Problem Set 10 Solutions

This problem set is due **at 11:59pm** on **Wednesday, May 10, 2017.**

EXERCISES (NOT TO BE TURNED IN)

Problem 10-1. Rock Packing [60 points]

You and your friend Alice decide to take a Geology elective together. Below are the descriptions of each of your decision problems:

- Alice's Problem - BOX-PACKING(A, m) : Alice is given n rocks of weights $A = [a_1, a_2, \ldots, a_n]$ units for all i and m boxes that can support up to 1 unit of weight each. Note that each a_i is a real value between 0 and 1. Alice must decide if she can pack the rocks into the boxes such that all rocks are packed and no box is over its capacity.
- Your Problem - DESIRED-WEIGHT(C, w) : You are given n rocks of weights $C = [c_1, c_2, \ldots, c_n]$ and a target weight w. Note that in this case each c_i is a natural number. Your job is to decide if there exists a graoup $G \subseteq C$ such that the sum of the weights of rocks in G is exactly equal to w.

(a) [10 points] Show that each of the above decision problems are in **NP**.

Solution:

- BOX-PACKING - Given an assignment of rocks to boxes, adding up the weights of rocks in each box, confirming that the weight of each box is below the capacity of 1 unit, and checking that every rock was assigned can be done in $O(n + m)$ time.
- DESIRED-WEIGHT - Given a subset G of C, adding up the weights of rocks in G, confirming the total weight is equal to w, and checking that G is a valid subset can be done in $O(n)$ time.

(b) [25 points] Bob is also taking the Geology elective and has the following decision problem:

- Bob's Problem - EQUAL-WEIGHT(B) : Bob is given n rocks of weights $B = [b_1, b_2, \ldots, b_n]$ units all of which have nonnegative real values. Bob's job is decide if it is possible to divide the n rocks into two piles, B_1 and B_2, of equal weight, such that every rock is either in pile B_1 or pile B_2.

Your professor tells you that your problem, DESIRED-WEIGHT, is **NP**-complete. Both Bob and Alice have not been unable to solve their problems and would like to show that their problems are also **NP**-complete. Show that both Bob and Alice's problems are **NP**-complete.
Hint: Consider two steps. First show that you can reduce DESIRED-WEIGHT to EQUAL-WEIGHT. Then show you can reduce EQUAL-WEIGHT to BOX-PACKING.

Solution: First, we have that Bob's problem is in NP since we can construct a polynomial length and verifiable certificate. This is because given an assignment of rocks to two piles B_1 and B_2, adding up the weights of each pile, comparing that they are equal, and checking that every rock was assigned can be done in $O(n)$ time.

- First we show DESIRED-WEIGHT reduced to EQUAL-WEIGHT. Suppose we are trying to solve the DESIRED-WEIGHT problem for input C and w. Define the sum of all the rock weights in C to be s:

$$s_c = \sum_{i=1}^{n} c_i$$

Create a new set of rocks B equal to C, but with two new rocks added in of weights $s + w$ and $2s - w$:

$$B = C + [s_c + w, 2s_c - w]$$

If a subset G of C that weighs w exists, EQUAL-WEIGHT(B) will return two piles B_1 and B_2 where:

$$B_1 = G + [2s_c - w]$$

$$B_2 = C - G + [s_c + w]$$

where the weight of $B_1 = B_2 = 2s_c$.

We must also show that the opposite direction, specifically if EQUAL-WEIGHT(B) is true, then DESIRED-WEIGHT(C, w) is also true. If EQUAL-WEIGHT(B) is true, then we must have that $2s_c - w$ and $s_c + w$ must be in different piles (if they were in the same pile that pile would have total weight $3s_c$ and the remaining pile would have to have weight s_c, which is clearly not equal. Clearly, each pile must have total weight $2s_c$ since the the total weight across all elements in B is $4s_c$. The pile containing $2s_c - w$ must therefore have a subset of rocks from C whose total weight is w for that pile to have a total weight of $2s_c$. This proves that DESIRED-WEIGHT(C, w) is true if EQUAL-WEIGHT(B) is true.

- Next we show EQUAL-WEIGHT reduced to BOX-PACKING. Suppose we are trying to solve the EQUAL-WEIGHT problem for input B. Define the sum of all the rock weights in B to be s:

$$s_b = \sum_{i=1}^{n} b_i$$

Create a new set of n rocks A with weights:

$$a_i = \frac{2b_i}{s_b}$$

If an equal weight split of rocks of B is possible, then BOX-PACKING$(A, 2)$ will pack each of two boxes to full capacity exactly. The split of rocks into the two boxes corresponds to the split of rocks into equal weight piles B_1 and B_2.

