CSE 6140 / CX 4140

Computational Science & Engineering Algorithms

Final Exam

Please type all answers.

1. (20 points)

This problem analyzes group decision making. Consider a dataset that consists of decisions made by a particular governmental committee. We want to identify a small set of influential members of the committee.

Here's how the committee works. It has a set $M = \{m_1, ..., m_n\}$ of n members, and over the past year it has voted on t different issues. On each issue, each member can vote either "Yes", "No", or "Abstain"; the overall effect is that the committee presents an affirmative decision on the issue if the number of "Yes" votes is strictly greater than the number of "No" votes (the "Abstain" votes don't count for either side), and it delivers a negative decision otherwise.

We have a table consisting of the vote cast by each committee member on each issue. We'd like to consider the following definition. We say that a subset of the members $M' \subset M$ is *decisive* if, had we looked just at the votes cast by the members in M', the committee's decision on *every* issue would have been the same. (In other words, the overall outcome of the voting among the members in M' is the same on every issue as the overall outcome of the voting by the entire committee.) Such a subset can be viewed as an "inner circle" that reflects the behavior of the committee as a whole.

Here's the question: Given the votes cast by each member on each issue, and given a parameter k, we want to know whether there is a decisive subset consisting of at most k members. We'll call this an instance of the *Decisive Subset Problem*.

Example. Suppose we have four committee members and three issues. We're looking for a decisive set of size at most $k = 2$, and the voting went as follows.

Issue #	m_1	m_2	m_3	m_4
Issue 1	Yes	Yes	Abstain	No
Issue 2	Abstain	No	No	Abstain
Issue 3	Yes	Abstain	Yes	Yes

Then the answer to this instance is "Yes", since members m_1 and m_3 constitute a decisive subset.

Prove that Decisive Subset is NP-complete. Reduce from Vertex Cover.

To show that this problem is in NP, we can consider the following. Given a subset M' of size $\leq k$, we can check if M' is decisive by:

- For each issue, compute the votes of the members in M'
- Check if the overall committee decision matches the decision of M' on all issues
- There are t issues and n members, so this verification involves going through $O(t \cdot n)$ data, which is polytime.

Thus, the problem is in NP.

We can reduce the problem from Vertex Cover, where a graph $G = (V, E)$ and a parameter k determine whether there exists a subset $V' \subseteq V$ of size $\leq k$ such that every edge in E has at least one endpoint in V'.

Consider the input $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, along with some parameter k. We can make the voting dataset like:

- Each vertex $v \in V$ corresponds to a committee member m_v
- Each edge $e = (u, v) \in E$ corresponds to a voting issue
- For each edge $e = (u, v)$:
  - m_u votes "Yes"
  - m_v votes "Yes"
  - All other members vote "No"
- The committee decides "Yes" on an issue if at least one member from M' votes "Yes" for each edge e.

A subset M' of committee members is decisive if and only if every edge $e \in E$ is covered by at least one member of M'. This is basically the same thing as saying M' is a vertex cover of G.

If G has a vertex cover of size $\leq k$, then there exists a decisive subset M' of size $\leq k$ in the voting dataset.

In other words, if the Decisive Subset Problem has a solution M' of size $\leq k$, then M' corresponds to a vertex cover of size $\leq k$ in G. Thus it is NP-complete.

2. (10 points)

a. If you've found a problem to be NP-complete but would still like to pursue an efficient solution what is an approach you can take?

There are a few methodologies to follow when a problem is NP-complete to find a efficient solution.

- Approximation algorithms create solutions close to the optimal answer. They sacrifice exactness for efficiency, useful for this specific context. They aim to get within a specific factor (AKA the approximation ratio) of the optimal solution, being a reliable approach for an efficient solution.
- Heuristics are problem solving strategies that help find good enough solutions quickly without a guarantee of optimality or a provable bound on quality. We also see fast results as compared to approximation algorithms, being useful for large sets of data.

b. What is the P = NP problem?

The basic question being asked is whether each and every single problem that can be verified in polynomial time also solvable in polynomial time. If P = NP, it would mean that all problems for which solutions can be verified efficiently can also be solved efficiently. As of now, there is no proof for P = NP, with most arguing $P \neq NP$.

