CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

•

Homework 10

Due on Tuesday, April 25th, 2017 at 11:59am

•

Solutions by

Michael Fan

26697596

In collaboration with

Rosa Choe

Problem 1: Binary Gambler

★★ **Level**

After the CS170 project, you go to Las Vegas to celebrate. You buy k tokens in the casino, and you are playing the following game until you run out of tokens:

1. At each turn the house chooses a color, GREEN or RED, and you try to guess the color.
2. If you guess wrong, you lose 1 token.
3. If you guess correctly, you keep your token, and the casino donates one dollar to your charity that you founded.

You have 1024 friends watching. On each turn, they all tell you what they think the next color will be. One of your friends can secretly see the future, and always guesses correctly, although you don't know which of your friends it is.

How many tokens do you need in order to guarantee an infinite support for your charity? In other words, what should k be so that you never run out of tokens?

Solution I need 11 tokens in order to guarantee an infinite support for my charity. Every time I need to guess the color to a house, I ask all of my friends for their guesses and go with the majority guess. In the case that I guess right, I lose no tokens. In the case that I guess wrong, I can eliminate at least half of the people remaining (because the majority is wrong). Therefore, after $\log_2 1024$ wrong guesses, only the friend who is always correct will be left. Therefore, I need 11 tokens to make sure that I have 1 token remaining when I narrow down to the friend who is always correct.

Problem 2: Ain't Nobody Fresher Than My

★★★ Level

Kanye West has gained access to the Facebook social graph. For years now, he has boasted of having the largest, most impressive clique in the network, and now he wants to confirm this.

As Kanye's social media advisor, you inform him that sadly, as the CLIQUE problem is **NP**-complete, there is little hope of being able to do this. "Fine," he says, "can we just find a clique of at least $\frac{|V|}{4}$ people?"

Note: for the CLIQUE problem, you are given a graph G, and you want to find a clique of at least k vertices, or return that no such clique exists.

a. Prove that in fact, the QUARTER-CLIQUE problem (finding a clique of size at least $\frac{|V|}{4}$) is also **NP**-complete.

Solution We will show a reduction from CLIQUE to QUARTER-CLIQUE. This will imply that QUARTER-CLIQUE is at least as hard as CLIQUE, and thus prove that QUARTER-CLIQUE is also NP-Complete. To show this reduction, we need to find a polynomial time algorithm A which will transform an input CLIQUE to an input to QUARTER-CLIQUE and a polynomial time algorithm B which will transform a solution to QUARTER-CLIQUE to a solution of CLIQUE.
A is as follows: given an input to CLIQUE, $(G(V, E), k)$, we will add unconnected dummy vertices to the graph until there are $4k$ vertices in the graph total. If there are more than $4k$ vertices in the graph to start with, then do not add any vertices. This is a polynomial time function as unconnected vertices can be added in constant time.
B is as follows: if there is a solution to QUARTER-CLIQUE, then we return that solution as our output for CLIQUE. If there is no solution, then there is no solution for CLIQUE. This is because upon finding a solution for QUARTER-CLIQUE, we have found a clique of size at least $4k/4 = k$ as there are at least $4k$ vertices in the graph input to QUARTER-CLIQUE. None of the dummy vertices will ever be used because they are exclusively their own cliques of size 1 (in the case that someone queried for a clique of size 1 from the original graph, we would simply return any of the original vertices). If there is no solution, then there is no clique of size at least k in the original graph as none of the dummy nodes are actually considered in the solution.

b. Kanye is going to give an incorrect reduction. Give an instance of CLIQUE for which Kanye's reduction gives the wrong answer, and explain why. Kanye is interested in another problem: "In my industry, musicians don't like to talk to one another. But we each talk to our various agents. Likewise, each agent may talk to many artists, but not to one another. Can I find k musicians who all talk to k agents, who in turn all talk to those same k artists?"

Formally stated, we have the CLIQUE WITH AGENTS problem:

> Given a bipartite graph $G = (M \cup A, E)$ and a number k, find subsets $M' \subseteq M$ and $A' \subseteq A$, $|M'| = |A'| = k$, such that $(m, a) \in E \ \forall m \in M', a \in A'$. We call $M' \cup A'$ a *biclique* of size $2k$.

Kanye believes CLIQUE WITH AGENTS is **NP**-hard. He proposes the following reduction:

> "Reduce from CLIQUE. Given a graph $G = (V, E)$ and a desired clique size k,
>
> (i) Create a new bipartite graph $G' = (M, A, E')$, initially empty.
>
> (ii) For each $v \in V$, add a musician m_v to M and an agent a_v to A. Add an edge (m_v, a_v) to E'.
>
> (iii) For each $(u, v) \in E$, add edges (a_u, m_v) and (a_v, m_u) to E'.
>
> Then, if there is a size-k clique in G, there will be a size-$2k$ biclique in G'. Conversely, if there is no k-clique in G, all bicliques in G' are smaller than $2k$."

Solution Consider an input to the CLIQUE with graph G being a complete bipartite graph, $K_{3,3}$. We create the input graph, G', to QUARTER-CLIQUE per Kanye's instructions from G. We see that in G, there are no cliques of size 3. According to Kanye, there should be no bicliques of size $2k$ in G' where

Solution (cont.)

$k = 3$. However, there are multiple bicliques of this size. Contradiction.

