Homework Assignment

December 11, 2019

Please type down your answer using Latex. Please hand in your answers before the last class of the course.

Question 1

Given two matroids $M_1 = (S, \Pi_1)$ and $M_2 = (S, \Pi_2)$ and a weight function $w : S \to \mathbb{R}_{\geq 0}$, the objective is to find the common independent set $I \in \Pi_1 \cap \Pi_2$ so that $w(I)$ is maximised.

When all weights in $w$ are uniform, in class, we showed that Edmonds’ algorithm can find the optimal solution in polynomial time. Indeed, even when weights in $w$ are not uniform, the problem still can be solved in polynomial time (again by Edmonds).

On the other hand, a very straightforward Greedy algorithm is the following. Initially let $I = \emptyset$. We add the max-weight element $e$ into $I$ if $I + e \in \Pi_1$ and $I + e \in \Pi_2$ and we do this repeatedly until no more element can be added. Prove that the final outcome $I$ will be a $\frac{1}{2}$-approximation.

**Answer:** let $I = \{e_1, \ldots, e_t\}$ be the outcome of Greedy and $OPT$ be the optimal set. We define two injections $\phi_1$ and $\phi_2$ from $I$ to $OPT$ so that (1) $w(e_i) \geq w(\phi_1(e_i)), w(\phi_2(e_i))$ for all $i$, and (2) $\cup_{e_i \in I}\{\phi_1(e_i), \phi_2(e_i)\} = OPT$. This will be enough to establish that $I$ is a $0.5$-approximation.

The two injections can be constructed as follows. Add $e_1$ into $OPT$. This creates at most two circuits, $C_1$ in $M_1$ and $C_2$ in $M_2$. Define $\Phi_1(e_1)$ as the heaviest element in $C_1 - e_1$ and $\Phi_2(e_1)$ the heaviest element in $C_2 - e_2$. In case the circuit $C_1$ and/or $C_2$ does not exist, let $\Phi_1(e_1) = e_1$ and/or $\Phi_2(e_1) = e_1$. Now define a new independent set $OPT' = OPT + e_1 - \phi_1(e_1) - \phi_2(e_1)$. Continue the same process by adding $e_2$ into $OPT'$ and repeat.

Question 2

Given a bipartite graph $G = (A \cup B, E)$ and a weight function $w : E \to \{1, 2\}$, below is an algorithm that solves the maximum weight matching problem.
First consider the subgraph $G' = (A \cup B, E')$ consisting of only edges $E' = \{e|w(e) = 2\}$. Compute a maximum cardinality matching $M_1$ in $G'$. Find a minimum integral vertex cover $C_1$. (Here we write $C_1(v) = 1$ if $v \in C_1$, otherwise, $C_1(v) = 0$). Now create a second graph $G'' = (A \cup B, E'')$ consisting of edges $E'' = \{e = (a, b)|w(e) - C_1(a) - C_1(b) = 1\}$. Now again try to find a maximum cardinality matching $M_2$ in $G''$ under the condition that $M_2$ is derived from $M_1$ by augmenting) (so you should first prove that all edges in $M_1$ are still part of $E''$.)

Prove that $M_2$ will be the optimal solution.

**Answer:** Let $C_2$ be the minimum integral vertex cover $C_2$ in $G''$. Observe the following: (1) the sum of $C_1$ and $C_2$ form a weighted vertex cover of $G$, namely, for every edge $e = (u, v) \in E$, $C_1(u) + C_1(u) + C_2(u) + C_2(v) \geq w(e)$, (2) $\forall e = (u, v) \in M_2$, $w(e) = C_1(u) + C_1(u) + C_2(u) + C_2(v)$, and (3) for each vertex $v \in A \cup B$, if $C_1(v) + C_2(V) > 0$, $v$ is matched in $M_2$.

(1) implies that $C_1 + C_2$ form a dual solution, while (2) and (3) imply that $M_2$ and $C_1 + C_2$ satisfy the complementary slackness condition, therefore, by linear programming duality, $M_2$ is optimal matching.

**Question 3**

Prove that a graph $G = (V, E)$ has two edge-disjoint spanning trees if and only if the following condition hold:

For every $p$, if $V_0, V_1, \cdots V_p$ form a partition of $V$, then the number of edges with two endpoints in different $V_i$ is at least $2p$.

The following hint gives away almost everything. Create a second copy $G' = (V', E')$ of the graph, where $V'$ and $E'$ are just the copies of $V$ and $E$ respectively. In particular, every edge $e \in E$ will have a “twin” $e' \in E'$. Define a partition matroid $M_1$ over $E \cup E'$, where a set $I$ is independent if and only if $|I \cap \{e, e'\}| \leq 1$ for every $e \in E$. The second matroid $M_2$ is the graph matroid defined on the union of $G$ and $G'$. In particular, a set $I$ is independent if and only if there is no cycle in $I \cap E$ nor in $I \cap E'$. Now apply matroid intersection theorem.

