Algorithms, 2023 Fall, Homework 8
(Due: Nov 22)

November 15, 2023

Problem 1 : (strong NP-Completeness) In the NPC proof of the subset sum problem, for each hyperedge e, we created a number w_e, which has $3n$ digit. Hence, the value of w_e is in fact exponentially large. Such a reduction is called a weakly polynomial time reduction (the value of the created instance may be exponentially large). Otherwise, the reduction is a strongly polynomial time reduction (all other reductions we did in the class were strongly polynomial). Show it is not possible to obtain a NPC proof for subset sum by a strongly polynomial time reduction.

If we can only prove a problem is NPC by weakly polynomial time reduction, the problem is weakly NP-complete. So subset sum and knapsack problems are weakly NP-complete. Otherwise, it is strongly NP-complete. Argue that strongly NP-completeness implies that there is even no pseudo-polynomial algorithm for the problem (of course assuming $P \neq NP$).

Problem 2 : We introduced a strongly NPC problem in previous class, called 3-PATITION problem, which is very similar to the subset sum problem. In this problem, we are given $3n$ positive integers $a_1, \ldots, a_n$ and a bound $B \in \mathbb{Z}^+$, such that $B/4 < a_i < B/2$ Can we partition the numbers into n groups, each having exactly 3 numbers, such that the sum of each group is exact B. The problem is strongly NP-complete. In other words, the problem is NP-complete even B is polynomially bounded (i.e., $B \leq \mathsf{poly}(n)$). You can assume the strongly NP-completeness of the problem from now on, even we are not going to prove it.

Using 3-PARTITION problem, show the following problem is NPC in the strong sense. We are given a set T of n tasks. Each task i has a release time r_i, a deadline d_i and a length ℓ_i. The question is whether there is a feasible schedule for all tasks in T. (in your proof, all numbers r_i, d_i, ℓ_i have to be bounded by $\mathsf{poly}(n)$).

Problem 3 : Consider the #(independent set) problem, that is to count how many independent sets (of any size) in a given graph G. This problem is in fact #P-hard (which is the counting version of the NP-hardness, informally). So it is unlikely we can solve the problem in polynomial time (if #P$\neq$ P). Show for #(independent set), we can either obtain an FPTAS (fully polynomial-time approximation scheme, that is an algorithm which can produce a $(1 \pm \epsilon)$-approximation in $poly(\text{input size}, 1/\epsilon)$ time), or the problem can not be approximated within any polynomial factor. (hint: show if there exists an algorithm with polynomial factor, you can boost the ratio to an FPTAS).

Problem 4 : KT book pp 655, problem 8.

Problem 5 : (KT pp.658 problem 12, Facility Location) You are given a graph $G(V, E)$ with distance measure d. d is not necessarily a metric. Each node s has a facility opening cost f_s. You are asked to open a set A of facilities to provide services to every node in the graph. Given the set A of opened facilities, the service cost of a node v is defined to be $d(v, A) = \min_{s \in A} d(v, s)$. The goal is to select a subset A such that the total cost (facility cost plus service cost)

$$\sum_{s \in A} f_s + \sum_{v \in V} d(v, A)$$

is minimized. Design a polynomial time approximation algorithm with an approximation factor of $O(\log |V|)$. (hint: recall the approximation algorithm for the set cover problem)

Sidenote: If the distance d satisfies the triangle inequality, there are constant factor approximation algorithms for the problem, using a variety of different techniques, such as linear program rounding, primal dual, local search.

Problem 6 : (KT pp657, problem 11) In the approximation algorithm for knapsack, we chosen to discretize the value of each item. Instead of discretizing the value, try to discretize the weight of each item. Show what approximation guarantee you can achieve using this approach. (it may not be a true approximation, in the sense that the solution may not be feasible. But you are allowed to find solutions that are "approximately feasible". Formalize what guarantee you can get.)

Problem 7 : In this problem, we prove that the vertex solution of the vertex cover Linear Program relaxation is half-integral: Every vertex of the polyhedron defined by the vertex cover Linear Program relaxation has coordinates that all belong to $\{0, 1/2, 1\}$. (Hint: We need the following fact: for every vertex of polyhedron P, there is a coefficient vector c such that $\{c^T x \mid x \in P\}$ achieves the maximum value at the vertex. Show that if a vertex $x \in P$ is not half integral, there is some vector c such that the value of $c^T x$ can be improved.)

Bonus problems: Bonus problems are problems that may be (may be not) more challenging than other problems. You do not lose points if you do not answer bonus problems. But you will get extra points if you get correct answers for such problems. **To make your life easier, we give you two weeks for the bonus problems)**.

Problem 1 : A unit disk graph is an intersection graph of unit disks: In particular, we are given a set of unit disks in $\mathbb{R}^2$, Each vertex corresponds to a unit disk. Two vertices are connected by an edge iff two corresponding disks intersect.

Let us consider the maximum independent set problem in unit disk graphs (assume the unit disk representation is given as the input). We would like to obtain a $1 - \epsilon$-approximation (PTAS) for a fixed constant $\epsilon > 0$. ℓ is a positive integer (it should only depends on ϵ) you need to fix (which is the side length of a grid cell). Suppose the grid is $\Xi_{a,b} = \cup_{i \in \mathbb{Z}}(x = i\ell + a) \cup \cup_{i \in \mathbb{Z}}(y = i\ell + b)$. Let $G_{a,b}$ be the unit disk graph obtained by the disks which do not intersect $\Xi_{a,b}$. Show that there exist $0 \leq a, b \leq \ell$

such that

$$|\text{max independent set of } G_{a,b}| \geq (1-\epsilon)\mathsf{OPT}.$$

(we call this technique the shifting technique. It has many applications in approximation algorithms). I have provided enough hints. Now, it is up to you to state the entire algorithm and provide a complete analysis (also state the running time).