최적화근사해법
제 2주 과제

홍 성 필

서울대학교 산업공학과
최적화 연구실

A supplementary lecture note for the book *Approximation algorithms* by V. Vazirani

2007년 봄학기

1.1 Give a factor 1/2 algorithm for the following.
Problem 1.9(Acyclic subgraph) Give a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic.

1.3 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 indeed a vertex for G and $|S| \leq 2 \cdot \mathrm{OPT}$.

1.13 Let L be the language consisting of all prime numbers. Show that $L \in \mathrm{NP}$.

2.1 Given an undirected graph $G = (V, E)$, the *cardinality maximum cut problem* asks for a partition of V into sets S and $\overline{S}$ so that the number of edges running between these sets is maximized. Consider the following greedy algorithm for this problem. Here v_1 and v_2 are arbitrary vertices in G, and for $A \subset V, d(v, A)$ denotes the number of edges running between vertex v and set A.

Algoruthm 2.13

1. Initialization:
   $A \leftarrow \{v_1\}$
   $B \leftarrow \{v_2\}$
2. For $v \in V - \{v_1, v_2\}$ do:
   if $d(v, A) \geq d(v, B)$ then $B \leftarrow B \cup \{v\}$
   else $A \leftarrow A \cup \{v\}$.
3. Output A and B.

2.2 Consider the following algorithm for the maximum cut problem, based on the of *local search*. Given a partition of V into sets, the basic step of the algorithm, called *flip*, is that of moving a vertex from one side of the other. The following algorithm finds a *locally optimal solution* under the flip operation, i.e., a solution which cannot be improved by a single flip.
The algorithm starts with an arbitary of V. While there is a vertex such that flipping it increase the size of the cut, the algorithm flips sych a vertex. (Observe the a vertex qualities for a flip if it has more neighbors in its own partition than in the other side.) The algorithm terminates when no vertex qualities for a flip. Show that this algorithm terminates in polynomial time, and achieves an approximation guarantee of 1/2.

2.3 Consider the following generalization of the maximum cut problem.
Problem 2.14(MAX k-CUT Given an undirected graph $G = (V, E)$ with nonnegative edge costs, and on integer k, find a partition of V into sets $S_1, \ldots, S_k$ so that the total cost of edges running between these sets is maximized.
Give a greedy algorithm for this problem that achieves a factor of (1-$\frac{1}{k}$). Is the analysis of your algorithm of factor 1/4.

2.13 Use layering to get a factor f approximation algorithm for set cover, where f is the frequency of the most frequent element. Provide a tight example for this algorithm.

3.2 Let $G = (V, E)$ be a graph with nonnegative edge costs. S, the *senders* and R, the *receivers*, are disjoint subsets of V. The problem is to find a minimum cost subgraph of G that has a path connecting each receiver to a sender (any sender suffices). Partition the instances into two cases: $S \cup R = V$ and $S \cup V \neq V$. Show that these two cases are in **P** and **NP**-hard, respectively.

3.3 Give an approximation factor preserving reduction from the set cover problem to the following problems, thereby showing that it is unlikely to have a better approximation guarantee than $O(\log n)$.

Problem 3.14 (Directed Steiner tree) $G = (V, E)$ is a directed graph with nonnegative edge costs. The vertex set V is partitioned into two sets, *required* and *Steiner*. One of the required vertices, r, is special. The problem is to find a minimum cost tree in G rooted into r that contains all the required vertices and any subset if the Steiner vertices.

3.6 Give an $O(\log n)$ factor approximation algorithm for the following problems.

Problem 3.15 (Asymmetric TSP) We are given a directed graph G on vertex set V, with a nonnegative cost specified for edge $(u \to v)$, for each pair $u, v \in V$. The edge costs satisfy *the directed triangle inequality*, i.e., for any three vertices u, v, and w, $\text{cost}(u \to v) \leq \text{cost}(u \to w) + \text{cost}(w \to v)$. The problem is to find a minimum cost cycle visiting every vertex exactly once.