (u, v)$ in $L(G(f))$ must satisfy

$$\text{dist}(s, v) = \text{dist}(s, u) + 1.$$  \hspace{1cm} (2)

Observe that after augmentation, all edges of the first type and a subset of the second type still exist in $G(f + g)$. Moreover, some new edges can pop up in $G(f + g)$—these are the reverse edges used by the blocking flow $g$ in $L(G(f))$. But notice that even for them, Condition (1) still holds.

By the above observations, if we have an $s$-$t$ path in $G(f + g)$ with the same length from $s$ to $t$ as in $G(f)$, such a path must be composed of only edges in $L(G(f))$. But this would contradiction the assumption that $g$ is a blocking flow in $L(G(f))$.

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 $b : V \rightarrow \mathbb{Z}_{>0}$ 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 $E'$.

**Answer:** We repeatedly apply DFS on $L(G(f))$ to find a path $P$ from $s$ to $t$. Once such a path $p$ is found, we send a unit of flow along $p$ (thus removing all edges in $L(G(f))$ along $P$, except possible the first and the last edge). It can happen that in subsequent rounds, the DFS “backtracks” from an edge. In this case, we can just remove this edge permanently, it as will longer be useful. We stop when DFS can no longer give us a $s$-$t$ path.

To see that the complexity, suppose that in iteration $i$, we find path $P_i$ and have to backtrack from $x_i$ edges. Thus we spend $O(|P_i| + x_i)$ time for the DFS and flow.
augmentation. And after this DFS, the number of edges in $L(G(f))$ decreases by at least $|P_i| + x_i + 2$. Suppose that we augment the flow $u$ times, then $\sum_{i=1}^u |P_i| + x_i - 2 \leq m$. Trivially, $u \leq m$. We thus conclude that the total running time is $O(m)$.

**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 $k$ incoming edges. Prove that Dinic’s algorithm takes $O(\sqrt{|V|k|E|})$ time.

**Answer:** By 2(a) and 2(b), we know that after we run Step 2 $\sqrt{|V|k}$ times, the distance from $s$ to $t$ in the latest residual network $G(f)$ is at least $\sqrt{|V|k} + 1$. Furthermore, we have spent $O(\sqrt{|V|k|E|})$ time so far.

Recall that to find a max-flow in $G$, it suffices to find a maximum flow in $G(f)$ and use it to augment $f$. We claim that $G(f)$ has the max-flow value of at most $\sqrt{|V|k}$. If this is true, we then just have to run Step 2 another $\sqrt{|V|k}$ times, using in total $O(\sqrt{|V|k|E|})$ for the entire algorithm.

To see the claim, observe that every node other than $s$ and $t$ must still have at most $k$ outgoing or incoming edges in $G(f)$. Furthermore, the max-flow of $G(f)$ can be decomposed into $s$-$t$ paths $P_1, \cdots, P_x$. Each such path must use at least $\sqrt{|V|k}$ vertices that are neither $s$ nor $t$. By a counting argument, if $x > \sqrt{n}k$, at least a node $v \notin \{s, t\}$ is used more than $k$ times by these paths. A contradiction.

**Question 2**

We say a directed graph $G = (V, E)$ is $k$-edge-connected if there are $k$ edge-disjoint paths starting at $u$ and ending at $v$ for every $u, v \in E$.

Let us now add $k$ directed edges $(u_1, v_1), (u_2, v_2), \cdots, (u_k, d_k)$ in an arbitrary manner into the graph $G$. Prove that there are $k$ edge-disjoint cycles, each containing one of the $(u_i, v_i)$ edges.

**Answer:** Create an artificial vertex $r$ and add an directed edge from $r$ to each $v_i$. Easily one can verify that every $r$-cut has size at least $k$. Thus Edmonds’ arborescences theorem implies the existence of $k$ edge-disjoint $r$-arborescences. These latter then give us the paths from $v_i$ to $u_i$ for all the $i$ that we want.

**Question 3**

In class we do not have time to talk about the weighted version of the maximum matching in a general graph: given $G = (V, E)$ and a weight on the edges $w : E \to \mathbb{R}^+$, find a matching $M \subseteq E$ so that $w(M) = \sum_{e \in M} w(e)$ is maximised.
This problem can be solved by a polynomial time algorithm, as shown by Edmonds. In this exercise, we show that an LP-rounding approach can give us a 2/3-approximation.

Consider the following linear program:

\[
\sum_{e \in \delta(u)} x_e \leq 1 \quad \forall \text{ vertex } u \in V \tag{3}
\]

\[
x_e \geq 0 \quad \forall \text{ edge } e \in E \tag{4}
\]

Clearly, all matchings are contained in this polytope (but the extreme points of this polytope do not always correspond to matchings). Suppose that \(x^*\) is a fractional point in this polytope. Show how to round \(x^*\) into an integral matching \(M\) in polynomial time so that

\[
w(M) \geq \frac{2}{3} w(x^*) = \frac{2}{3} \sum_{e \in E} w(e) x_e^*. \]

For people who are already familiar with this material, please do not use Carathéodory’s theorem.

