Problem Set 9 Solutions

This problem set is due **at 11:59pm** on **Thursday, April 30, 2015.**

Exercise 9-1. Read CLRS, Chapter 35.

Exercise 9-2. Exercise 35.2-3.

Exercise 9-3. Exercise 35.4-2.

Exercise 9-4. Read Prof. Demaine's notes on fixed-parameter algorithms.

Problem 9-1. Knapsack [25 points]

In the Knapsack problem, we are given a set $A = \{a_1, \ldots, a_n\}$ of items, where each a_i has a specified positive integer *size* s_i and a specified positive integer *value* v_i. We are also given a positive integer *knapsack capacity* B. Assume that $s_i \leq B$ for every i. The problem is to find a subset of A whose total size is at most B and for which the total value is maximized.

In this problem, we will consider approximation algorithms to solve the Knapsack problem.

Notation: For any subset S of A, we write s_S for the total of all the sizes in S and v_S for the total of all the values in S. Let Opt denote an optimal solution to the problem.

(a) [5 points] Consider the following greedy algorithm Alg_1 to solve the Knapsack problem:
Order all the items a_i in non-increasing order of their *density*, which is the ratio of value to size, $\frac{v_i}{s_i}$. Make a single pass through the list, from highest to lowest density. For each item encountered, if it still fits, include it, otherwise exclude it.

Prove that algorithm Alg_1 does not guarantee any constant approximation ratio. That is, for any positive integer k, there is an input to the algorithm for which the total value of the set of items returned by the algorithm is at most $\frac{v_{Opt}}{k}$.

Solution: Fix any k. Consider the two-item input list a_1, a_2, where $v_1 = 2$, $s_1 = 1$, $v_2 = 2k$, and $s_k = 2k$, with knapsack capacity $B = 2k$. Since $\frac{v_1}{s_1} = 2$ and $\frac{v_2}{s_2} = 1$, Alg_1 considers the items in order a_1, a_2, and includes a_1 and excludes a_2. The total value achieved is only 2, but the largest achievable total value is $2k$.

(b) [7 points] Consider the following algorithm Alg_2.
If the total size of all the items is $\leq B$, then include all the items. If not, then order all the items in non-increasing order of their densities. Without loss of generality, assume that this ordering is the same as the ordering of the item indices. Find the smallest index i in the ordered list such that the total size of the first i items exceeds B (i.e., $\sum_{j=1}^{i} s_j > B$, but $\sum_{j=1}^{i-1} s_j \leq B$). If $v_i > \sum_{j=1}^{i-1} v_j$, then return $\{a_i\}$. Otherwise, return $\{a_1, \ldots, a_{i-1}\}$.
Prove that Alg_2 always yields a 2-approximation to the optimal solution.

Solution: We know that fractional knapsack problem can be solved via greedy algorithm; the optimal solution for fractional knapsack problem takes first $i-1$ items, and takes some fraction α of item i. That is, fractional knapsack optimal solution v_{fOpt} is

$$v_{fOpt} = \sum_{j=1}^{i-1} v_j + \alpha v_i.$$

Since every knapsack problem is a valid fractional knapsack problem, we know that $v_{Opt} \leq v_{fOpt}$.
Now consider the solution output by Alg_2.

$$\max(\sum_{j=1}^{i-1} v_j, v_i) \geq \max(\sum_{j=1}^{i-1} v_j, \alpha v_i) \geq \frac{v_{fOpt}}{2},$$

because at least one of the two terms needs to be larger than half of v_{fOpt}. Therefore, we have

$$v_{Opt} \leq v_{fOpt} \leq 2\max(\sum_{j=1}^{i-1} v_j, v_i),$$

which shows that Alg_2 is a 2-approximation algorithm.

Solution: This is an alternate solution.
Assume for contradiction that there is some input instance for which Alg_2 does not achieve a 2-approximation to the optimal solution. Then for i as specified in the algorithm, the maximum of v_i and $\sum_{j=1}^{i-1} v_j$ is strictly less than $\frac{v_{Opt}}{2}$. That is, both v_i and $\sum_{j=1}^{i-1} v_j$ are strictly less than $\frac{v_{Opt}}{2}$. Then $\sum_{j=1}^{i} v_j < v_{Opt}$. We also know that $\sum_{j=1}^{i} s_j > B$.
Let $C = \{a_1, \ldots, a_i\} - Opt$, $D = Opt - \{a_1, \ldots, a_i\}$, and $E = \{a_1, \ldots, a_i\} \cap Opt$. We know that $v_C = \sum_{j=1}^{i} v_j - v_E$, and $v_D = v_{Opt} - v_E$. Since $\sum_{j=1}^{i} v_j < v_{Opt}$, we now have that $v_C < v_D$. We also know that $s_C > s_D$, since $\sum_{j=1}^{i} s_j > B$ and $s_D \leq B$. Therefore, the density of C is strictly less than the density of D (i.e., $\frac{v_C}{s_C} < \frac{v_D}{s_D}$).

