CS 170
Spring 2017

Efficient Algorithms and Intractable Problems
Prasad Raghavendra and Sanjam Garg

Homework 10

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

- *Shortcut questions*: Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Tuesday, April 25, at 11:59am

1. (★★ level) Binary gambler

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: $k = 11$.

Main idea:

1. We know that there is one expert that is *never* wrong, so at each turn we throw out all the experts that guessed incorrectly.
2. Among the remaining experts, we take the majority opinion.

Pseudocode:

procedure BINARYGAMBLER
  $S \leftarrow [n]$. ▷ Initialization:

  for each turn **do** ▷ Main loop:
    if (# experts in S that guess GREEN) $> |S|/2$ **then**
      Guess GREEN.
    else
      Guess RED.
    $S \leftarrow S \setminus$ (experts that guessed incorrectly).

Analysis: Every time we guess incorrectly, at least half of the experts in S were incorrect. Therefore, after we make $\log_2 1024 = 10$, S contains only one expert - who is always right.

2. (★★★ level) Ain't Nobody Fresher than My

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: QUARTER-CLIQUE is in **NP**: if someone claims that $H \subset G$ is a semi-clique, we simply check that it has $|V|/4$ nodes and that all edges are present. This takes $O\left(|V|^2\right)$ time.

To show that QUARTER-CLIQUE is **NP**-hard, we reduce from CLIQUE. Given a graph G and desired clique size k, we first let $G' = G$, and add to G' the following:

a) if $k = \frac{|V|}{4}$, then add nothing to G'

b) if $k > \frac{|V|}{4}$, then we want to add q vertices $x_1 ... x_q$ to G' such that $k = \frac{|V|+q}{4}$. Solving for q, we get $q = 4(k - \frac{|V|}{4})$. We do not add any edges.

c) if $k < \frac{|V|}{4}$, then we want to add q vertices $x_1 ... x_q$ to G' such that $\frac{|V|+q}{4} = k + q$. Solving for q, we get $q = \frac{4}{3}(\frac{|V|}{4} - k)$. In addition, we add an edge between every pair of added vertices (q_i, q_j). Finally, we add an edge between every q_i and every $v \in V$.

We then run QUARTER-CLIQUE on G'. If a clique S is found, we remove from S any added vertices, then we return S. Otherwise if no valid clique is found, we return no solution.

Note that for b), if $k < \frac{|V|}{4}$, x is negative and thus invalid. Similarly, for c), if $k > \frac{|V|}{4}$, x is again negative and thus invalid.

Proof of reduction: First, we prove the direction

"If we find a solution S to QUARTER-CLIQUE(G'), then S minus added vertices is a solution to CLIQUE(G,k)"

a) if $k = \frac{|V|}{4}$, we let $G' = G$. If QUARTER-CLIQUE finds a clique S, we know that $|S| = \frac{|V|}{4} = k$. Since no vertices are added, we return S, which is a valid size-k clique.

b) if $k > \frac{|V|}{4}$, we add dummy vertices $x_1 ... x_q$ for $q = 4(k - \frac{|V|}{4})$. This means that if QUARTER-CLIQUE finds a clique, it has size $\frac{|V'|}{4} = k$. Furthermore, S is guaranteed to not contain any of the added vertices since $x_1 ... x_q$ have no incident edges and cannot be part of any clique. Thus S is a valid size-k clique of G.

c) if $k < \frac{|V|}{4}$, we add dummy vertices $x_1 ... x_q$ for $q = \frac{4}{3}(\frac{|V|}{4} - k)$. Suppose QUARTER-CLIQUE finds a clique S of size $\frac{|V'|}{4}$. S can contain at most q added vertices. After removing the added vertices, we get at least $\frac{|V'|}{4} - q = \frac{|V|+q}{4} - q = \frac{|V|}{4} - \frac{3}{4}q = k$ vertices remaining in S. Thus S minus any added vertices gives a clique of size at least k in G.

