CS387: Approximation Algorithms

Homework 1

Please submit your solutions to chao.liao.95@gmail.com (as PDF files obtained from LaTeX or Word sources).

You may work in groups of up to three students. Please include the information (student name and id) of all group members in your solution.

Deadline: Wednesday, 6 July 11:59pm

Problem 1 (Maximum Independent Set)

Given a graph G, an *independent set* is a set of vertices such that no two are neighbors. The Maximum Independent Set (MIS) problem is a famous NP-complete problem. Interestingly, the complement of an independent set is a vertex cover, so the complement of the MIS is the minimum vertex cover. Thus, solving one exactly also solves the other exactly. We've seen how to get a 2-approximation for vertex cover. Even though it is complementary, the situation for MIS is much worse.

Early in the study of approximation hardness, MIS was shown to be MAX-SNP-hard, meaning there is some constant to within which it *cannot* be approximated (unless P=NP). This is also true for vertex cover, but the situation for MIS is much worse as the following problem shows.

In class we used a "rescaling" argument to show that for many problems, if one had an absolute approximation algorithm, one could transform it into a better absolute approximation algorithm for the problem (and in fact, solve the problem exactly). Here we will show that for MIS, a similar result holds true for relative approximation: that any constant-factor relative approximation algorithm can be improved to a better constant factor relative approximation.

Suppose one has an α-(relative) approximation algorithm for MIS. Consider the following "graph product" operation for a graph G. Create a distinct copy G_v of G for each vertex v of G. Then connect up the copies as follows: if (u, v) is an edge of G, then connect every vertex in G_u to every vertex in G_v.

(a) Prove that if there is an independent set of size k in G, then there is an independent set of size k^2 in the product graph.

(b) Prove that given an independent set of size s in the product graph, one can find an independent set of size $\sqrt{s}$ in G.

(c) Prove that if there is an α-approximation for MIS for some constant α, then there is a polynomial approximation scheme for MIS.

Since MIS was shown to be MAX-SNP-hard, meaning there is some constant to within which it cannot be approximated (unless P = NP) this proves that MIS has no constant-factor relative approximation.

Problem 2 (Bin Packing)

The following is the NP-hard problem of bin packing: Given n items with sizes $a_1, ..., a_n \in (0, 1]$, find a packing of the items into unit-sized bins that minimizes the number of bins used. Let B^* denote the optimum number of bins for the given instance. Bin packing is a lot like $P||C_{max}$, but somewhat more difficulty because you have no flexibility to increase the bin sizes.

(a) Suppose that there are only k distinct item sizes for some constant k. Argue that you can solve bin-packing in polynomial time. **Hint:** no new algorithm needed here!

(b) Suppose that you have packed all items of size greater than ϵ into B bins. Argue that in linear time you can add the remaining small items to achieve a packing using at most $\max\{B, 1 + (1 + \epsilon)B^*\}$ bins.

(c) For $P||C_{max}$, we reduced to the previous case by rounding each job size up to the next power of $(1 + \epsilon)$. Why doesn't that work for bin packing?

(d) Consider instead the following grouping procedure. Fix some constant k. Order the items by size. Let S_1 denote the largest n/k items, S_2 the next largest n/k, and so on. Suppose that you increase the size of each item to equal the largest size in its group, so that there are only k distinct sizes. Argue that this increases the optimal number of bins by at most n/k. **Hint:** imagine setting aside the jobs in S_1. Argue that the remaining items, with their increased sizes, can still fit into the bins used by the original packing.

(e) By applying the grouping procedure to items of size greater than $\epsilon/2$, solving the result optimally, and then adding the small items, devise a polynomial time scheme that uses at most $(1 + \epsilon)B^* + 1$ bins.

Observe that the algorithm above just misses the definition of polynomial approximation scheme, because of the additive error of 1 bin. In practice, of course, this is unlikely to matter. The above scheme is known as an *asymptotic PTAS* since its approximation ratio is $(1 + \epsilon)$ in the limit as the optimum value grows.

The techniques above have been augmented to give an algorithm that finds a packing using $B^* + O(\log^2 B^*)$ bins — giving asymptotic approximation ratio 1. Indeed, at present it remains conceivable that we might achieve $B^* + O(1)$ bins in polynomial time!

Problem 3 (K-Cluster)

You are given a collection of n points in some metric space (i.e., the distances between the points satisfy the triangle inequality). Consider the problem of dividing the points into k clusters so as to minimize the maximum diameter of (distance between any two points in) a cluster.

(a) Suppose the optimum diameter d is known. Devise a greedy 2-approximation algorithm. **Hint:** consider any point and all points within distance d of it.

(b) Consider the algorithm that (k times) chooses as a "center" the point at maximum distance from all previously chosen centers, then assigns each point to the nearest center. By relating this algorithm to the previous algorithm, show that you get a 2-approximation.

Problem 4 (Scheduling)

Consider the problem of scheduling, on one machine, a collection of jobs with given *processing times* p_j, *due dates* d_j, and *lateness penalties (weights)* w_j paid for jobs that miss their due dates, so as to minimize the total lateness penalty. (If we let U_j denote the indicator variable for job j completing after its due date, we want to minimize $\sum w_j U_j$.)

(a) Argue that any feasible subset of jobs (that can all together be completed by their due dates) might as well be scheduled in order of increasing deadline (so it is sufficient to find a set without worrying about order). **Hint:** if two adjacent jobs in the sequence are out of order, swap them.

(b) Assuming the lateness penalties are polynomially bounded integers, give a polynomial time dynamic program that finds the fastest completing maximum weight feasible subset.

(c) Give a fully polynomial time approximation scheme for the original problem of minimizing lateness penalty with arbitrary lateness penalties.