

Problem Set 4—NOT TO BE TURNED IN

(30) **Problem 1.** In class we showed we can convert a Steiner tree input that doesn't satisfy the triangle inequality to one that does satisfy it and this preserves approximations (we can use an approximate solution in the transformed setting to get a good approximation for the original setting).

a) Show that the theorem we proved in class on transforming an arbitrary Steiner input to a metric one shows that if we have an ϵ -approximation algorithm for metric Steiner tree, then we also get an ϵ -approximation algorithm for any Steiner problem (metric or not)

b) Consider transforming a TSP problem that doesn't satisfy the triangle inequality to one that does (use the same trick as for Steiner tree, i.e. use shortest path distances). Does this produce a new problem that does satisfy the triangle inequality?

c) Show why the transformed problem cannot be used to give us a good approximation for general TSP.

(30) **Problem 2.** Problem 3.2 in the text.

(25) **Problem 3.** In class we discussed clique problem: given an undirected graph $G=(V,E)$ find a set C of vertices such that every pair of vertices in C is connected by an edge in E .

In the yes-no version where we require the set C to be of size at least k (where k is an input to the problem), then the problem is NP-complete. If we want the largest size set it is NP-hard.

a) Suppose that k were fixed (that is we have the k -vertex clique problem: given a graph G find a clique of size at most k) so the graphs change, but not k . Show that this is solvable in polynomial time.

b) Clearly if we could solve the optimization version (find a largest clique), we could also solve the yes-no version. Show the reverse is also true: if we can solve the yes-no version in polynomial time, we can also solve the optimization version in polynomial time. (hint: you may need multiple calls to the yes-no solver).

This second result shows that either both versions are in P, or neither is.

(30) **Problem 5.** Consider the following scheduling problem. You are given n jobs which you want to schedule on p processors. There are a total of k resources which a job might need to be able to be run (think of these as memory, disk, ...). $R_i, i = 1, 2, \dots, k$ is the total amount of resource i available. $r_{i,j}$ is the amount of resource i required by job j in order to be executed. A set of jobs $S, |S| \leq p$, can all be run simultaneously only if for each resource i , the sum of $r_{i,j} \leq R_i$ when summed over all jobs j in S (the total amount of resource i available meets the total amount demanded by all jobs in S).

Each job requires one unit of processing time, and the goal is to complete the final job as early as possible. There are no release times, deadlines or precedence constraints.

For each of the following settings show that the problem is: a) polynomial by giving a polynomial time algorithm, or b) weakly NP-hard by proving it is NP-hard and giving a pseudopolynomial algorithm, or c) strongly NP-hard by giving a strong NP-hardness proof.

- i) There are 3 different resources and two processors.
- ii) Same as i) but there are 3 processors.
- iii) There is only one resource and $n/2$ processors.
- iv) Same as iii) except our goal is to schedule a set of jobs which uses as much of the single resource as possible (thus we want a set of jobs which ideally uses a total of R_1 units of resource 1, or if not, uses as close to this as possible).