CSE 664: Homework #4

Daniel Bickley

Problem 1

LOAD-TRUCK Problem:
Input: Multi-set of integers $w = \{w_1, w_2, ...w_n\}$, integer K
Output: Set T of subsets of W
Goal: Minimize $|T|$, $\forall w \in W, w \in \bigcup_{i=o}^{|T|} T_i, \forall j \in T, \sum_{k=0}^{|T_j|} T_{j_k} \leq K$, every set in T is disjoint.

1. Give an example of a problem instance where the greedy algorithm will produce a set T where $|T| >$ the size of the optimal solution.

   Consider $w = \{5, 5, 5, 5, 5, 1, 1, 1, 1, 1\}, K = 6$.

   The size of T in the optimal solution is 5. Each $5 \in W$ can be paired with a $1 \in W$, resulting in each subset in T having a sum of $6 \leq K$. As there are five different pairs, the optimal solution will produce a set T such that $|T| = 5$.

   The greedy algorithm will produce a set T_g where $|T_g| = 6$. For T_1, T_2, T_3, T_4, the algorithm will add one integer of value 5 in each $T_{1...4}$. It will not be able to add the next integer to the subset (which will be a 5) without violating the size requirement for the subset. At this point in the greedy algorithm, the algorithm needs to place $\{5, 1, 1, 1, 1, 1\}$ into more subsets. The algorithm will create a subset of $\{5, 1\}$ and a second subset of $\{1, 1, 1, 1\}$. So, the greedy solution will generate a set T_g of size 6.

2. Show the greedy algorithm is a 2-approximation.

   Proof. Let O denote the optimal output to the problem. T denotes the output of the greedy algorithm. K denotes the maximum sum of each subset in the output T_m denotes the largest subset in T O_m denotes the largest subset in O W denotes the multi-set of integers available for the problem.

   The maxinum number of subsets that can be produced by the algorithm is $|W|$.

   Consider the set T. T is the set of size n of subsets created by the greedy algorithm.

   Consider T_1 and T_2. T_1 consists of elements $T_{1_1}, T_{1_2}, ...T_{1_n}$, while the elements of T_2 are $T_{2_1}, T_{2_2}, ...T_{2_n}$. In order for the greedy algorithm to produce a second subset, $\sum_{i=i}^{|T_1|} T_{1_i} + T_{21} \geq K$. So, $T_{21} = K - \sum_{i=i}^{|T_1|} T_{1_i}$. Let the difference between K and $\sum_{i=i}^{|T_1|} T_{1_i}$ be denoted as d_1.

   Consider the optimum set of subsets O. Each subset in O is generated by the same values that were used to generate T. The optimum subsets will minimize the additional space left in each subset. Let the difference between K and the ith subset be denoted as b_1 So,

   $$\sum_{i=1}^{|T|} d_i \geq \sum_{t=1}^{|O|} b_i$$

   Additionally, we know that $|T|K - \sum_{i=1}^{|T|} d_i$ is the sum of every $w_i \in W$. Additionally, $|O|k - \sum_{t=1}^{|O|} b_i$ is the sum of every $w_i \in W$. So,

   $$|T|K - \sum_{i=1}^{|T|} d_i = |O|k - \sum_{t=1}^{|O|} b_i$$

Rearranging the equation,

$$|T|K - |O|K = \sum_{i=1}^{|T|} d_i - \sum_{t=1}^{|O|} b_i$$

$$|T| - |O| = \frac{\sum_{i=1}^{n} d_i - \sum_{t=1}^{|O|} b_i}{K} \leq \frac{\sum_{i=1}^{n} d_i}{K}$$

$$|T| - |O| \leq \frac{\sum_{i=1}^{n} d_i}{K}$$

$$|T| - |O| \leq \frac{\sum_{i=1}^{|T|} K - T_i}{K} \leq \frac{\sum_{i=1}^{|T|} K - O_i}{K} \leq \frac{\sum_{i=1}^{|O|} K - O_i}{K} \leq |O|$$

$$|T| - |O| \leq |O|$$

$$|T| \leq 2|O|$$

As the greedy output is smaller than twice the optimal solution, the greedy algorithm is a 2 approximation. □

Problem 2

1. Find an instance where the total sum of S is less than half the total sum of a different feasible subset of A.

   Let $A = \{3, 8\}$ and $B = 10$.

   The given algorithm will produce a set $S = \{3\}$, as $3 \leq B$. Then, the algorithm will attempt to add 8 to S, but finds that the total sum is $11 > 10$. Therefore, the given algorithm will return $S = \{3\}$.

   A better total sum would be $S = \{8\}$. $8 \leq B$, and half of $S = 4$.

   Clearly, the total sum found by the given algorithm is less than half the total sum of a different feasibile solution.

2. Give a O(n log n) poly-time 2-approximation.

   Initially $S = \emptyset$
   Define $T = 0$
   Sort A in decreasing order via an arbitrary $O(nlogn)$ sorting algorithm.
   for $i = 1, 2, ...n$ **do**
       if $T + a_i \leq B$ **then**
           $S \leftarrow S \cup a_i$
           $T \leftarrow T + a_i$
       end if
   end for

   Proof of $O(nlgn)$ time.

   Proof. The algorithm is poly time. The most intensive step of the algorithm is sorting A in decreasing order. This will take $O(nlogn)$ via an efficient sorting algorithm. The rest of the algorithm takes linear time, so the algorithm is clearly poly time. □

   Proof that the algorithm is a 2-Approximation

Proof. Assume that every value in $A \leq B$.

