**15-850: Advanced Algorithms** **CMU, Spring 2024**

**HW #2** (out Monday Feb 12, 2024) **Due:** Monday February 26, 2024

Same as HW1: Collaboration in a group of 2-3 is encouraged. Please solve *two* of the four problems.

1. **Spans, Ranks, and Duals.** In HW1 you saw matroids; we'll explore them a little more in this problem. Let $\mathcal{M} = (U, \mathcal{I})$ be a matroid. A few more definitions:

   - *Rank of a set.* For $S \subseteq U$, let $\mathsf{rank}(S)$ be the size of a largest independent set inside $S$; i.e., $\mathsf{rank}(S) = \max\{|T| \mid T \subseteq S, T \in \mathcal{I}\}$.
   - *Span of a set.* For $S \subseteq U$, let $\mathsf{span}(S)$ be the largest $T \subseteq U$ such that $S \subseteq T$ and $\mathsf{rank}(S) = \mathsf{rank}(T)$. *(We will see in part (d) that this is well-defined.)*

   We use $S + e$ to denote $S \cup \{e\}$, and $S - e$ to denote $S \setminus \{e\}$.

   (a) (Do not submit) Show that $S \in \mathcal{I}$ if and only if $\mathsf{rank}(S) = |S|$.

   (b) Show the following inequality for any $S \subseteq T \subseteq U$, and any $e \in U$:

   $$\mathsf{rank}(S+e) - \mathsf{rank}(S) \geq \mathsf{rank}(T+e) - \mathsf{rank}(T).$$

   (c) Using part (b) repeatedly, prove that for any $A, B \in U$,

   $$\mathsf{rank}(A) + \mathsf{rank}(B) \geq \mathsf{rank}(A \cup B) + \mathsf{rank}(A \cap B).$$

   This shows the rank function is "*submodular*".

   (d) (Do not submit) Suppose $T_1$ and $T_2$ each satisfy $S \subseteq T_i \subseteq U$ and $\mathsf{rank}(S) = \mathsf{rank}(T_i)$. Show that $\mathsf{rank}(S) = \mathsf{rank}(T_1 \cup T_2)$. Deduce that $\mathsf{span}(S)$ is well-defined.

   (e) (Do not submit) For $S \subseteq U$, let $X(S) = \{e \in U \mid \mathsf{rank}(S+e) = \mathsf{rank}(S)\}$. Prove that $X(S)$ is identical to $\mathsf{span}(S)$.

   (f) Suppose each element in $U$ has a distinct non-negative weight $w_e \in \mathbb{R}_{\geq 0}$. Consider the following LP for the problem of finding the max weight independent set:

   $$\begin{aligned} \max \sum_e w_e x_e & \\ s.t. \quad \sum_{e \in A} x_e \leq \mathsf{rank}(A) & \qquad \forall A \subseteq U, A \neq \varnothing \\ x_e \geq 0 & \end{aligned}$$

   For any set $S \subseteq U$, let $\chi_S \in \{0,1\}^{|U|}$ be its characteristic vector. Show that $S \in \mathcal{I}$ if and only if $\chi_S$ is feasible for this LP.

   (g) Write down the dual of this LP.

   (h) The greedy algorithm starts with the empty set, and each time picks the max-weight element $e$ that does not lie in the span of the current set of picked elements. Let $G$ be the set eventually output by the greedy algorithm, and let $w(G) := \sum_{e \in G} w_e$ be the weight of this set. Give a feasible dual solution $y^*$ such that the dual objective function equals $w(G)$ the primal objective function.

   *Hint: If greedy picks elements $g_1, g_2, \ldots, g_r$ in this order, just focus on the sets $R_i := \mathsf{span}(\{g_1, g_2, \ldots, g_i\})$ for $i \in [r]$. Also, if you need something concrete to think about, think about the "graphical" matroid where $U$ is the set of edges of a graph and $\mathcal{I}$ are the acyclic subsets.*

2. **Sparse Spanners.** Given an undirected graph $G$ with edge lengths $\ell_e$, a subgraph $H$ is a *spanner* with stretch $\gamma \geq 1$ if for every edge $(x, y) \in E(G)$,

$$d_H(x, y) \leq \gamma \cdot d_G(x, y).$$

Spanners are the deterministic analogue of (probabilistic) low-stretch (spanning) trees, at the expense of more edges.

