CS 670 (Fall 2022) — Assignment 1

Due: 09/12/2022

In all homeworks, you can use all facts from class and from the textbook without reproving them.

Note: the solutions to many homework problems can probably be found fairly easily online or in textbooks. Recall that using these resources to solve homework problems is considered cheating, and will result in corresponding sanctions.

(1) [10 points]

The VCG (Vickrey-Clarke-Groves) mechanism is a celebrated mechanism/algorithm in economics (three Nobel prizes). It prescribes a general way to adjust payments to get agents to reveal their costs/values truthfully. Here, we look at the following setting. There are n agents. Subsets of the agents together can perform a crucial task; we call such sets *feasible*, and denote the collection of all feasible sets of agents by $\mathcal{F}$. Agents have costs c_e for participating in a team, and the VCG mechanism will pick the feasible set of agents with the smallest sum of costs. The crucial insight of the VCG mechanism is that you shouldn't just pay an agent c_e, or they may lie to get paid more. Rather, you should pay an agent a bonus equaling how much more expensive a solution would be without this agent around.

To make this precise, let T^* be the cheapest feasible solution (minimizing $\sum_{e\in T} c_e$ among solutions $T \in \mathcal{F}$). Then, for each agent $e \in T^*$, let T_e^* be the cheapest feasible solution (again minimizing $\sum_{e\in T} c_e$) among solutions $T \in \mathcal{F}$ with $e \notin T$, i.e., not using e. Then, agent e should be paid $p_e := c_e + c(T_e^*) - c(T^*)$. For example, if there is an equally cheap solution without the agent, the agent does not get any bonus. On the other hand, if an agent e is indispensable, in the sense that every feasible solution contains e, then e must be paid an infinite amount — this "makes sense" as in order to get any feasible set, e must be included, so they can charge whatever they want. Notice that the bonus is always non-negative, so each agent gets paid at least their cost. From an economics perspective, the cool insight about VCG is that agents have no incentive to lie — they always get at least the same payment by revealing their true costs. While this is actually not hard to prove, that's not what this problem is about.

One way to determine payments would be simply to find the cheapest solution T^* and pay each $e \in T^*$ the amount p_e defined above. Instead, one might try to lower total payments by choosing for each agent $e \in T^*$ some value $b_e \geq c_e$ with the property that for every agent $e \in T^*$, there exists some solution T_e' with the property that $\sum_{e\in T^*} b_e = \sum_{e\in T_e'} b_e$. This way, with the proposed payments b_e, each agent e could be left out without increasing the cost. One might hope that the total payment $\sum_{e\in T^*} b_e$ might now be smaller. It turns out that for general set systems, this can well happen. Here, you are to prove that it does not happen for matroids.

More precisely, we assume that the feasible sets are exactly the bases[1] (maximal independent sets) of a matroid;[2] furthermore, to avoid infinite payments, we assume that $\bigcap_{S\in\mathcal{F}} S = \emptyset$, so that no agent is indispensable. Let $c_e \geq 0$ be arbitrary costs for the agents. Let T^*, T_e^* and the p_e be defined as above. Let the b_e minimize $\sum_{e\in T^*} b_e$ among all assignments with $b_e \geq c_e$ for all e and the additional property that for each e, there exists a set T_e' not containing e with $\sum_{e\in T_e'} b_e = \sum_{e\in T^*} b_e$. Prove that for all such assignments b_e, we have $\sum_{e\in T^*} b_e \geq \sum_{e\in T^*} p_e$, so no payment can be saved by trying to assign these b_e.

For your proof, the following stronger-looking version of the exchange property is probably going to be useful — you can use it without proving it. (The proof is not super-hard, but definitely not easy, either.)

Proposition 1 *Let $\mathcal{B}$ be the bases (maximal independent sets) of a matroid over elements E. Then, for every two sets $S, T \in \mathcal{B}$, there is a bijection f between $S \setminus T$ and $T \setminus S$ such that for all $e \in S \setminus T$, the set $S \setminus \{e\} \cup \{f(e)\} \in \mathcal{B}$.*

[1] We could of course include all supersets of the bases as well — the mechanism will never pick them, because they contain redundant agents.

[2] In fact, one can show that the following statement is true if and only if the feasible sets are the bases of a matroid. But you only need to show the "if" direction — the "only if" is a bit more complicated.

Just as a reminder, recall that a *matroid* is a set system $(E, \mathcal{I})$, where $\mathcal{I} \subseteq 2^E$, with the following properties:

- $\emptyset \in \mathcal{I}$ (non-trivial).
- If $S \in \mathcal{I}, S' \subseteq S$, then $S' \in \mathcal{I}$ (downward closed).
- If $S, S' \in \mathcal{I}, |S| < |S'|$, then there exists an $e \in S' \setminus S$ such that $S \cup \{e\} \in \mathcal{I}$ (exchange property).

(2) [10 points]
In class, we saw that Dijkstra's Algorithm cannot be used as is for graphs with negative edge weights. Here, we will explore a way to make this work, so long as the input graph G has no negative cycles, i.e., cycles C with $\sum_{e \in C} w_e < 0$. (If you have negative cycles, then the notion of "shortest path" is not even well defined.)

Our input is a (let's say directed) graph $G = (V, E)$ with edge weights w_e, which could be positive or negative. Add a new node s and connect it to each original node $v \in V$ with a directed edge of weight $w_{(s,v)} = 0$. Now use some shortest-path algorithm that works with negative edge weights (such as Bellman-Ford, which most of you know, and we will probably briefly review in about two weeks) to find the shortest-path distance from s to each node v. Call this distance b_v; it could be negative. Now, change the weights of the edges $e = (u, v) \in E$ in the original graph to $w'_e = w_e + b_u - b_v$.