Problem 3: Multiway Cut

★★★ Level

In the MULTIWAY CUT problem, the input is an undirected graph $G = (V, E)$ and a set of terminal nodes $s_1, s_2, \ldots, s_k \in V$. The goal is to find the minimum set of edges in E whose removal leaves all terminals in different components.

a. Show that this problem can be solved in polynomial time when $k = 2$.

Solution Run the max flow algorithm with all flows set to 1 for each edge and with the 2 terminal nodes set to be s and t. The min-cut identified in the algorithm will be the minimum set of edges that can be removed to separate s and t. The proof of correctness is the same as that of the max-flow algorithm as the min-cut is identified as part of the midterm.

b. Give an approximation with ratio at most 2 for the case $k = 3$. (Hint: use part a, your algorithm shouldn't be too complicated)

Solution Given 3 vertices A, B, C, choose any 2 arbitrary vertices A, B and run the above algorithm with A and B as the vertices to separate. Now, run the above algorithm again with A and C as the 2 vertices to separate on the subgraph that they are contained in. The set of edges that we remove will be the set of distinct edges cut in the first run E_1 combined with the edges cut in the second run E_2. We will now prove that this is at most 2 times worse than the optimal solution. Suppose that the optimal solution is composed of a set of edges E_0. Then, $|E_0| \geq |E_1|$ and $|E_0| \geq |E_2|$ as it will take at least as many edges to separate 3 vertices than 2 vertices.

$$2|E_0| \geq |E_1| + |E_2|$$

However, E_1 and E_2 may contain overlapped edges.

$$|E_1 + E_2| \geq |E_1|$$

Thus,

$$2|E_0| \geq |E_1 + E_2|$$

and we have show that the approximation has ratio at most 2.

Problem 4: TSP via Local Merging

★★★★★ Level

The METRIC TRAVELING SALESMAN problem is the special case of TSP in which the edge lengths form a *metric*, which satisfies the triangle inequality and other properties (review section 9.2.2). In class, we saw a 2-approximation algorithm for this problem, which works by building a minimum spanning tree.

Here is another heuristic for metric TSP:

Summary: maintain a current *subtour* τ on a subset of V, then expand it to include all nodes in G

function TSP-LOCAL-MERGE(complete undirected graph $G = (V, E, \ell)$ with metric distance)
    $\tau \leftarrow [v_1, v_2]$ where v_1 and v_2 are the closest pair of nodes in G
    while $|\tau| < |V|$ **do**
        $v_i, v_j \leftarrow v_i, v_j$ such that $v_i \in \tau, v_j \in \{V \setminus \tau\}, (v_i, v_j) \in E$, and $l(v_i, v_j)$ is minimized
        Modify τ by inserting v_j immediately after v_i

Prove that the above "local merging" algorithm also approximates metric TSP to a factor of 2.

Hint: Note the similarity between this algorithm and Prim's.

Clarification: τ is the list of cities to visit, in order.

Solution This algorithm begins by adding the vertices from the lightest edge in the tree to a accumulating set, and continues by continuously picking the lightest edge from a vertex in the growing vertex cover to a vertex outside of it. Note that the edges that are picked by this algorithm are exactly that which Prim's Algorithm would pick for the graph. Consider the growing vertex set $\tau_t = (a, b, c)$ at iteration t of the algorithm. If the next vertex that is added d is appended to the end of the list, then $weight(\tau_{t+1}) = weight(\tau_t) + l(c, d)$ and the tour weight is exactly that of the partial MST which is upper bounds the metric TSP to a factor of 2. If the next vertex is added somewhere in the middle of the list, for example, $\tau_t = (a, b, d, c)$, then $weight(\tau_{t+1}) = weight(\tau_t) - l(b, c) + l(b, d) + l(d, c)$. By the triangle inequality, $l(d, c) \leq l(b, d) + l(b, c)$, $l(b, d) \geq l(d, c) - l(b, c)$. Thus, $weight(\tau_{t+1}) \leq weight(\tau_t) + 2 * l(b, d)$. So, for every edge along the MST, we add on at most 2 times the weight of the edge. Thus, the final tour that we construct is bounded by 2 times the MST weight which upper bounds 2 times the TSP path.

Problem 5: Finding Zero(s)

★★★★★ Level

This homework has been shortened to allow you to work on the programming project. Your finished project code, writeup, and output file submissions will be due at the same time as the next homework!

Explain on the high level how your phase II algorithm works, or how you plan for it to work. Give some estimates and justifications for how long it will take to solve 300 instances.

Note: You may change your project ideas after submitting this homework, but this question is meant to encourage you to make progress on the project.

Solution We will build a series of adjacency sets to model the constraints and then stochastically sample the list of items and greedily choose the best n items that satisfy all class constraints while staying under the cost and weight constraints. We will also try to make an ILP algorithm with stochastically chosen items and take the max of the output of this and the greedily chosen solution. The bulk of the time of solving an instance for a greedy algorithm will be building up the constraint graph. To build a max of $200,000$ constraints, I estimate that this will take a degree more time than it took generate the constraints, so around 100 seconds. To get an estimation via a greedy approach, it will take at most as much time as building up the constraints, so around 200 seconds in total. The ILP solver will take around 5 times as long as the greedy solution, so around 700 seconds in total. To solve 300 instances, it will take around 210000 seconds in total which is around 60 hours.