Hint: create a bipartite graph \(H = (V' \cup V'', F)\), where each vertex \(u \in V\) has two copies \(u' \in V'\) and \(u'' \in V''\) and, for each edge \(e = (u, v) \in E\), create \((u', v'')\), \((u'', v')\) in \(F\). The weights of \(F\) are inherited from \(w\) in a natural way. Observe that we can easily map a fractional solution from \(G\) to \(H\) and vice versa.

**Answer:** Define \(y^*\) using \(x^*\) in the obvious way: if \(x^*_uv = t\), set \(y^*u'v'' = y^*u''v' = t\). Observe that \(w(y^*) = 2w(x^*)\).

If the support of \(y^*\) contains an even cycle \(e_1, e_2, \ldots, e_{2d}\), and assume that \(\sum_{i=1}^{d} w(e_{2i-1}) \geq \sum_{i=1}^{d} w(e_{2i})\), we increase \(y^*\) on all \(e_{2i-1}\) and decrease \(y^*\) on all \(e_{2i}\) for all \(i\) until some edge is rounded to 0 and 1. Repeat this until the support is a forest. Apply a similar procedure on all paths of length \(> 1\) until the support of \(y^*\) is just a simple matching in \(H\) (convince yourself that the above process can be done while guaranteeing that \(y^*\) is a feasible fractional point in the polytope). At this point, in case that some support of \(y^*\) is less than 1, make it just 1. Let the final fractional be \(\overline{y}\) and by our rounding procedure, \(w(\overline{y}) \geq w(y^*)\).

We now map \(\overline{y}\) in \(H\) back to \(G\): define \(\pi\) by setting \(\pi_{uv} = \frac{y^*u'v'' + y^*u''v'}{2}\). Observe that

\[
w(\pi) = \frac{w(\overline{y})}{2} \geq \frac{w(y^*)}{2} = w(x^*),
\]

and \(\pi\) consists of isolated edges (the value of \(\pi\) on them is 1) and odd/even cycles (the value of \(\pi\) on them is 0.5).

We create \(M\) based on \(\pi\) as follows. Take all isolated edges. For even cycles, take every other edge along the cycle (of course take those that have larger total weight). For an odd cycle consisted of \(e_1, e_2, e_3, \ldots, e_{2d+1}\), suppose that \(e_2\) has the largest weight. Take \(e_2\) and either \(\{e_{2i}\}_{i=2}^{d}\) or \(\{e_{2i+1}\}_{i=2}^{d}\) according to their respective weights. It is easy to see that \(M\) is a matching and \(w(M) \geq \frac{2}{3} w(\pi)\).
Question 4

Let $G = (V, E)$ be the given graph, where $E$ is decomposed into a spanning tree $T$ and the rest $F$. Suppose that there is a weight $w : F \to \mathbb{R}_{\geq 0}$ on the edges $F$. A subset of edges $S \subseteq F$ is feasible if every edge $e \in T$ belongs to a cycle in $T \cup S$. Our objective is to find a feasible set $S$ with the minimum weight $w(S)$.

This is an NP-hard problem. But we can get a 2-approximation algorithm in polynomial time, using the branching algorithm.

Here is the hint. Create a weighted directed graph $H$ as follows. First choose an arbitrary vertex $r$ as the root and directed all edges of $T$ towards $r$. Next, for each edge $e = (u, v) \in F$, we create one or two arcs as follows. If $u$ is the ancestor of $v$, we create a directed arc from $u$ to $v$. If $u$ and $v$ are not ancestor/descendant of each other, let $x$ be their least common ancestor and create two arcs $(x, u)$ and $(x, v)$. What remains to be done is to define the weights properly to all the arcs in $H$ and use the branching algorithm.

Answer: For each edge $e \in T$, let $w'(e) = 0$. If $(u, v)$ is an arc connecting an ancestor to a descendant, let $w'(u, v) = w(u, v)$. Finally, in the case that $x$ is the least common ancestor of $u$ and $v$, let $w'(x, u) = w'(x, v) = w(u, v)$. Then find a min-weight arborescence $A$ rooted at $r$.

In the solution, if an arc $(u, v) \in A$, take its pre-image edge in $F$. It is easy to see that our final solution $A_f$ is feasible in $G$ and

$$w(A_f) \leq w'(A) \leq w'(OPT_H).$$

To show that $A_f$ is a 2-approximation, we argue that

$$w(OPT_G) \geq \frac{w'(OPT_H)}{2}.$$

We proceed by constructing a solution $\overline{A}$ in $H$ (an arborescence rooted at $r$ based on $OPT_G$ so that $w'(\overline{A}) \leq 2w(OPT_G)$. Let $\hat{A}$ be all arcs in $T$ and all those arcs generated by $F \cap OPT_G$. Easily, $w'(\hat{A}) \leq 2w(OPT_G)$. Furthermore, it is easy to verify that $\hat{A}$ is strongly-connected, implying that it contains an arborescence $\overline{A} \subseteq \hat{A}$ rooted at $r$ (for instance, do a DFS inside $\hat{A}$ starting at $r$). Thus $w'(\overline{A}) \leq w'(\hat{A}) \leq 2w(OPT_G)$. 

5