3. (10 points)

The optimization version of the vertex cover problem is to find the smallest set of vertices in a graph that covers all edges. Consider this problem on a bipartite graph G with nodes $x_1, x_2, ..., x_k$ and $y_1, y_2, ..., y_l$. Here $k < l$. There is an edge from every node of the form x_i to every node of the form y_j. Consider solving this problem using local search. How many local minima are there in the energy landscape of this problem? What is the solution at the global minimum?

We can define a local minima in the energy landscape to be some subset S such that removing any vertex from S results in S no longer being a valid vertex cover.

Within G:

- To cover all the edges, at least one side of the bipartite graph (either x's or y's) must be included in the vertex cover
- Any subset of size k from $\{x_1, \ldots, x_k\}$ or size l from $\{y_1, \ldots, y_l\}$, that covers all edges could represent a local minimum

There are 2 possible ways to get a vertex cover:

- Include all k vertices from $\{x_1, \ldots, x_k\}$
- Include all l vertices from $\{y_1, \ldots, y_l\}$

Thus, there are 2 local minima in the energy landscape.

The solution at the global minimum would come from the smallest vertex cover. Given that $k < l$, the size of the global minimum vertex cover is k. The solution would thus be to include all k vertices from $\{x_1, \ldots, x_k\}$.

4. (20 points)

We define the *Escape Problem* as follows. We are given a directed graph $G = (V, E)$ (picture a network of roads). A certain collection of nodes $X \subset V$ are designated as populated nodes, and a certain other collection $S \subset V$ are designated as safe nodes. (Assume that X and S have no nodes in common.) In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. A set of evacuation routes is defined as a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.

Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.

We can rewrite this problem in terms of a flow network. The populated nodes X can be sources where people start. The safe nodes S can be sinks where people must arrive. The goal would be to find a flow that routes one unit from each node in X to some node in S such that each node in X sends exactly one unit of flow, flow is conserved, and the total flow to S is $|X|$.

We can make the flow network as follows:

- Use all the nodes in V as they are in graph G as nodes for the flow network
- Use the directed edges of graph G for edges in the flow network, where each edge in E gets a capacity of 1
- Add a source node s and connect it to all the nodes in X with edges of capacity 1
- Add a sink node t and connect it to all the nodes in S with edges of capacity 1

We can solve this problem using Edmond Karp, finding a maximum flow from s to t. If the maximum flow equals $|X|$, then it is possible to construct an evacuation route. Otherwise, it is not possible.

Constructing the flow network involves adding $|X|+|S|+|E|$ edges, which is polytime in the size of G. Edmond Karp is also known to run in polytime ($O(VE^2)$). Thus, we can decide in polynomial time whether such a set of evacuation routes exists.

5. (20 points)

You opened a restaurant and you're trying to save money on ingredients. There is a set of n possible raw ingredients you could buy, $I_1, I_2, ..., I_n$ (e.g., bundles of greens, jugs of rice, and so forth). Ingredient I_j must be purchased in units of size $s(j)$ grams (any purchase must be for a whole number of units), and it costs $c(j)$ dollars per unit. Also, it remains safe to use for $t(j)$ days from the date of purchase.

Over k consecutive days, you want to make a set of k different daily specials, one each day. The order in which you schedule the specials is up to you. The i^{th} daily special uses a subset $S_i \subseteq \{I_1, I_2, ..., I_n\}$ of the raw ingredients. Specifically, it requires $a(i, j)$ grams of ingredient I_j. You want to keep your meals fresh, so there's a final constraint: for each daily special, the ingredients S_i are partitioned into two subsets: those that must be purchased on the very day when the daily special is being offered, and those that can be used any day while they're still safe.

For example, the mesclun-basil salad special needs to be made with basil that has been purchased that day; but the arugula-basil pesto special can use basil that is several days old, as long as it is still safe.

You can save money as follows. When you buy a unit of a certain ingredient I_j, you may not need the whole thing for the special you're making that day. Thus, if you can follow up quickly with another special that uses I_j but doesn't require it to be fresh that day, then you can save money by not having to purchase I_j again. Of course, scheduling the basil recipes close together may make it harder to schedule other recipes close together – this is where the complexity comes in.

So we define the *Daily Special Scheduling Problem* as follows: Given data on ingredients and recipes as above, and a budget x, is there a way to schedule the k daily specials so that the total money spent on ingredients over the course of k days is at most x?

Prove that *Daily Special Scheduling* in NP-complete. Reduce from Hamiltonian Cycle.