Let O denote a subset of A that contains the optimal total sum for the problem instance. Let S_o denote the total sum of O. Consider an arbitrary A_k in the algorithm. Let S_a denote the current total sum of A at this point in the algorithm. The algorithm is attempting to add A_k into S. Because A is currently in decreasing order, $A_1 \geq A_2 \geq ... \geq A_k$.

We will break the proof up into two parts, and break those parts up into two subsequent parts, whether or not there exists an A_k such that $A_k \geq \frac{B}{2}$.

Suppose there exists an A_k such that $A_k \geq \frac{B}{2}$

In this situation, there are two different situations, either $|S| = 0 or |S| \neq 0$.

If $|S| = 0$, A_k can be added to S as $A_k \leq B$. Note that $A_k \geq \frac{B}{2}$. Consider the optimal subset of A, O. As the total sum of $O \leq B$, and $2(A_k) \geq B$ then $2(A_k) \geq S_o$. Therefore, the output of the algorithm will be at least half as large as the maximum total sum of the problem instance.

If $|S| \neq 0$, and A is sorted in decreasing order, then there must be some value S_k in S such that $S_k \geq A_k$. So, by the same logic shown in the first case, the output of the algorithm will be at least half as large as the maximum total sum of the problem instance.

Suppose there is no A_k such that $A_k \geq \frac{B}{2}$

In this situation, there are two different situations, either $\sum_{i=0}^{|A|} a_i \leq B$, or $\sum_{i=0}^{|A|} a_i > B$ If $\sum_{i=0}^{|A|} a_i \leq B$, then the output of our algorithm will also be the optimal output, as each output would use every single value in A.

Now, suppose $\sum_{i=0}^{|A|} a_i > B$. As each $a_i < \frac{B}{2}$, there must be an a_k such that the current total sum of $S \geq \frac{B}{2}$. Then, our algorithm will produce a set S that is at least $\frac{B}{2}$. Note that the optimal total sum for the problem instance, S_o, must be less than or equal to B. We know that when attempting to add A_k to S, that $S_a \geq \frac{B}{2}$. So, the algorithm has achieved a total sum which is greater than half of B, and is therefore greater than half of the optimal total sum. So, the algorithm's output will be at least half of the maximum total sum of any feasible set.

□

Problem 3

$HIT-SET-APROXIMATION\{$

create set $S = \emptyset$
create row vector D, corresponding to the decision variables for a Linear Program
create column vector V, corresponding to the right side of the constraint inequalities
create row vector C, corresponding to the objective function
create vector M, corresponding to the coefficients of the constraint inequalities
for each $a_i \in A$ **do**
  Create decision variable x_i and add x_i to D
  Add $w(a_i)$ to C
end for
for each $B_i \in B$ **do**
  Create a constraint where if $a_k \in B_i$, $M_{i,k} = 1$, else $M_{i,k} = 0$
end for
for each $D_i \in D$ **do**
  Create a constraint where $D_i \geq 0$

Problem 3 continued on next page...

Create a constraint where $-D_i \geq -1$
end for
Solve the Linear Program where C is to be minimized, D is the set of decision variables, B is the value of each constraint, and A encodes each of the constraints.
for each $d_i \in D$ **do**
if $d_i >= \frac{1}{b}$ **then**
add d_i to S
end if
end for
return S

Proof that the algorithm is poly-time

Proof. Translating the instance of HIT-SET into a linear program takes, at very worst, $O(n^2)$ time. The algorithm simply loops through the input, placing values in vectors, so the first section is poly-time.
Solving the Linear Program also takes poly-time, as there are known algorithms that solves linear programming problems in poly-time.

Clearly, the algorithm is poly-time. □

Proof that the algorithm produces a Hitting Set

Proof. Each set in B corresponds to one constraint on the linear program. The Linear Program will produce a set of decision variables, each with a value in the interval $[0, 1]$. The coefficients of each constraint correspond to whether or not a given node is contained within a set in B. The dot product of the coefficients of each constraint of the values of the decision variables will be greater than 1. As the coefficient of decision variable x_k is 0 in constraint V_m if and only if A_k is not contained in the set in B_m, then the only decision variables that can contribute to the sum are every $A_n \in B_m$. Note that when the linear program is satisfied, the dot product of the decision variables and the constraints are at least 1. As there are no more than b nodes in any subset in B, then there must be some decision variable with a value of $\frac{1}{b}$. As a node is added to the Hitting-Set if it's corresponding decision variable has a value greater than $\frac{1}{b}$, then there will be at least one node in the Hitting Set for each subset in B, as there will be some decision variable in each constraint that has a value greater than $\frac{1}{b}$.

Therefore, the algorithm produces a valid hitting set. □

Proof that the algorithm is a b-approximation

Proof. By the proof above, the algorithm will produce a hitting set. The algorithm will round up each decision variable that has a value greater than $\frac{1}{b}$. The linear program will produce values for an optimal hitting set, however with fractional values, which does not translate to actually adding a node to the hitting set. To translate the linear program into a hitting set, we round every value greater than or equal to $\frac{1}{b}$ to 1, meaning that the value from our algorithm can be up to b times the linear program's output.
Note that the Linear Program will produce a solution that has weight that is smaller or equal to the optimum weight hitting set, as it is not constrained to have a value that is either 0 or 1. However, the output of our algorithm is within b times the linear program's output. So, the weight of the output of our algorithm must be less than b times the optimum weight.

Therefore, our algorithm is a b-approximation of the weighted hitting set problem.

□