(a) (Do not submit) Use the triangle inequality to show that for all $x, y \in V$, even if $(x, y)$ is not an edge, $d_G(x, y) \leq d_H(x, y) \leq \gamma \cdot d_G(x, y)$.

Clearly if $H = G$, we can set $\gamma = 1$. The goal is to find $H$ with few edges, such that $\gamma$ is also small. Let's give two different constructions of good spanners.

(b) *Approach #1.* Sample $t = 4 \log n$ trees $T_1, T_2, \ldots, T_t$ from an $\alpha$-stretch (randomized) low-stretch *spanning* tree, i.e., a low-stretch tree whose distribution is on spanning trees of the graph with the same edge lengths. Let $H$ be the union of all these edges.

i. Show that for any fixed edge $(x, y) \in E(G)$,

$$\mathbf{Pr}[d_H(x, y) \geq 2\alpha\, d_G(x, y)] \leq 2^{-t}.$$

(Hint: for any single value of $i$, bound $\mathbf{Pr}[d_{T_i}(x, y) \geq 2\alpha\, d_G(x, y)]$.)

ii. Use the following result by Abraham and Neiman to show that with probability $1 - \frac{1}{n^2}$, the graph $H$ is an $O(\log n \log \log n)$-stretch spanner with $O(n \log n)$ edges.

**Theorem 1** (Abraham-Neiman 2012)**.** *For any graph, there is a low-stretch spanning tree distribution with stretch $\alpha = O(n \log n \log \log n)$.*

(c) *Approach #2.* It turns out that constructing spanners is far simpler than constructing low-stretch (spanning) trees. In fact, a simple greedy algorithm suffices.

i. *Large-girth graphs are Sparse.* Define the *girth* of graph $G$ to be smallest number of edges on any cycle in $G$. We first show that any graph $G$ with $m$ edges and $n$ nodes, and girth *strictly more than* $g$ must have $m \leq O(n + n^{1+1/\lfloor g/2 \rfloor})$.

A. The average degree of $G$ is $\bar{d} := \frac{2m}{n}$. Show that there exists a subset $S \subseteq V$ such that the induced subgraph $H := G[S]$ has minimum degree at least $\bar{d}/2$. [Hint: drop some low-degree vertices.]

B. For this graph $H$ and any vertex $v \in H$, show that the number of distinct vertices reachable within $\lfloor g/2 \rfloor$ hops from $v$ is at least $(\bar{d}/2 - 1)^{\lfloor g/2 \rfloor}$.

C. Prove: the number of edges $m$ in the original graph $G$ satisfies $m \leq O(n + n^{1+1/\lfloor g/2 \rfloor})$.

ii. The algorithm is a variant of Kruskal's algorithm for $\alpha \geq 1$. Consider the edges of $G$ in increasing order of lengths $e_1, e_2, \ldots, e_m$. Initialize $H_0 = \varnothing$. When considering edge $e_i = (x, y) \in E(G)$, if the current distance $d_{H_{i-1}}(x, y) \leq \alpha\, d_G(x, y)$, then discard $e$ (i.e., set $H_i \leftarrow H_{i-1}$), else take it (i.e., set $H_i \leftarrow H_{i-1} \cup \{e_i\}$).

A. (Do not submit.) Show that if we set $\alpha = n - 1$, then you will get Kruskal's algorithm. Also, observe that by construction, the graph $H$ at the end of the process is an $(n-1)$-stretch spanner. (In fact, an $(n-1)$-stretch spanning tree.)

B. If we set $\alpha = O(\log n)$, use (c) with $g = \Theta(\log n)$ to show the final graph $H$ is a $O(\log n)$-stretch spanner with $O(n)$ edges.

C. (Do not submit.) What about different values of $\alpha$? What kind of tradeoff do we get? This is called the *spanner tradeoff* and is asymptotically tight assuming a popular girth conjecture of Erdös.

3. **Min-weight Perfect Matchings.** Suppose $G = (L, R, E)$ is a undirected bipartite graph with (possibly negative) *integer* edge weights $w_e$, suppose $M$ is some perfect matching. Let $H_M$ be the digraph obtained by directing edges in $E \setminus M$ from left-to-right and putting weights $w_e$ on them, and directing all edges in $M$ from right-to-left and putting weights $-w_e$ on them.