Next, we prove the direction "if there is no solution S to QUARTER-CLIQUE, then there is no k-clique in G". It is a little easier to prove the contrapositive instead:

"if there is a valid k-clique in G (call it T), then QUARTER-CLIQUE will find a clique S"

a) if $k = \frac{|V|}{4}$, then T is a clique in G' with size $\frac{|V'|}{4}$

b) if $k > \frac{|V|}{4}$, then again T is a clique in G' with size $\frac{|V'|}{4}$

c) if $k < \frac{|V|}{4}$, $T \cup \{x_1 ... x_q\}$ is a clique in G' with size $k + q = \frac{\|V'\|}{4}$ (simple algebra)

Note that $O\left(|V|^2\right)$ is an upper bound on the time taken by the reduction.

(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 a complete bipartite graph $G = (L,R,E)$, $|L| = |R| = k$, which itself is a biclique. After the reduction, G' will have a biclique (L, A_R), where A_R is the set of agents for nodes in R. This biclique has size $2k$, even though there was clearly no clique of size k in G.

3. (★★★ level) Multiway Cut

In the MULTIWAY CUT problem, the input is an undirected graph $G = (V,E)$ and a set of terminal nodes $s_1, s_2, ..., s_k \in V$. 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 $k = 2$.

(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:

(a) This is the same problem as finding a (s_1, s_2) min-cut, which can be done by a maximum flow computation in polynomial time.

(b) Find an (s_1, s_2) mincut $E_1 \subseteq E$ using maximum flow. Assume that s_3 falls on the s_1's side of the cut (the other side is symmetric). Compute then a (s_1, s_3) mincut $E_2 \subseteq E$, and output $E_1 \cup E_2$. To see this is a 2-approximation, consider the optimal multiway cut E^*: because E^* is both a (s_1, s_2) cut and a (s_1, s_3) cut, we have $|E_1| \leq E^*$ and $|E_2| \leq E^*$. Hence, $|E_1 \cup E_2| \leq |E_1| + |E_2| \leq 2|E^*|$, as required. Our algorithm simply runs the max-flow/min-cut algorithm twice; thus, our algorithm has the same time complexity.

4. (★★★★★ level) TSP via Local Merging

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:

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: Observe that this algorithm picks nodes in precisely the same way as Prim's algorithm.

First, observe that the lightest edge (v_1, v_2) in G is a valid Prim's MST edge. Our subtour starts with two copies of this edge. Thenceforth, each edge (v_i, v_j) is also an edge that would appear in the MST selected by

1: Summary: maintain a current *subtour* τ 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: $\quad\tau \leftarrow [v_1, v_2]$ where v_1 and v_2 are the closest pair of nodes in G
4: $\quad$**while** $|\tau| < |V|$ **do**
5: $\quad\quad v_i, v_j \leftarrow v_i, v_j$ 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: $\quad\quad$Modify τ by inserting v_j immediately after v_i

Prim's. If a v_j is added to the end of τ, then that means we have used the cheapest possible edge to include v_j in τ, which is optimal. So suppose this is not the case. Suppose that v_j was inserted in the middle of τ. Then we will have swapped out an adjacent $v_i, v_k \in \tau$, for an adjacent $v_i, v_j, v_k \in \tau$. So, on iteration t of the while-loop, we have

$$\text{cost}(\tau_{t+1}) = \text{cost}(\tau_t) - \ell(v_i, v_k) + \ell(v_i, v_j) + \ell(v_j, v_k)$$

By the triangle inequality, $\ell(v_j, v_k) \leq \ell(v_i, v_j) + \ell(v_i, v_k)$, or $\ell(v_j, v_k) - \ell(v_i, v_k) \leq \ell(v_i, v_j)$. Thus

$$\text{cost}(\tau_{t+1}) \leq \text{cost}(\tau_t) + 2 \cdot \ell(v_i, v_j)$$

so for every edge weight that Prim's would incur, we incur at most twice as much. Since any TSP tour is at least as costly as an MST (review section 9.2.3 if this is unclear), we conclude that

$$\text{cost}(\tau_{|V|}) \leq 2 \cdot \text{cost}(\text{MST}) \leq 2 \cdot \text{OPT}$$

Finally, because the algorithm can implemented a la Prim's, its running time is polynomial, making it a valid approximation algorithm. □

5. (★★★★★ level) Project Progress Report

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: There are many possibilities! Keep going!