Tutorial 1

Approximation and Online Algorithms

Submission Deadline - 16th August 2026

All problems marked (*) need to be submitted by the submission deadline.

---

1. Design a $\frac{1}{2}$-approximation algorithm for the problem of finding a maximum cardinality set of edges from a given graph so that the resulting subgraph is acyclic.

2. Design a factor 2 approximation algorithm for the problem of finding a minimum cardinality maximal matching in an undirected graph.

3. Consider the following factor 2 approximation algorithm for the cardinality vertex cover problem: Find a depth first search tree in the given graph, G, and output the set, say S, of all the non-leaf vertices of this tree. Show that S is indeed a vertex cover for G and $|S| \leq 2 \cdot \text{OPT}$.

4. Following is the k-suppliers problem: You are given a positive integer k, and a set of vertices V, along with distances d_{ij} between any two vertices i, j that obey the same properties as in the k-center problem. However, now the vertices are partitioned into suppliers $F \subseteq V$ and customers $D = V \setminus F$. The goal is to find k suppliers such that the maximum distance from a supplier to a customer is minimized. In other words, we wish to find $S \subseteq F$ with $|S| \leq k$ that minimizes $\max_{j \in D} d(j, S)$.

   (a) Give a 3-approximation algorithm for the k-suppliers problem.

   (b) Prove that there is no α-approximation algorithm for $\alpha < 3$ unless P = NP

5. Show that for any input to the problem of minimizing the makespan on identical parallel machines for which the processing requirement of each job is more than one-third the optimal makespan, the longest processing time (LPT) rule computes an optimal schedule.

6. We consider scheduling jobs on identical machines, but jobs are now subject to precedence constraints. We say $i \prec j$ if in any feasible schedule, job i must be completely processed before job j begins processing.

   A natural variant on the list scheduling algorithm is one in which whenever a machine becomes idle, then any remaining job that is available is assigned to start processing on that machine. A job j is available if all jobs i such that $i \prec j$ have already been completely processed.

   Show that this list scheduling algorithm is a 2-approximation algorithm for the problem with precedence constraints.

7. In this problem, we consider a variant of the problem of scheduling on parallel machines so as to minimize the length of the schedule. Now each machine i has an associated speed s_i, and it takes p_j/s_i units of time to process job j on machine i. Assume that machines are numbered from 1 to m and ordered such that $s_1 \geq s_2 \geq \cdots \geq s_m$. We call these *related machines*.

   (a) A ρ-relaxed decision procedure for a scheduling problem is an algorithm such that given an instance of the scheduling problem and a deadline D, either produces a schedule of length at most $\rho \cdot D$ or correctly states that no schedule of length D is possible for the instance. Show that given a polynomial-time ρ-relaxed decision procedure for the problem of scheduling related machines, one can produce a ρ-approximation algorithm for the problem.

(b) Given a deadline D, we label every job j with the slowest machine i such that the job could complete on that machine in time D; that is, $p_j/s_i \leq D$. If there is no such machine for a job j, it is clear that no schedule of length D is possible.

If machine i becomes idle at a time D or later, it stops processing. If machine i becomes idle at a time before D, it takes the next job of label i that has not been processed, and starts processing it. If no job of label i is available, it looks for jobs of label $i+1$; if no jobs of label $i+1$ are available, it looks for jobs of label $i+2$, and so on. If no such jobs are available, it stops processing. If not all jobs are processed by this procedure, then the algorithm states that no schedule of length D is possible.

Prove that this algorithm is a polynomial-time 2-relaxed decision procedure.

8. (*) In the *minimum-cost Steiner tree problem*, we are given as input a complete, undirected graph $G = (V, E)$ with nonnegative costs $c_{ij} \geq 0$ for all edges $(i, j) \in E$. The set of vertices is partitioned into *terminals* R and *nonterminals* (or *Steiner vertices*) $V - R$. The goal is to find a minimum-cost tree containing all terminals.

(a) Suppose initially that the edge costs obey the triangle inequality; that is, $c_{ij} \leq c_{ik} + c_{kj}$ for all $i, j, k \in V$. Let $G[R]$ be the graph induced on the set of terminals; that is, $G[R]$ contains the vertices in R and all edges from G that have both endpoints in R. Consider computing a minimum spanning tree in $G[R]$. Show that this gives a 2-approximation algorithm for the minimum-cost Steiner tree problem.

(b) Now we suppose that edge costs do not obey the triangle inequality, and that the input graph G is connected but not necessarily complete. Let c'_{ij} be the cost of the shortest path from i to j in G using input edge costs c. Consider running the algorithm above in the complete graph G' on V with edge costs c' to obtain a tree T'. To compute a tree T in the original graph G, for each edge $(i, j) \in T'$, we add to T all edges in a shortest path from i to j in G using input edge costs c. Show that this is still a 2-approximation algorithm for the minimum-cost Steiner tree problem on the original (incomplete) input graph G. G' is sometimes called the *metric completion* of G.