Now recall that the densities of $\{a_1, \ldots, a_n\}$ are non-increasing. Consider $\frac{\sum_{j:v_j \in C} v_j}{\sum_{j:v_j \in C} s_j} - \frac{v_i}{s_j} = \frac{\sum_{j:v_j \in C}(v_j s_i - v_i s_j)}{\sum_{j:v_j \in C} s_i s_j}$. We know that $\frac{v_j}{s_j} \geq \frac{v_i}{s_i}$ for $a_j \in C$, and thus $v_j s_i - v_i s_j > 0$. This implies that $\frac{v_C}{s_C} \geq \frac{v_i}{s_i}$, and similar argument holds for $\frac{v_D}{s_D} \leq \frac{v_j}{s_j}$. Therefore, $\frac{v_C}{s_C} \geq \frac{v_i}{s_i} \geq \frac{v_D}{s_D}$. This contradicts our result from the previous paragraph that says $\frac{v_C}{s_C} < \frac{v_D}{s_D}$, and Alg_2 must be a 2-approximation.

(c) [5 points] Let $A = \{a_1, \ldots, a_n\}$ be the input ordered arbitrarily, and V be the largest value for any item; then nV is an upper bound on the total value that could be achieved by any solution. For every $i \in \{1, \ldots, n\}$ and $v \in \{1, \ldots, nV\}$, define $S_{i,v}$ to be the smallest total size of a subset of $\{a_1, \ldots, a_i\}$ whose total value is exactly v; $S_{i,v} = \infty$ if no such subset exists.

Give a dynamic programming algorithm Alg_3 to solve Knapsack exactly. Specifically, give a recurrence to compute all values of $S_{i,v}$, and explain how to use this to solve the Knapsack problem. Analyze its time complexity.

Solution:

$S_{1,v} = s_1$ if $v_1 = v$, and $= \infty$ otherwise.

For $1 \leq i \leq n-1$:

$S_{i+1,v} = \min\{S_{i,v}, s_{i+1} + S_{i,v-v_{i+1}}\}$ if $v_{i+1} \leq v$, $S_{i,v}$ otherwise.

The computation of all of these values takes time $O(n^2 V)$. As usual for dynamic programming algorithms, we can add bookkeeping to calculate the actual subsets as we go along. The maximum total value achievable is $\max v \| S_{n,v} \leq B$. The final output is a subset of A whose total value is this maximum.

Since the time complexity $O(n^2 V)$ depends linearly on V, which is represented in binary in the input to the problem, Alg_3 is a *pseudo-polynomial-time* algorithm.

(d) [8 points] Finally, we develop Alg_4, which is a *Fully Polynomial Time Approximation Scheme (FPTAS)* for Knapsack. The idea is to use an exact dynamic programming algorithm like the one in Part (c), but instead of using the given (possibly large) values for the items, we use versions of the given values that are suitably scaled and rounded down. As in Part (c), order $A = \{a_1, \ldots, a_n\}$ arbitrarily, and let V be the largest value for any item.

For any $\varepsilon, 0 < \varepsilon < 1$, Alg_4 behaves as follows:

For each item a_i with value v_i, define a scaled value $v_i' = \lfloor (\frac{v_i}{V})(\frac{n}{\varepsilon}) \rfloor$.

Using these scaled values (and the given sizes), run Alg_3 and output the set C of items that it returns.

Prove that Alg_4 is a FPTAS for Knapsack.

Solution:

By Part (c), the running time of the algorithm is $O(n^2\lfloor(\frac{V}{V})(\frac{n}{\varepsilon})\rfloor) = O(n^2\lfloor\frac{n}{\varepsilon}\rfloor)$, which is polynomial in n and $\frac{1}{\varepsilon}$.

It remains to show that the set C returned by the algorithm has total value $v_C \geq (1-\varepsilon)v_{Opt}$.

Let K denote $\frac{\varepsilon V}{n}$, so each $v'_i = \lfloor\frac{v_i}{K}\rfloor$. It follows that $v'_i K \geq v_i - K$. Considering all the elements of O, we get $v'_{Opt}K \geq v_{Opt} - nK$.

Now consider the set C that is returned by the dynamic programming step. We have that $v_C \geq Kv'_C$. Also, since the set C is optimal in terms of the scaled values, we have $v'_C \geq v'_{Opt}$. Therefore, we have:
$v_C \geq Kv'_C \geq Kv'_{Opt} \geq v_{Opt} - nK = v_{Opt} - \varepsilon V \geq v_{Opt} - \varepsilon v_{Opt} = (1-\varepsilon)v_{Opt}$. This is as needed.

Problem 9-2. Fixed-Parameter Algorithms [25 points]

We consider the *Tournament Edge Reversal* problem. Define a ***tournament*** to be a directed graph $T = (V, E)$ such that, for every pair of vertices $u, v \in V$, exactly one of (u, v) and (v, u) is in E. Furthermore, define a ***cycle cover*** to be a set $A \subset E$ of directed edges of a tournament T such that, every directed cycle of T contains at least one edge from A.