(a) Prove that the new weights w'_e are all non-negative.

(b) Prove that for each pair (u, v) of nodes in the original graph, if we compute the shortest-path distance from u to v with respect to the new weights w'_e, and then subtract $b_u - b_v$ from this distance, we get the shortest-path distance from u to v with respect to the original edge weights w_e.

(In case you're wondering what's the point if we're going to call Bellman-Ford anyway: Bellman-Ford is slower than Dijkstra. By calling Bellman-Ford n times (once for each node), we can now compute the shortest paths for all $\Theta(n^2)$ pairs using Dijkstra, which is faster.)

(3) [10 points]
Here is yet another Minimum Spanning Tree algorithm. As always, $G = (V, E)$ is an undirected connected graph with non-negative edge weights w_e, and you can assume that all w_e are distinct.

Algorithm 1 Yet Another Minimum Spanning Tree Algorithm (YAMSTA)

1: **while** there is more than one node remaining **do**
2: Each node v chooses its cheapest incident edge e_v.
3: Contract all chosen edges e_v, resulting in each component (of the graph with edges e_v) becoming one new node. (The new node inherits all edges incident on any of its nodes, but only keeps the cheapest among parallel edges.)
4: Output the set of all edges e_v (across all iterations).

One iteration of the algorithm is illustrated in Figure 1.

(a) Prove that in each iteration of this algorithm, the set of edges e_v that are contracted is acyclic.

(b) Prove that the algorithm computes a minimum spanning tree.

(c) Explain (and analyze) how to implement the algorithm to run in time $O(m \log m)$.

(4) [10 points]
When defining Fibonacci Heaps, we had the rule that once a node lost its *second* child, it itself was removed from the tree (together with all its subtrees). Suppose we changed this rule to "once a node loses its third child, it is removed from the tree". Does the amortized asymptotic time per operation stay the same? In other words, does a sequence of b inserts, c deletes-mins, and d decrease operations still take time $O(b + d + c \cdot \log n)$? Resolve this question by doing exactly one of the following two:

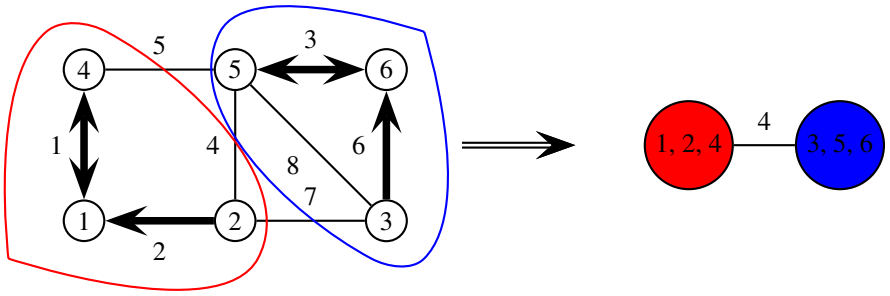

Figure 1: An illustration of a contraction step of the algorithm. The arrows go from the node selecting the edge to the other endpoint. Bold edges are selected, others not. The colored blobs show the contracted components.

(a) proving that the time stays the same. If you do this, you can use without proof the parts of the analysis that stay the same; you only need to focus on the parts that change, and explain how those changes affect the analysis.

(b) giving a sequence of operations for which the asymptotic (that is, big-Oh) time is longer. If you do this, you should of course also explain why your sequence of operations causes an asymptotic slowdown.

(5) [0 points]

Chocolate Problem (1 chocolate bar): On most/all of the homeworks, we will also have a chocolate problem. Chocolate problems are significantly harder, and they do not per default affect your grade. The reward is actual chocolate.[3] Chocolate problem solutions should be submitted separately from the rest of your solution, since David (not Yusuf) will grade them. You are welcome to solve chocolate problems as a team — in that case, you split the chocolate reward. Optionally, you can also designate a chocolate problem as a substitute for one regular problem; in that case, your score for the chocolate problem will be substituted for the regular problem.[4] Obviously, if you substitute the chocolate problem score for the regular problem, you cannot solve the chocolate problem as a team.

The chocolate problem for Homework 1 is the following, somewhat motivated by a recent pandemic you may have heard about. You are given an undirected graph (social network) in which each edge $e = (v, v')$ has an interval $I_e = [\ell_e, u_e]$ on it. The meaning is that you *know* that v and v' met at some point during the interval I_e, but have no idea when exactly. You also know that some source node s started with some infectious disease, and that a set $P \subseteq V$ ended up with the disease, while a disjoint set $N \subseteq V$ definitely did not catch the disease. (There may be nodes that are neither in P nor in N.) The disease has the property that when two nodes v, v' meet, one of whom has the disease, it is instantaneously transmitted to the other, who is also immediately infectious.

Give (and analyze, i.e., prove correctness and running time) a polynomial-time algorithm which is given G, P, N and the I_e, and either outputs meeting times $t_e \in I_e$ for each edge that explain the observations, or (correctly) concludes that it is impossible that all of P got the disease while no one in N did.

[3] If you have preferences/allergies, feel free to note them on your solution.

[4] This is so that you don't have to solve more problems which might be a bit tedious. Remember that chocolate problems are much harder.