We must also show that the opposite direction, specifically if BOX-PACKING$(A, 2)$ is true, then an equal weight split of rocks of B is possible. If BOX-PACKING$(A, 2)$ is true, then we must have exactly filled both boxes since the sum of all a_i is 2 and we have only 2 boxes. Therefore, there exists some set of k rocks in A whose weight sums to 1, which we call $a_{i_1}, a_{i_2}...a_{i_k}$. such that $\sum_{j=1}^{k} a_{i_j} = \frac{2\sum_{j=1}^{k} b_{i_j}}{s_b} = 1$. This means that there exists sum set of k b_i with sum $\frac{s_b}{2}$. Since the sum of all b_i is s_b, there must exist an equal weight split of rocks of B, proving the other direction.

Because both problems are in NP and there is a reduction from an NP-complete problem to both of them, both the problems are NP-complete as desired.

(c) [10 points] Consider the same DESIRED-WEIGHT problem. Since it is NP Complete, there is no known polynomial algorithm to solve the deicion problem. However, we can solve it in *pseudo-polynomial* time. In this case, the pseudo-polynomial algorithm will run fast as long as the rocks aren't too heavy. Suppose that every rock weight is bounded by k. Describe an algorithm to solve this BOUNDED-DESIRED-WEIGHT problem in time polynomial in n and k and include runtime analysis.

Solution: We solve this problem using dynamic programming. First, if $w > n \cdot k$, return $\emptyset$ since then w is too large. We define $D[i, s]$ to be equal to a subset of rock weights summing to s, only picking from the first i rocks and null if no such subset exists.
We then initialize D to be $\emptyset$ everywhere.
For $i = 1$, we have $D[1, c_1] = [c_1]$. Then, for all $i > 1$, we also have that $D[i, c_i] = [c_i]$. We also have that for all $j \in [0, w - c_i]$:

$$D[i, j + c_i] = D[i - 1, j] + [c_i] \quad \texttt{if} \quad D[i - 1, j] \neq \emptyset$$

Finally, we can also decide not to use the i^{th} rock, in which case we have the final recurrence that for all $j \in [0, w]$:

$$D[i, j] = D[i - 1, j] \quad \texttt{if} \quad D[i - 1, j] \neq \emptyset$$

In the end, we just return $D[n, w]$. Clearly, the above definition recurrences for D are valid. The algorithm has $O(n \cdot w)$ with $O(1)$ time per subproblem since each $D[i, j]$ only considers a constant number of previous values in D. This gives a total running time of $O(n \cdot w)$. We also have that $w = O(n \cdot k)$ so the total running time can be expressed as $O(n^2 \cdot k)$.

(d) [5 points] Now, consider the DESIRED-WEIGHT problem where the inputted rock weights C are sorted and has the property that each rock is greater than twice the weight of the rock before it. We call this new decision problem the SUPERINCREASING-DESIRED-WEIGHT problem.
Describe a polynomial time algorithm to solve the SUPERINCREASING-DESIRED-WEIGHT problem. Include running time analysis and brief argument for correctness.

Solution: We solve with a Greedy approach. Note that since $c_i > 2c_{i-1}$, we get $c_i > \sum_{j=1}^{i-1} c_j$.
The greedy algorithm is as follows. Maintain a running sum s of your current subset G. Iterate through rocks in order of largest to smallest from c_n to c_1. If $s + c_i < w$, add rock i to G, else move on. If $s = w$, return G. Suppose we didn't include a rock i when $s + c_i < w$. All the other rocks weights we would consider $c_{i-1}, \ldots, c_1$ sum to less than c_i, so c_i is required if w is to be reached. Since each rock is considered once, the running time is $O(n)$.

(e) [10 points]
Consider the search version of the DESIRED-WEIGHT deicion problem where you must actually find the subset G of rocks that sums to the target weight w. Let's call this the SEARCH-DESIRED-WEIGHT problem and the decision version the DECISION-DESIRED-WEIGHT problem.

Now, say you have a black-box algorithm for DECISION-DESIRED-WEIGHT that runs in $O(1)$ time. This algorithm takes in C and w and returns YES or NO depending on if a valid subset G exists. Describe a polynomial time algorithm using the DECISION-DESIRED-WEIGHT black box to solve the SEARCH-DESIRED-WEIGHT problem. Include running time analysis and brief argument for correctness.

Solution: First check DECISION-DESIRED-WEIGHT(C, w) to see if a solution exists. If one does, do the following:
Iterate through C searching for a rock until a rock i exists such that
DECISION-DESIRED-WEIGHT$(C \backslash \{c_i\}, w - c_i)$ exists. Add rock i to G and repeat this process with $(C \backslash \{c_i\}, w - c_i)$ instead of (C, w) starting from rock $i + 1$. Continue until we have $w = 0$, specifically that we have found a set of weight summing form G.

