CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

●

Homework 10

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

●

Solutions by

Ninh DO

25949105

In collaboration with

None

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
Main idea: We alway choose the majority guess each turn (at least a half). After each turn, we eliminate the incorrect guesser.
If the turn is a correct guess, we eliminate some guessers, but at least 0 guesser. If the turn is an incorrect guess, we eliminate at least half guessers. So after $\log_2 1024 = 10$ incorrect guesses, by binary elimination we have at most two guessers left. Then we need $k = 11$.
Pseudocode:

```
G = 1024
while still playing:
    g = # of GREEN guessers / G
    if g >= 1/2:
        choose GREEN
    else:
        choose RED
    G = G eliminating incorrect guessers
```

Proof of correctness:
See main idea above.
Runtime: $O(\log n)$ where n is the initial number of guessers.

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

NP: If we have a solution, that is given a subgraph G' of V' vertices. We can verify if $|V'| \geq \frac{|V|}{4}$ in $O(1)$ time, and we can verify if G' is a full graph with all mutually-connected vertices in $O(|V|^2)$ time. Thus the problem is NP.

NP-hard: We try to reduce the CLIQUE problem to the QUARTER-CLIQUE problem. There are three scenarios:

1 if $k = \frac{|V|}{4}$, we do nothing.

2 if $k > \frac{|V|}{4}$, we add m more vertices do that $k = \frac{|V|+m}{4}$, thus $m = 4k - |V|$. These new vertices are not connected. Given the graph G with $|V|$ vertices and an integer k, the CLIQUE problem of the graph G has a solution iff the QUARTER-CLIQUE problem of the new graph G' with $|V| + m$ vertices has a solution. In fact, if the CLIQUE problem has a solution of k-vertex clique, this clique does not contain m additional vertices since these vertices are not connected, thus we can find $\frac{|V|}{4}$-vertex clique from this k-vertex clique to make the solution to the QUARTER-CLIQUE problem. On the other hand, if the QUARTER-CLIQUE problem has the solution of $\frac{|V|+m}{4} = k$-vertex clique, then this clique does not contain m additional vertices for the disjoint reason, and this clique is also a solution to the CLIQUE problem.

3 if $k < \frac{|V|}{4}$, we add m more vertices do that $k + m = \frac{|V|+m}{4}$, thus $m = \frac{|V|}{3} - \frac{k}{3}$. These new vertices are made to be mutually connected and each additional vertex is connected to all vertices in G. Given the graph G with $|V|$ vertices and an integer k, the CLIQUE problem of the graph G has a solution iff the QUARTER-CLIQUE problem of the new graph G' with $|V| + m$ vertices has a solution. In fact, if the CLIQUE problem of G has a solution of k-vertex clique, thus we can find $\frac{|V|}{4} + m$-vertex clique for the graph G' to make the solution to the QUARTER-CLIQUE problem. On the other hand, if the QUARTER-CLIQUE problem has the solution of $\frac{|V|+m}{4} = k + m$-vertex clique, then eliminating m vertices from this clique gives the solution of k-vertex clique to the CLIQUE problem.

In case of no solution, we modify the graph G in the same manner as the above scenarios:

1 if $k = \frac{|V|}{4}$, both the problem are the same.

2 if $k > \frac{|V|}{4}$, we add $m = 4k - |V|$ discrete vertices to G. Given the graph G with $|V|$ vertices and an integer k, the CLIQUE problem of the graph G has NO solution iff the QUARTER-CLIQUE problem of the new graph G' with $|V| + m$ vertices has NO solution. In fact, if the QUARTER-CLIQUE problem has NO solution and assume the CLIQUE problem has a solution of k-vertex clique, this clique does not contain m additional vertices since these vertices are not connected, thus we can find $\frac{|V|}{4}$-vertex clique from this k-vertex clique to make the solution to the QUARTER-CLIQUE problem, it contradicts the fact that the QUARTER-CLIQUE problem has NO solution. On the other hand, if the CLIQUE problem has NO solution and assume the QUARTER-CLIQUE problem has the solution of $\frac{|V|+m}{4} = k$-vertex clique, then this clique does

Solution (cont.)

not contain m additional vertices for the disjoint reason, and this clique is also a solution to the CLIQUE problem, it contradicts the fact that the CLIQUE problem has NO solution.

