CS 170
Fall 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 Wednesday, November 29, at 4:59pm

0. Who did you work with?

List all your collaborators on this homework. If you have no collaborators, please list "none".

1. (★★★ level) Circular Reductions

Given a set S of non-negative integers $[a_1, a_2, \ldots, a_n]$, consider the following problems:

- **Partition**: Determine whether there is a subset $P \subseteq S$ such that $\sum_{i \in P} a_i = \sum_{j \in S \setminus P} a_j$
- **Subset Sum**: Given some integer k, determine whether there is a subset $P \subseteq S$ such that $\sum_{i \in P} a_i = k$

(a) Find a linear time reduction from Partition to Subset Sum. Then prove your reduction is correct.

(b) Find a linear time reduction from Partition to Knapsack (for Knapsack, refer to pages 164-168 of textbook). Then, prove your reduction is correct.

(c) Find a linear time reduction from Subset Sum to Partition. Then prove your reduction is correct. (Hint: think about adding certain elements to the set S)

2. (★★★★ level) Finding Zero(s)

Consider the problem INTEGER-ZEROS.

INPUT: A multivariate polynomial $P(x_1, x_2, x_3, \ldots, x_n)$ with integer coefficients, specified as a sum of monomials.

OUTPUT: Integers $a_1, a_2, \ldots a_n$ such that $P(a_1, a_2, a_3, \ldots, a_n) = 0$.

Show that 3-SAT reduces in polynomial time to INTEGER-ZEROS. (You do not need to show that INTEGER-ZEROS is in **NP**: in fact, it is known *not* to be in **NP**).

Hint 1: Given a 3-SAT formula ϕ in the variables $x_1, x_2, \ldots, x_n$, your reduction f will produce a polynomial P in the same variables such that satisfying assignments correspond to $0, 1$ valued zeros of P.

Hint 2: If your polynomial constructed above has exponential amount of terms, it is not optimal! You need to reduce it to polynomial amount. Think about the equality $a^2 + b^2 = 0$ if and only if $a = b = 0$ when a, b are real to achieve it. [1]

[1] Fun fact: This problem INTEGER-ZEROS is *really* hard: the decision version of the problem is *undecidable*. This means that there is provably no algorithm which, given a multivariate polynomial, will decide correctly whether or not it has integer zeros. However, if we relax the problem to ask if there are *real* numbers at which the given polynomial vanishes, then the problem surprisingly becomes decidable! For Blue and Gold jingoists: the above decidability result was proven by Alfred Tarski, a Berkeley professor, in 1949. The undecidability of INTEGER-ZEROS was proven by Yuri Matiyasevich (at the ripe old age of 23!) in 1970, building upon previous work of another Berkeley professor, Julia Robinson.

3. (★★★ level) Connectivity-Preserving Subgraph

In the CONNECTIVITY-PRESERVING SUBGRAPH problem (CPS), we are given:

1. A directed graph $G = (V, E, w)$ with nonnegative weights w.
2. A set of k *connectivity demands* (s_i, t_i).

The task is to find a subgraph $H \subseteq G$, if one exists, such that for every demand (s_i, t_i), there exists an $s_i \to t_i$ path in H. Furthermore, this subgraph should minimize the total edge weight $\sum_{e \in H} w(e)$.

(a) Suppose CPS is restricted to having only one connectivity demand. What familiar problem is this?
Here, no proof is needed. Just state the problem.

(b) It turns out that CPS (for general k) is an **NP**-hard optimization problem. Give a k-approximation algorithm for CPS.
Describe the algorithm precisely; justify its correctness and running time. No pseudocode.

4. (★★★ 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 algorithm with ratio at most 2 for the case $k = 3$. (Hint: use part a)

(c) Give an approximation algorithm with ratio at most $k - 1$ for any k.

(d) **For Fun (No extra credit)** Give an approximation with ratio strictly less than 2 for the case $k = 3$. Anything less than 2 will get 1 point, but a ratio of at most 4/3 will get 2 points.

(e) **For Fun (No extra credit)** Give an approximation with ratio strictly less than 2 for general k.

5. (★★★★ 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:

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**
        $v_j \leftarrow$ the node in $V \setminus \tau$ that is closest to any node v_i in τ
        $v_k \leftarrow$ the node that follows v_i in τ.
        Modify τ by replacing v_i, v_k with v_i, v_j, v_k

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.

6. (★★★★★ level) Steiner Tree

The Steiner tree problem is the following:

Input: An undirected graph $G = (V, E)$ with non-negative edge weights $\text{wt} : E \to \mathbb{N}$, a set $S \subseteq V$ of special nodes.

Output: A Steiner tree for S whose total weight is minimal

A Steiner tree for S is a tree composed of edges from G that spans (connects) all of the special nodes S. In other words, a Steiner tree is a subset $E' \subseteq E$ of edges, such that for every $s, t \in S$, there is a path from s to t using only edges from E'. The total weight of the Steiner tree is the sum of the weights included in the tree, i.e., $\sum_{e \in E'} \text{wt}(e)$.

The Steiner tree problem has many applications in different areas, including creating genealogy trees to represent the evolutionary tree of life, designing efficient networks, to even planning water pipes or heating ducts in buildings. Unfortunately, it is NP-hard.

Here is an approximation algorithm for this problem:

1. Compute the shortest distance between all pairs of special nodes. Use these distances to create a modified and complete graph $G' = (S, E_S)$, which uses the special nodes as vertices, and the weight of the edge between two vertices is the shortest distance between them.
2. Find the minimal spanning tree of G'. Call it M.
3. Reinterpret M to give us a Steiner tree for G: for edge in M, say an edge (r, s), select the edges in G that correspond to the shortest path from r to s. Put together all the selected edges to get a Steiner tree.

Prove that this algorithm achieves an approximation ratio of 2.

(Note that the efficiency of an approximation algorithm does not contradict NP-completeness because it gives an approximate (rather than an exact) solution to the problem.)