We only iterate over every rock once since a subset of the rocks is guaranteed to sum to w. Therefore, the total running time is $O(N)$.

Problem 10-2. Proving NP-Completeness [40 points]

(a) [15 points] Let TRIPLE-SAT denote the set of Boolean formulas that have at least three distinct satisfying assignments. In other words, TRIPLE-SAT is the problem: given a Boolean formula, decide whether it has at least three distinct satisfying assignments.
Prove that TRIPLE-SAT is NP-complete.

Solution: To show that TRIPLE-SAT is in NP, for any input formula ϕ, we need only guess three distinct assignments and verify that they satisfy ϕ.

To show that TRIPLE-SAT is NP-hard, we reduce SAT to it. Let ϕ denote an instance of SAT and suppose that the set of variables in ϕ are $X = \{x_1, \ldots, x_n\}$. We transform ϕ into a formula ϕ' over a new variable set X' as follows:

- $X' = \{x_1, \ldots, x_n\} \cup \{y, z\}$.
- $\phi' = \phi$.

Now we claim ϕ is satisfiable iff ϕ' has at least 3 satisfying assignments. If ϕ is satisfiable, then we can augment any particular assignment by adding any of the 4 possible pairs of values for y and z to give at least four satisfying assignments overall. On the other hand, if ϕ is unsatisfiable, then so is ϕ'.

(b) [25 points] Define BAGEL to be the decision problem where given (G, p, k), where $G = (V, E)$ is an undirected graph, p maps each vertex $u \in V$ to a nonnegative integer $p(u)$, and k is a nonnegative integer, the following holds: There exists a subset $U \subseteq V$ such that no two vertices in U are neighbors in G, and such that $\sum_{u \in U} p(u) \geq k$.
Prove that BAGEL is NP-Hard by reducing 3SAT to BAGEL. Note that 3SAT is different from TRIPLE-SAT.

Solution: Let $\phi = C_1 \wedge C_2 \wedge \ldots C_m$ be an instance of 3SAT, where each clause C_c has three literals chosen from $\{x_i, \bar{x}_i \| 1 \leq i \leq n\}$. We construct (G, p, k) as follows.

The vertices V of G are $\{v_{c,j} \| 1 \leq c \leq m, 1 \leq j \leq 3\}$, where $v_{c,j}$ corresponds to literal j in clause C_c. We label each vertex $v_{c,j}$ with x_i or $\bar{x}_i$, whichever appears in position j of clause C_c. The edges E of G are of two types:

- For each clause C_c, an edge between each pair of vertices corresponding to literals in clause C_c, that is, between v_{c,j_1} and v_{c,j_2} for $j_1 \neq j_2$.
- For each i, an edge between each pair of vertices for which one is labeled by x_i and the other by $\bar{x}_i$.

The function p maps all vertices to 1. The threshold k is equal to m. We claim that $\phi \in$ 3SAT iff $(G, p, k) \in$ BAGEL.

First, suppose that $\phi \in$ 3SAT. Then there is some truth assignment A mapping the variables to $\{\texttt{true}, \texttt{false}\}$. We know that A must make at least one literal per

clause true; for each clause, select the vertex corresponding to one such literal to be in the set U. Since there are m clauses, this yields exactly $m = k$ vertices, so the total profit is k. Moreover, we claim that U cannot contain two neighboring vertices in G: Suppose for contradiction that $u, v \in U$ and $(u, v) \in E$. Then the edge (u, v) must be of one of the two types above. But u and v cannot correspond to literals in the same clause because we selected only one vertex for each clause. And u and v cannot be labeled by x_i and $\bar{x}_i$ for the same i, because we selected only vertices labeled by variables or their negations that A makes true (and A does not make both a variable and its negation true). Since neither possibility can hold, U cannot contain two neighboring vertices. Therefore, $(G, p, k) \in$ BAGEL.

Conversely, suppose that $(G, p, k) \in$ BAGEL. Then there is some subset $U \subseteq V$, $|U| \geq m$, containing no two neighbors in G. Since U does not contain neighbors, it cannot contain two vertices from the same clause. Therefore, we must have $|U| = m$, with exactly one vertex from each clause. Also since U does not contain neighbors, U cannot contain two vertices with contradictory labels. Now define a truth assignment A for the variables: Define $A(x_i) = \texttt{true}$ if some vertex with label x_i is in U. Define $A(x_i) = \texttt{false}$ if some vertex with label $\bar{x}_i$ is in U. For other variables the truth value is arbitrary. Since U does not contain two vertices with contradictory labels, assignment A is well-defined. Also, assignment A satisfies all clauses, since each clause contains a literal corresponding to a vertex in U and the definition of A ensures that A makes this literal true. Therefore, ϕ is satisfiable.

Since we can reduce 3SAT to BAGEL, we must have that BAGEL is NP-hard as desired.