3 if $k < \frac{|V|}{4}$, we add $m = \frac{|V|}{3} - \frac{k}{3}$ mutually connected vertices to G and make each new vertex connect to all vertices in G. Given the graph G with $|V|$ vertices and an integer k, the CLIQUE problem of the graph G has NO solution iff the QUARTER-CLIQUE problem of the new graph G' with $|V| + m$ vertices has NO solution. In fact, if the QUARTER-CLIQUE problem has NO solution and the CLIQUE problem of G has a solution of k-vertex clique, thus we can find $\frac{|V|}{4} + m$-vertex clique for the graph G' to make the solution to the QUARTER-CLIQUE problem, it contradicts the fact that the QUARTER-CLIQUE problem has NO solution. On the other hand, if the CLIQUE problem has NO solution and the QUARTER-CLIQUE problem has the solution of $\frac{|V|+m}{4} = k + m$-vertex clique, then eliminating m vertices from this clique gives the solution of k-vertex clique to the CLIQUE problem, it contradicts the fact that the CLIQUE problem has NO 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

Implementing the algorithm above for a complete bipartite graph $G = (L, R, E)$ with $|L| = |R| = k$. We come up with the graph G' with a biclique (L, A_R) of size $2k$, where A_R is the set of agents for nodes in R. But we do not have a clique of size k in G. 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
For $k = 2$, we have two nodes s_1, s_2. Finding a min-cut (s_1, s_2) can be run in polynomial time using maximum flow algorithm.

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
Main idea:
For $k = 3$, we have there nodes s_1, s_2, s_3. First we run the max flow algorithm to find a min-cut for (s_1, s_2), we call this one E_1. If s_3 is in the s_1 side of the min-cut, we find the min-cut for (s_1, s_3), called E_2. Otherwise, we find find the min-cut for (s_2, s_3). The solution is the union of E_1 and E_2.
Pseudocode:

```
E_1 = min-cut(s1, s2}
if s3 in the side of s1:
        E_2 = min-cut(s1, s3}
else:
        E_2 = min-cut(s2, s3}
return E_1 + E_2
```

Proof of correctness:
If we have an optimal multiway E_o, then $|E_1| < |E_o|$ and $|E_2| < |E_o$ since E_o includes the min-cuts of (s_1, s_2) and (s_1, s_3) (or (s_2, s_3)). Thus, $|E_1 + E_2| \leq |E_1| + |E_2| \leq 2|E_o|$
Runtime:
Polynomial since we run row max-flow algorithm.

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:

```
1: Summary: maintain a current subtour $\tau$ on a subset of $V$, then expand it to include all nodes in $G$

2: function TSP-LOCAL-MERGE(complete undirected graph $G = (V, E, \ell)$ with metric distance)
3:     $\tau \leftarrow [v_1, v_2]$ where $v_1$ and $v_2$ are the closest pair of nodes in $G$
4:     while $|\tau| < |V|$ do
5:         $v_i, v_j \leftarrow v_i, v_j$ 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
6:         Modify $\tau$ 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 is similar to the Prim's algorithm, so the first edge (v_1, v_2) is the lightest edge which is a part of MST. Based on the above algorithm, when we add a vertex v_j we have:

$weight(\tau_{new}) = weight(\tau_{old}) + l(v_i, v_j) + l(v_j, v_k) - l(v_i, v_k)$

According to the triangle inequality: $l(v_j, v_k) - l(v_i, v_k) < l(v_i, v_j)$, thus:

$weight(\tau_{new}) \leq weight(\tau_{old}) + 2l(v_i, v_j)$

It does mean whenever we add a new edge, we never exceed twice its length. Since any TSP is at least as long as an MST, so:

$weight(\tau_{end}) \leq 2 * weight(MST) \leq 2 * TSP_{opt}$

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

Main idea:

First I process the list of constraints by using a hash table and put all items in to this table. For each key item, I make a linked list of the items that conflicts this key.

Next, I loop through each item and add it to the basket if it does not conflict with any items in the basket. If it does, I calculate the new effective value, defined as the ratio of return value and weight, of adding new item and eliminating the conflict items. If the new effective value is larger, I decide to add the new item. Otherwise, I move to the next item.

Definitely before adding any item or change the shopping basket, we have to check for the total price and the total weight so far to make sure our basket does not exceed P and W, which are the price and weight limits, respectively.

The order of looping is going through most conflicting item to least conflicting item to minimize the number of eliminating item. Thus, be have to sort items based on their number of conflicting items in the hash table.

In each loop, if I eliminate the conflicling items, I have an inner loop to go through the conflicting items of the elimiated conflicting items to see if I can add any more item that may help inclease the total return value.

Runtime:

Reading file take $O(N+C)$, where N and C are the number of items and constraints, respectively. We assume that the average number of conflicting items in each constraint line does not depend on any input parameter. It may be not a very valid assumption but it is acceptable in this stage since we pay more attention on achieving the goal of the problem than dealing with different types of input.

Sorting items in the order of the decreasing number of conflicting item takes $O(N \log N)$ time.

Looping through items takes $O(N^2)$ time.

The total runtime is $O(N^2)$.

To solve 300 instances, it takes around 6 hours in average based on the expolation of the runtime of running 20 instances (both provided and generated by my coding).

The project is still in progress, we expect to change the algorithm in different ways. Thus, the run time may change significantly.