To show that this problem is in NP, we can consider the following. Given a schedule of k daily specials, we can verify in polytime whether:

- The total cost of the ingredients is within budget x, found by checking the cost, which requires summing up the ingredient costs for each day - $O(kn)$
- The perishable ingredients are used within their safe limit $t(j)$, found by ensuring ingredient usage adheres to the time constant - $O(kn)$

Thus, the problem is in NP as the verification is polytime.

We can reduce the problem from the Hamiltonian Cycle, where a graph $G = (V, E)$ and we have to check for a cyrcle that visits every vertex exactly once.

We can map each vertex $v \in V$ to correspond to a daily special, with $|V|$ specials total. Additionally, each edge $(u, v) \in E$ will correspond to an ingredient that connects two consecutive specials. If we schedule special u followed by special v, we require the ingredient for edge (u, v).

Each edge (u, v) in the graph corresponds to ingredient I_{uv}:

- Has a cost $c(uv) = 1$
- Has a safe usage time $t(uv) = 1$ day, meaning it must be used on the same day it is purchased

The budget $x = |V|$ ensures that exactly $|V|$ ingredients can be used, which forces the solution to form a valid cycle. Additionally, each ingredient I_{uv} can only be used for a valid consecutive transition between specials u and v. The schedule of daily specials must use exactly $|V|$ ingredients, corresponding to a cycle that visits all vertices.

If a Hamiltonian Cycle exists in G, we can construct a schedule of daily specials where the sequence of specials corresponds to the Hamiltonian Cycle, each transition uses an ingredient that corresponds to an edge in the cycle, and the total cost is $|V|$, which is within the budget.

If the Daily Scheduling Problem has a solution, the created sequence corresponds to a Hamiltonian Cycle in G. This is because the budget $x = |V|$ ensures that exactly $|V|$ ingredients are used and the time constraint $t(uv) = 1$ ensures that the transitions form a valid cycle with no repeated vertices.

Constructing the specials, ingredients and constraints from G takes polytime. Verifying the solution, as mentioned earlier, also takes polytime.

Thus it is NP-complete.

6. (20 points)

a. In a standard s-t Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of the Maximum-Flow problem with node capacities.

Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative node capacities $c_v \geq 0$ for each $v \in V$. Given a flow f in this graph, the flow though a node v is defined as $f_{in}(v)$. We say that a flow is feasible if it satisfies the usual flow-conservation constraints and the node-capacity constraints: $f_{in}(v) \leq c_v$ for all nodes.

Give a polynomial-time algorithm to find an s-t maximum flow in such a node-capacitated network.

Hint: Consider splitting each node into two (don't split the source and the sink).

Replace each node v (except for s and t) with two nodes, such that v_{in} represents incoming flow to v and v_{out} represents outgoing flow to v. Add an edge (v_{in}, v_{out}) with capacity c_v to enforce the capacity on the vertex v. For every incoming edge (u, v) in the original graph, replace it with an edge (u, v_{in}) and for every outgoing edge (v, w), replace it with an edge (v_{out}, w).

On the transformed graph, we can use Edmond Karp to find the maximum flow for the $s - t$ flow network. The resulting flow repurposes the vertex capacities and flow into a typical edge capacity based flow network.

Transforming the graph involves splitting each node, besides s and t which adds $O(|V|)$ nodes and $O(|E|)$ edges. Running the max-flow algorithm is is $O(VE^2)$ which is still polynomial.

b. Consider again the *Escape Problem* in question 4. Suppose we want to enforce an even stronger version of the "no congestion" condition (iii). Thus we change (iii) to say "the paths do not share any nodes". With this new condition, show how to decide in polynomial time whether a set of evacuation routes exists.

For every node $v \in V - (X \cup S)$, we can split v into two nodes (v_{in} and v_{out}). We can add an edge (v_{in}, v_{out}) with capacity 1 to ensure that only one path can pass through v. Then, we can replace each original edge $(u, v) \in E$ with an edge (u_{out}, v_{in}).

Add a source node s and connect it to all the nodes in X. Add a sink node t and connect it to all the nodes in S. With this, run a max-flow algorithm from s to t. The max flow value will represent the number of path from X to S that are node-disjoint. If the max flow equals $|X|$, then there exists a set of $|X|$ paths that

work, and no such paths otherwise.

Splitting the nodes doubles the number of nodes and edges, which is $O(V + E)$. Running max-flow would be $O(VE^2)$. Thus, it is in polytime.