**Answer:** Observe that given $E_1, E_2 \subseteq E$ so that $E_1 \cap E_2 = \emptyset$ and neither $E_i$ contains a cycle, $E_1 \cup E_2$ are a common independent set of $M_1$ and $M_2$. Therefore to prove that there are two disjoint spanning trees, we just have to argue that the minimum of

$$r_1(E \cup E' \setminus A) + r_2(A),$$

is at least $2(|V| - 1)$. Let $A$ be the minimizer. We can assume that if $e \in A$, then its twin $e' \in A$ as well and vice versa. Consider $G_i$ after we remove edges in $A$. Suppose that $G_i$ has $p + 1$ connected components, then $r_1(E \cup E' \setminus A) = 2(|V| - p - 1)$ and by the statement of the problem, $r_2(A) \geq 2p$. The proof follows.

2
Question 4

In this exercise we will develop a max-flow algorithm. It is interesting in that we try to augment a “flow” in the residual network $G(f)$, instead of a “path” in $G(f)$, as is done in the standard Ford-Fulkerson algorithm.

Let us recall a few facts. For each directed edge $e = (u, v)$ in the original graph $G = (V, E)$, we create (potentially) two arcs in $G(f)$, $e_1$ from $u$ to $v$, with capacity $c(e) - f(e)$, and $e_2$ from $v$ to $u$ with capacity $f(e)$. $G(f)$ by itself then is also a network with edge capacity. Suppose that $\tilde{f}$ is a flow in $G(f)$. Then we can augment $f$ by $\tilde{f}$ as follows: for every edge $e \in E$, if $e_1$ and $e_2$ are the two corresponding edges in $G(f)$, let $f(e) := f(e) + \tilde{f}(e_1)$ (if $e$ and $e_1$ are in the same direction) and $f(e) := f(e) - \tilde{f}(e_2)$ (if $e$ and $e_2$ are in the opposite direction). It is easy to verify that $f$ remains a flow and its value is increased by exactly the value of $\tilde{f}$.

So the idea is that if we can construct a flow of (relatively) large value in $G(f)$, we do not have to augment $f$ too many times. We now introduce a subroutine, which is helpful in constructing a large flow in $G(f)$ later on.

**Subroutine X**

Step 0: $b(v) := 0$ and $L_v = \emptyset$ for all $v \in V$; $i = 1$. Let the source $s$ be node 1. $W = \{1\}$.

Step 1: For each node $w \in V \setminus W$, if there exists a node $i \in W$ so that $(i, w)$ is an arc in $G(f)$, let $b(w) := b(w) + c(i, w)$ and add the arc $(i, w)$ into the list $L_w$.

Step 2: Let the node $v \in V \setminus W$ be the node with the largest $b(v)$. Call $v$ node $i + 1$. $W := W \cup \{i + 1\}$. $i := i + 1$.

Step 3: If the newly added node is the destination $t$, stop. Otherwise, go back to Step 1.

You should be able to verify easily the following facts. They will be useful later.

1. The final set $W = \{1, 2, \cdots, k\}$ and all the lists $\bigcup_{i=1}^{k} L_i$ together define an acyclic subgraph $H$ of $G(f)$ and in fact the numbers $\{1, \cdots, k\}$ respects the topological order of $H$.

2. For each node $x \in W$, $b(x) = \sum_{e \in L_x} c(e)$.

3. In the beginning of Step 1, 

$$\sum_{e=(i,j),i \in W,j \notin W} c(e) = \sum_{j \in V \setminus W} b(j).$$
The subroutine $X$ can be implemented in $O(m + n \log n)$ time by adapting Dijkstra’s shortest path algorithm. You can take this for granted. We can now define the formal algorithm.

**Main Algorithm**

Step 0: Define $f := 0$ in $G$.

Step 1: Apply Subroutine $X$ on $G(f)$. Suppose that the final set $W = \{1, \cdots, k\}$. Let $\delta := \min_{i \in W} b(i)$. If $\delta = 0$, stop the algorithm; otherwise, define $\beta(k) := \delta$ and $\beta(v) := 0$ for each $v \neq k$. $\tilde{f} := 0$.

Step 2: For $i = k$ down to 1, do the following.

For each arc $e = (u, i)$ in $L_i$: $\tilde{f}(e) := \min\{\beta(i), c(u, i)\}$. $\beta(i) := \beta(i) - \tilde{f}(e)$ and $\beta(u) := \beta(u) + \tilde{f}(e)$.

Step 3: Augment $f$ by $\tilde{f}$. Go back to Step 1.

**Question 4(a)**

Prove that $\tilde{f}$ as constructed in Step 2 is a flow of value $\delta$ in $G(f)$.

**Answer:** we observe the following invariant: After vertex $x$ is processed,

- If an edge $e = (i, j)$ has $\tilde{f}(e) > 0$, $j \in \{x, \cdots, k\}$.
- $\sum_{i < x, j \geq x} \tilde{f}_{ij} = \delta$.
- $\sum_{i, k} \tilde{f}_{i,k} = \delta$.
- For $x \leq h < k$, $\sum_{i < h} \tilde{f}_{ih} = \sum_{j > h} \tilde{f}_{hj}$, except that $x = h = 1$ (the source node).
- $\tilde{f}(e) \leq c(e), \forall e \in G(f)$.