   (a) Prove: $H_M$ has a negative-weight cycle iff $M$ is not a min-weight perfect matching. [Hint: take the difference with a min-weight perfect matching.]

   (b) Consider the algorithm: Start with any perfect matching $M$. While there is a negative-weight cycle $C$ in $H_M$, set $M \leftarrow M \triangle C$. Show that you will eventually get a MwPM.

   If $T$ is the time to find a negative-weight cycle in an $n$-node graph, and $W := \max_{e \in E} |w_e|$, a naive bound on the runtime of the above algorithm would be $O(nW) \cdot mn$, plus the time to find the first perfect matching. (Do you see why? Recall that Bellman-Ford can find the negative-weight cycle in time $T = O(mn)$.)

   (c) For matching $M$, define $w(M) := \sum_{e \in M} w_e$. Show that if $w(M) > w(M^*)$ then there exists a cycle in $H_M$ with weight-ratio (defined in HW#1) no more than $\frac{w(M^*) - w(M)}{n}$, where $M^*$ is a min-weight perfect matching.

   (d) Change the algorithm in (b) to say *"While there exists a negative-weight cycle in $H_M$, let $C$ be the minimum weight-ratio, and set $M \leftarrow M \triangle C$"*. Bound the runtime of this algorithm by $O(n \log(nW)) \cdot T'$, plus the time to find the first perfect matching. Here $T'$ is the time to find a min weight-ratio cycle.

   Since in HW1(#4) we gave an algorithm to find the min weight-ratio cycle in time $T' = O(mn(\log(nW)))$, we get an algorithm that finds a MwPM in time $O(mn^2(\log(nW))^2)$.

4. **A Market-Based Bipartite Max-Matching Algorithm.** Given $G = (I, B, E)$, the left vertices are items, the right are buyers. Let $n = \max(|I|, |B|)$. For each edge $ib \in E$, let $v_{ib} = 1$; if $ib \notin E$, then $v_{ib} = 0$. The initial prices are $p_i = 0$; define the utility of buyer $b$ for item $i$ under prices $\bar{p} = (p_1, p_2, \ldots, p_{|I|})$ to be

$$u_{ib}(\bar{p}) = \max\{v_{ib} - p_i, 0\}.$$

   For some parameter $\delta \in (0, 1)$, consider the following algorithm:

   (A1) Start with the empty matching $M = \varnothing$.

   (A2) Pick any unmatched buyer $b$ such that its highest-utility item $i$ has $u_{ib}(\bar{p}) \geq \delta$. Match $(i, b)$, which may require dropping $(i, b')$ for some other $b'$ from the current matching. (So now $i$ is assigned to $b$ instead of $b'$.) Raise the price $p_i \leftarrow p_i + \delta$.

   (A3) If for each unmatched buyer, $\max_i u_{ib}(\bar{p}) < \delta$, let the current matching be denoted $M_1$. Use the augmenting paths algorithm to augment $M_1$ to a maximum matching $M^*$.

   We now show that $M_1$ is a "large" matching, and so step (A3) does not take "much" time.

(a) Show that $|M_1| \geq |M^*| - n\delta$.

(Hint: you could try to use the following LP:

$$\begin{aligned} \max \textstyle\sum_{ib} v_{ib} x_{ib} & \\ \textstyle\sum_b x_{ib} \leq 1 & \qquad \forall i \in I \\ \textstyle\sum_i x_{ib} \leq 1 & \qquad \forall b \in B \\ x_{ib} \geq 0 & \end{aligned}$$

And show that $|M_1| \geq OPT_{LP} - n\delta$ using the duals.)

(b) Show how to use the appropriate data structures to implement the algorithm so that all executions of step (A2) can be done in a total of $O(\frac{1}{\delta} \cdot (m + n \log n))$ time.

(c) Choose the value of $\delta$ so that the running time of the entire algorithm is as close to $O(m\sqrt{n})$ as possible. (You may assume that each augmenting path can be found in $O(m)$ time, and hence (A3) takes a total of $O(m) \times (|M^*| - |M_1|)$ time.)

*Observe that this algorithm is different from the one in lecture: there the prices went up in lock-step, here we just pick a single item and unilaterally raise its price by a tiny constant. We also don't get a perfect matching here at the end, we have to do some correction (Step (A3)). BTW, what can you say about the performance of such an algorithm when you have general weights? (You don't have to submit any answers for these.)*