COMP 182: Algorithmic Thinking
The Knapsack Problem and Greedy Algorithms

Luay Nakhleh

The Knapsack Problem is a central optimization problem in the study of computational complexity. We are presented with a set of n items, each having a value and weight, and we seek to take as many items as possible to maximize the total value, but with respect to a constraint that the total weight cannot exceed a pre-defined maximum weight. More formally, the Knapsack Problem is defined as:

> **Input:** Set S of n items $x_1, x_2, \ldots, x_n$, where items x_i has weight w_i and value v_i, for $1 \le i \le n$, and a number W.
>
> **Output:** Subset $S' \subseteq S$ such that (1) $\sum_{x_i \in S'} w_i \le W$ and (2) $\sum_{x_i \in S'} v_i$ is maximum over all subsets that satisfy (1).

Condition (1) in the problem statement concerns feasibility, while Condition (2) concerns optimality over all feasible solutions.

Unfortunately, there is no known polynomial-time algorithm for the Knapsack Problem (it is NP-hard).

However, if we make a seemingly simple relaxation to one of the problem's constraints, we obtain a problem that has a polynomial time solution. Notice that in the Knapsack Problem, each item x_i is either not taken to be part of the solution, in which case it contributes 0 to the total weight and total value, or it is taken in its entirety, in which case it contributes w_i to the total weight and v_i to the total value. But what if we can take fractions of items? That is, what if are allowed to take 0.5 amount of item x_i and 0.37 amount of item x_j, and so on? This is known as the Fractional Knapsack Problem, and defined as follows.

> **Input:** Set S of n items $x_1, x_2, \ldots, x_n$, where items x_i has weight w_i and per-unit value v_i, for $1 \le i \le n$, and a number W.
>
> **Output:** $y_1, y_2, \ldots, y_n$ such that (1) for $1 \le i \le n$, $0 \le y_i \le w_i$, (2) $\sum_{1 \le i \le n} y_i \le W$ and (3) $\sum_{1 \le i \le n}(y_i \cdot v_i)$ is maximum over all subsets that satisfy (1) and (2).

Please note here that v_i is now the per-unit value. This is why the problem seeks to maximize the sum of $y_i \cdot v_i$ terms (for example, if $y_3 = 2.4$, this means the solution includes 2.4 units of item 3, contribution a value of $2.4 \cdot v_3$).

This simple relaxation (from either taking the item in its entirety or not taking it at all to taking a fraction of it) turns the problem into a polynomially solvable one by a greedy algorithm as follows:

```
1. Sort the items in non-increasing order of their per-unit value $v_i$. Let the sorted list be $x_1, x_2, \ldots, x_n$;
2. Initialize $W'$—the total weight of the items included so far—to 0;
3. For $i \leftarrow 1$ to $n$
   (a) If $W' = W$
       i. $y_i \leftarrow 0$;                 // item $i$ cannot be taken
       ii. $W' \leftarrow W' + y_i$;
   (b) Else
       i. If $(W - W') \ge w_i$
          A. $y_i \leftarrow w_i$;           // all of item $i$ is taken
          B. $W' \leftarrow W' + y_i$;
       ii. Else
          A. $y_i \leftarrow W - W'$;    // a fraction of item $i$ is taken
          B. $W' \leftarrow W' + y_i$;
```

This is an $O(n \log n)$ greedy algorithm. We now prove that it is correct; that is, that the algorithm above yields an optimal solution to the Fractional Knapsack Problem.

Proof: Assume the items sorted in non-increasing per-unit values are $x_1, x_2, \ldots, x_n$, and let $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be the solution computed by the algorithm. Consider now an optimal solution to the problem (not necessarily computed by the algorithm above): $O = \langle o_1, o_2, \ldots, o_n \rangle$, where o_i is the weight units of item i according to this optimal solution. The proof will now proceed as follows: If $y_i = o_i$ for every $1 \leq i \leq n$, then we are done, since the solution computed by the algorithm is optimal. If there exists at least one $1 \leq j \leq n$ such that $y_j \neq o_j$, we will then show that we can convert the optimal solution O into Y without affecting the total value, thus establishing that Y is also an optimal solution.

Assume without loss of generality that the knapsack can be filled with the available items (that is, a solution to the problem has a total weight the equals W; otherwise, we can "adjust" the value of W by subtracting from it the total weight of all elements). We know that

$$\sum_{i=1}^{n} o_i = W$$

since the o_i's are an optimal solution, and

$$\sum_{i=1}^{n} y_i = W$$

by construction of the algorithm (that algorithm fills the knapsack). If $y_i = o_i$ for every $1 \leq i \leq n$, then the solution computed by the algorithm is optimal, and the proof is established. Now, assume that the solution computed by the algorithm differs from the optimal solution we are considering, and let j be the smallest index ($1 \leq j \leq n$) such that $y_j \neq o_j$. Since the algorithm is designed to take for every item the maximum amount possible, it follows that $y_j > o_j$. Let $d = y_j - o_j$. Consider the solution $Q = \langle q_1, q_2, \ldots, q_n \rangle$ that we construct as follows:

- for every $1 \leq i < j$: $q_i = o_i$ (also $q_i = y_i$);
- $q_j = y_j$;
- For $i = j + 1$ to n
  - $d' \leftarrow \min(q_i, d)$;
  - $q_i \leftarrow q_i - d'$;
  - $d \leftarrow d - d'$;

While this might look complicated, the idea of constructing Q is simple: leave all elements up to the $j-1$-st item the same as those in O, replace the j-th item by y_j, and then decrease from the total weight of the remaining $n - j$ items the weight d that was added to item j in Q.

After this procedure is done, the total value of the solution Q is equal to the total value of O (make sure you understand why!). Therefore, solution Q' is also optimal, and it now agrees with the solution Y computed by the algorithm all the way to at least index j. If $Q = Y$, then we are done. If $Q \neq Y$, then we repeat the same process as above, but with Q taking the role of O (that is, find the smallest j such that $q_j \neq y_j$, and do the procedure above). After doing this, we obtain a solution that is optimal and identical to Y. Therefore, Y is optimal. $\square$

Think! The Knapsack Problem does not have a polynomial-time greedy algorithm (we stated above that it is NP-hard). But suppose you were not convinced and wanted to prove, similar to the proof above, that a greedy algorithm (e.g., take items in non-increasing order of their values) would solve the problem. Where does the proof break down?