CS 573: Graduate Algorithms, Fall 2011
HW 6 (will not be graded)

This homework is on approximation algorithms. Solve as many as you can as practice for the final exam. Discuss the problems on the newsgroup and we will aid the process.

1. The input to the Bin Packing problem consists of n items where item i has a given size $s_i \in [0, 1]$. The goal is to pack these items into the fewest possible number of bins each of which is of size 1. You have seen that this problem is NP-Hard. Typical heuristics consider the items in some (adaptive) order and pack the the current item into some existing bin or open a new bin. A greedy heuristic is one which opens a new bin only if the current item does not fit into an existing bin. Show that any greedy policy uses at most $2OPT$ bins. Can you prove a better approximation using a specific policy? Consider the First-Fit-Decreasing policy that considers the items in non-increasing size order and places the item in the first bin that it fits into. Show an approximation ratio strictly better than 2 for this policy. A well-known result is that this policy uses at most $\frac{11}{9}OPT + 4$ bins.

2. The maximum independent set problem is NP-Hard and morever it is known that unless $P = NP$ one cannot obtain a $1/n^{1-\epsilon}$-approximation for any fixed $\epsilon > 0$. However, reasonable results can be obtained in various special cases. Consider a simple greedy algorithm that picks a vertex of minimum degree, removes it and its neighbors from the graph and continues as long as the remaining graph is non-empty. Show that this heuristic always outputs a solution of size $\Omega(n/(1 + \Delta))$ where Δ is the *average degree* of the graph. First figure out the case when Δ is the maximum degree. Show that the greedy algorithm gives a constant factor approximation in planar graphs.

3. Vertex Cover.

   (a) Consider the following algorithm that constructs a vertex cover C of a graph G.

```
VertexCover($G$):
  $C \leftarrow \emptyset$
  while $C$ is not a vertex cover
      pick an arbitrary edge $uv$ that is not covered by $C$
      add either $u$ or $v$ to $C$
  return $C$
```

   Prove that VertexCover can return a vertex cover that is $\Omega(n)$ times larger than the optimal vertex cover. You need to describe both an input graph with n vertices, for any integer n, and the sequence of edges and endpoints chosen by the algorithm.

   (b) Now consider the following randomized variant of the previous algorithm.

```
RandomVertexCover($G$):
  $C \leftarrow \emptyset$
  while $C$ is not a vertex cover
      pick an arbitrary edge $uv$ that is not covered by $C$
      with probability $1/2$
          add $u$ to $C$
      else
          add $v$ to $C$
      return $C$
```

Prove that the expected size of the vertex cover returned by RandomVertexCover is at most 2OPT, where OPT is the size of the smallest vertex cover.

(c) Let G be a graph in which each vertex v has a weight $w(v)$. Now consider the following randomized algorithm that constructs a vertex cover.

```
RandomWeightedVertexCover($G$):
  $C \leftarrow \emptyset$
  while $C$ is not a vertex cover
      pick an arbitrary edge $uv$ that is not covered by $C$
      with probability $w(v)/(w(u) + w(v))$
          add $u$ to $C$
      else
          add $v$ to $C$
      return $C$
```

Prove that the expected weight of the vertex cover returned by RandomWeightedVertexCover is at most 2OPT, where OPT is the weight of the minimum-weight vertex cover. A correct answer to this part automatically earns full credit for part (b).

4. Let $G = (V, A)$ be a directed graph with arc weights $c : A \to \mathcal{R}^+$. Define the density of a directed cycle C as $\sum_{a \in C} c(a)/|V(C)|$ where $V(C)$ is set of vertices in C.

   (a) A cycle with the minimum density is called a minimum mean cycle and such a cycle can be computed in polynomial time. How?

   Hint: Given density λ, give a polynomial-time algorithm to test if G contains a cycle of density $< \lambda$. Now use binary search.

   (b) Consider the following algorithm for ATSP. Given G (with c satisfying asymmetric triangle inequality), compute a minimum mean cycle C. Pick an arbitrary vertex v from C and recurse on the graph $G' = G[V - C \cup \{v\}]$. A solution to the problem on G can be computed by patching C with a tour in the graph G'. Prove that the approximation ratio for this heuristic is at most $2H_n$ where $H_n = 1 + 1/2 + \ldots + 1/n$ is the nth harmonic number.

5. Consider the k-dimensional knapsack problem. We are given n non-negative k-dimensional vectors $\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n$. Each vector has a non-negative weight, w_i for $\bar{v}_i$. We are also given a k-dimensional knapsack $\bar{V}$ and the goal is to find a maximum weight subset of vectors that pack in to V. A subset of vectors pack into $\bar{V}$ if their vector addition is less than $\bar{V}$ (co-ordinate wise). Prove that there is a pseudo-polynomial time algorithm for this problem when k is a fixed constant independent of n. What is the running time of your algorithm?

Prove that even for $k = 2$ and *unit weights* the problem does not admit an FPTAS. (Hint: use a reduction from the Partition problem.) **Extra credit:** Obtain a PTAS for this problem when k is a fixed constant independent of n (might need to use LP techniques).

6. Multi-processor scheduling: given n jobs $J_1, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$ and m machines $M_1, M_2, \ldots, M_m$. For identical machines greedy list scheduling that orders the jobs in non-increasing sizes has an approximation ratio of 4/3. See Section 2.3 in the Shmoys-Williamson book.

   Now consider the problem where the machines are not identical. Machines M_j has a speed s_j. Job J_i with processing time p_i takes p_i/s_j time to complete on machine M_j. Give a constant factor approximation for scheduling in this setting to minimize makespan (the maximum completion time). (Hint: consider jobs in decreasing sizes. Assuming $p_1 \geq p_2 \geq \ldots \geq p_n$ and $s_1 \geq s_2 \geq \ldots s_m$, show that $OPT \geq \max_{i \leq m}(\sum_{j \leq i} p_j / \sum_{j \leq i} s_j.)$

7. The input consists of a metric space (V, d) and k the goal is to find a subset $S \subseteq V$ of at most k centers to $\min \max_{v \in V} d(v, S)$. We saw a simple 2-approximation algorithm for k-center in class (do you recall the algorithm and its proof of correctness?). Here is another algorithm that also gives a 2-approximation. First assume you know the optimal solution value R^*. While there are vertices left pick an arbitrary vertex v, add it to the solution and remove all vertices at distance $2R^*$ from v. Show that this algorithm terminates with at most k vertices and gives a 2-approximation. How do you eliminate the assumption that you know the value R^*?

8. In the Multiple Knapsack problem you are given n items each with a size and profit as in the standard knapsack problem. However now you have m knapsacks each of the same size B. Consider a greedy algorithm that iteratively packs the next knapsack with the remaining items via some α-approximation for the single knapsack problem. Assuming $\alpha = 1$, that is, you have an exact algorithm for knapsack show that this algorithm gives a $(1 - 1/e)$-approximation for the multiple knapsack problem. Extra Credit: Suppose now the knapsacks are of different capacities $B_1, \ldots, B_m$. Show that the algorithm gives a 1/2-approximation (assuming $\alpha = 1$) irrespective of the order in which the knapsacks are considered. Can you analayze the approximation ratios if $\alpha > 1$?