By this invariant, after all vertices are processed, the final $\tilde{f}$ is a flow in $G(f)$.

This invariant can be established by induction on (decreasing) value of $x$. Suppose that we now process node $x$. By the invariant, node $x$ should have outgoing flow of value at most $\delta$. The capacities of incoming arcs in $L_x$ should be at least $\delta$, otherwise, $b(x) > \delta$, a contradiction to the choice of $\delta$. By this fact, we can easily establish the invariant.
Question 4(b)
Prove that if there is an augmenting path $P$ of value $\alpha$ in $G(f)$, then after Step 3, the flow value of $f$ increases by at least $\alpha$. In particular, if $\delta = 0$, there is no augmenting path in $G(f)$.

Hint: It suffices to argue that if there is an augmenting path of value $\alpha$, in Subroutine $X$, $b(i) \geq \alpha$ for all $i$ in the final set $W$.

**Answer:** During the execution of subroutine $X$ when $W = \{1, \cdots, i\}$ and destination $t \notin W$, there must exist a node $u \in W$ and another node $v \in V \setminus W$, so that $(u, v)$ is an edge in the augmenting path $P$. Then $b(v)$ is increased by at least $c(e)$, which is at least $\alpha$. In other words, the next node $i + 1$ to be added into the set $W$ must have $b(i + 1) \geq \alpha$.

Question 4(c)
Suppose that $\delta > 0$ in Step 1. Prove that $\tilde{f}$ constructed at the end of Step 2 has value at least $v(f_{opt}) - v(f)$. Here $f_{opt}$ is the maximum flow in the original network $G$ and $v(\cdot)$ refers to the value of the flow.

Hint: If we can show that the min-cut in $G(f)$ is at most $n\delta$, then by the previous exercise, we can easily establish the answer. Now suppose that $\delta = b(x)$ for some $x$ in the final $W$. How large is the cut $W' = \{1, \cdots, x - 1\}$ in $G(f)$?

**Answer:** Suppose that in the Subroutine $X$ we just finish processing vertex $x - 1$. Then

$$\sum_{e=(i, j), i \in W', j \notin W'} c(e) = \sum_{j \in V \setminus W'} b(j) \leq |V \setminus W'| b(x) \leq n\delta.$$

To see the first inequality, note that $x$ is chosen to be the one which has currently large $b$-value in $V \setminus W'$.

To finish, observe that $v(f_{opt}) - v(f)$ is exactly the value of the maximum flow in $G(f)$, which is upper-bounded by the minimum cut-size of $G(f)$, and we know it is at most $n\delta$ by the preceding inequality. Now by the fact established in Question 4(b), we obtain the answer.

Question 4(d)
Suppose that all edge capacities $c(e)$ are integers in the original network $G$ and $U = \max_{e \in E} c(e)$. Prove that the running time of this algorithm is $O(n(m + n \log n) \log nU)$.

5
**Answer:** let $f_1$ and $f_2$ be the two flows before and after Step 3 respectively. By the previous question, we know

$$v(f_{opt}) - v(f_2) \leq (1 - 1/n)(v(f_{opt}) - v(f_1)).$$

In other words, after $O(n)$ rounds of the algorithm, the difference in the flow values of the optimal and the current flow is at least halved. As $v(f_{opt})$ is at most $nU$, we conclude that we only need $O(n \log nU)$ rounds. Now as Steps 2 and 3 can be implemented in $O(m)$ time, we know the bottleneck lies in Subroutine $X$, which takes $O(m + n \log n)$ time. Now we arrive at the answer.

**Question 5**

The preflow-push algorithm gives the max-flow and of course the min-cut as well. But if we only want the min-cut, in fact, we do not have to run the algorithm to the very end. You can modify it in such a way so that we find the min-cut faster (not asymptotically, but at least improved by a constant factor).

The following observation is useful. When the distance label $d(v)$ of a node reaches $n$, there is something special about it. Build on this.

**Answer:** when $d(v) = n$, there is no more directed path from $v$ to $t$ in the residual network. So let us twitch the algorithm just a little bit: if an active node $v$ already has $d(v) \geq n$, we do not raise its label further. Consider the point that the algorithm is “blocked.” Let $U$ be the set of nodes having a directed path to $t$ in $G(f)$. We claim that $U$ is a min-cut.

Note $U$ contains $t$ but not $s$ and every node $v$ in $U$ has $d(v) < n$ and is non-active. At this point, every edge $e$ from $V \setminus U$ to $V$ (in the original network) has $f(e) = c(e)$ and from $U$ to $V \setminus U$ (in the original network) has $f(e) = 0$. If we now “unblock” the algorithm, no more flow will be ever exchanged between $V \setminus U$ and $U$ (in either direction). In other words, the final flow value is exactly the cut size of $(V \setminus U, U)$. 

6