(a) [5 points] Let a minimal cycle cover of T be a cycle cover with the least number of edges. Prove that reversing all the edges of a minimal cycle cover A turns T into an acyclic tournament. (*Hint:* Any edge $e \in A$ must be the *only* edge in A on some directed cycle of T.)

Solution: Let T' be the new tournament. Suppose for contradiction that T' contains a cycle C. Let F' be the set of edges of C that were obtained by reversing the edges of A, and let F be their reversals, which are edges in T. For each $e \in F$, let C_e be a directed cycle of T that e covers. From the hint, we know e is the only edge in A that is also in C_e.

Now we can construct a cycle C' in T, consisting entirely of edges of T that did not get reversed: Include all reverse of edges in $C - F'$ (which are valid edges in T), and for each $e \in C \cap F'$, include all the edges of C_e except for e. This yields a cycle C' in T that contains no edges in A, which is a contradiction.

In the Tournament Edge Reversal problem, we are given a tournament T and a positive integer k, and the objective is to decide whether T has a cycle cover of size at most k.

(b) [15 points] Show that this problem has a kernel with at most $k^2 + 2k$ vertices. (*Hint:* Define a ***triangle*** to be a directed cycle of length 3. Consider the number of times that a node or an edge appears in different triangles.)

Solution:

Part (a) implies that a tournament T has a cycle cover of size at most k if and only if it can be turned into an acyclic tournament by reversing directions of at most k edges. We will use this characterization for the kernel. We give two simple reduction rules:

Rule 1: If an edge e is contained in $\geq k+1$ triangles, then reverse e and reduce k by 1.

This rule is safe because if we do not reverse e, we must reverse at least one edge from each of $k+1$ triangles containing e. Thus e belongs to every cycle cover of size $\leq k$.

Rule 2: If a vertex v is not contained in any triangle, then delete v from T.

To see why this is safe, let X be the set of vertices u such that T contains the edge (v,u) (the outgoing neighbors of v), and let Y be the set of vertices u such that T contains the edge (u,v) (the incoming neighbors of v). X and Y partition the vertices in $V - \{v\}$. Since v is not contained in any triangle, there is no edge from X to Y. Thus, every directed cycle in T is either wholly contained within the subgraph induced by X or the subgraph induced by Y. Therefore, removing v and its incident edges from T does not affect the size of the minimum cycle cover.

Thus, starting with instance $(T;k) = (T_0;k_0)$, we apply our reduction rules repeatedly until they cannot be applied any longer, obtaining a sequence of equivalent instances $(T_1;k_1), (T_2;k_2), \ldots, (T_m;k_m) = (T';k')$, where neither Rule is applicable to the final instance $(T';k')$.

Claim: If T' has a cycle cover A' of size $\leq k'$, then T' has at most $k'(k'+2)$ vertices.

Proof of claim: Since Rule 2 is not applicable, every vertex of T' is in a triangle, which must contain some edge in A'. Since Rule 1 is not applicable, for every edge $e \in A'$, there are at most k' vertices other than e's endpoints that are in triangles containing e. Since $|A'| \leq k'$, it follows that T' has at most $k'(k'+2)$ vertices.

Thus, after reducing, we consider the final instance (T',k'). If T' has more than $k'(k'+2)$ vertices, then it cannot have a cycle cover of size $\leq k'$, and is a no-instance. By equivalence, (T,k) is also a no-instance. We return no in this case. Otherwise, we get the desired kernel with at most k^2+2k vertices.

(c) [5 points] Obtain a FPT algorithm for the Tournament Edge Reversal problem.

Solution:

Given an instance (T,k) for the Tournament Edge Reversal problem, use the algorithm from Part (b) to either determine that (T,k) is a no-instance or obtain a kernel (T',k') where $k' \leq k$ and T' has at most $k'(k'+2)$ vertices. In the former case, answer "no". In the latter case, run any algorithm with running time $g(k')$, for any function g, to solve the Tournament Edge Reversal problem on (T',k'). Return the result as the answer.

To reduce the problem, we have to check for all triangles, and check membership of all edges and vertices in the triangles. For vertices, to check all triangles, it takes $O(\binom{|V|}{2})$ per vertex. For edges, it takes $|V|$ per edge. Each reduction step thus takes

$O(|V|^3+|V||E|))$. There are at most $V+E$ reduction steps since each step removes an edge or a vertex. In total, reduction takes $O(|V|^4+|V|^3|E|+|V|^2|E|+|V||E|^2)$. Since $O(|E|) = O(|V|^2)$ in tournaments, this is $O(|V|^5)$. To find the minimum cycle cover, one solution is to check all possible subsets of edges in T'. There are $O(2^{k'^2(k'+2)^2})$ subsets, and for each subset we need to run a cycle finding algorithm, such as DFS, to ensure all cycles are covered. Therefore, the final time complexity is $O(|V|^5)+2^{O(k^4)}$.