Exercises: Knapsack Problem

To be returned for April 16th 2019.

The Simple Knapsack problem takes a set of integers \( S = \{ w_1, \cdots, w_n \} \) and an integer \( b \) as inputs. The objective is to compute a subset \( T \subseteq \{ 1, \cdots, n \} \) of items such that \( \sum_{i \in T} w_i \leq b \) and \( \sum_{i \in T} w_i \) is maximum. That is, we want to fill our knapsack without exceeding its capacity \( b \) and putting the maximum total weight in it.

1 Exact Algorithm via dynamic programming

Dynamic programming is a generic algorithmic method that consists in solving a problem by combining the solutions of sub-problems.

As an example, the Simple Knapsack Problem consists in computing an optimal solution for an instance \( S = \{ w_1, \cdots, w_n \} \) and an integer \( b \). Let \( \text{OPT}(S, b) \) denote such a solution. We will compute it using solutions for sub-problems with inputs \( S_i = \{ w_1, \cdots, w_i \} \) and \( b' \in \mathbb{N} \), for any \( i \leq n \) and \( b' < b \). That is, we will compute \( \text{OPT}(S, b) \) from all solutions \( \text{OPT}(S_i, b') \) for \( i \leq n \) and \( b' < b \).

**Algorithm 1** Dynamic programming algorithm for Simple Knapsack

```
 Require: A set of integers \( S = \{ w_1, \cdots, w_n \} \) and \( b \in \mathbb{N} \).
 Ensure: A subset \( \text{OPT} \subseteq \{ 1, \cdots, n \} \) of items such that \( \sum_{i \in T} w_i \leq b \)

1: For any \( 0 \leq i \leq n \) and any \( 0 \leq b' \leq b \), let \( \text{OPT}[i,b'] = \emptyset \);
2: For any \( 0 \leq i \leq n \) and any \( 0 \leq b' \leq b \), let \( \text{opt}_\text{cost}[i,b'] = 0 \);
3: for \( i = 1 \) to \( n \) do
4:   for \( b' = 1 \) to \( b \) do
5:     if \( w_i \leq b' \) and \( \text{opt}_\text{cost}[i-1,b'-w_i] + w_i > \text{opt}_\text{cost}[i-1,b'] \) then
6:       \( \text{OPT}[i,b'] = \text{OPT}[i-1,b'-w_i] \cup \{ i \} \)
7:     \( \text{opt}_\text{cost}[i,b'] = \text{opt}_\text{cost}[i-1,b'-w_i] + w_i \)
8:   else
9:     \( \text{OPT}[i,b'] = \text{OPT}[i-1,b'] \)
10:    \( \text{opt}_\text{cost}[i,b'] = \text{opt}_\text{cost}[i-1,b'] \)
11: return \( \text{OPT} = \text{OPT}[n,b] \)
```

**Question 1** Prove that Algorithm 1 has time-complexity \( O(n \cdot b) \).

**Question 2** Explain that we may assume that \( \max_i w_i \leq b \) and \( b \leq \sum_i w_i \) since, otherwise, the instance may be simplified.

Proof that, if \( \max_i w_i \leq b \leq \sum_i w_i \), Algorithm 1 proceed in polynomial-time if \( \max_i w_i \) is polynomial in \( n \) but exponential if \( \max_i w_i \) is exponential in \( n \).

Actually, the Knapsack Problem is an example of Weakly NP-hard (roughly, it can be solved in polynomial-time if the weights are polynomial).

**Question 3** Prove by induction on \( i \) and \( b' \) that the solution \( \text{OPT} = \text{OPT}[n,b] \) returned by Algorithm 1 is optimal.
2 Approximation Algorithm and PTAS

Algorithm 2 Greedy algorithm for Simple Knapsack

Require: A set of integers $S = \{w_1, \cdots, w_n\}$ and $b \in \mathbb{N}$.
Ensure: A subset $T \subseteq \{1, \cdots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$

1: $T = \emptyset$
2: total_weight = 0
3: Sort $S$. Let us assume that $w_1 \geq w_2 \geq \cdots \geq w_n$.
4: for $i = 1$ to $n$ do
5: if total_weight + $w_i \leq b$ then
6: Add $i$ to $T$
7: Add $w_i$ to total_weight
8: return $T$

Question 4 What is the time-complexity of Algorithm 2?

Question 5 Prove that Algorithm 2 is a $2$-approximation algorithm for the Simple Knapsack problem.

hint: let $T$ be the computed solution and assume it is not optimal. Let $j \geq 1$ be the smallest integer such that $j + 1$ is NOT in $T$. Show that $w_{j+1} \leq \sum_{i \leq j} w_i/j$ and that

$$\sum_{i \leq j} w_i \leq \sum_{i \in T} w_i \leq OPT < \sum_{i \leq j+1} w_i$$

A polynomial-time approximation scheme (PTAS) is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time in the size of the instance (not necessarily in $\epsilon$), produces a solution that is within a factor $1 + \epsilon$ of being optimal.

That is, when $\epsilon$ tends to 0, the solution tends to an optimal one, while the complexity increases (generally, the complexity is of the form $O(n^{f(1/\epsilon)})$ for some function $f$).

Question 6 Prove that Algorithm 3 has time-complexity $O(n^{\lceil 1/\epsilon \rceil + 1})$.

hint: prove that there are $O(n^k)$ subsets of size at most $k$ in a ground-set with $n$ elements.

Question 7 Prove that Algorithm 3 is a $(1+\epsilon)$-approximation algorithm for the Simple Knapsack problem.

hint: Consider an optimal solution $M$ and let $X^* = \{i_1, \cdots, i_k\}$ be the $k$ items with largest weight in $M$. Consider the iteration of Algorithm 3 when it considers $X^*$.

Actually, we can do better. Indeed, the Knapsack Problem admits a fully polynomial-time approximation scheme (FPTAS) algorithm, that is an algorithm that computes a solution that is within a factor $1 + \epsilon$ of being optimal in time polynomial both in the size of the instance AND in $1/\epsilon$. 

2
Algorithm 3 PTAS for Simple Knapsack

Require: A set of integers $S = \{w_1, \ldots, w_n\}$, $b \in \mathbb{N}$ and a real $\epsilon > 0$.

Ensure: A subset $T \subseteq \{1, \ldots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$

1: $best = \emptyset$
2: $best\_cost = 0$
3: $k = \lceil 1/\epsilon \rceil$
4: for Any subset $X \subseteq S$ of size $k$ do
5: \hspace{1cm} Complete $X$ using the Greedy Algorithm. That is :
6: \hspace{1.5cm} $T = X$
7: \hspace{1.5cm} $total\_weight = \sum_{i \in X} w_i$
8: \hspace{1.5cm} Sort $S \setminus X$. Let us assume that $S \setminus X = \{w_1, \ldots, w_{n-k}\}$ and $w_1 \geq w_2 \geq \cdots \geq w_{n-k}$.
9: \hspace{1.5cm} for $i = 1$ to $n-k$ do
10: \hspace{2.5cm} if $total\_weight + w_i \leq b$ then
11: \hspace{3.5cm} Add $i$ to $T$
12: \hspace{3.5cm} Add $w_i$ to $total\_weight$
13: \hspace{1.5cm} if $total\_weight > best\_cost$ then
14: \hspace{2.5cm} Replace $best$ by $T$
15: \hspace{1.5cm} end if
16: \hspace{1.5cm} end if
17: \hspace{1.5cm} end for
18: end for
19: return $T$