CPSC 320 2023S2: Tutorial 5 Solutions

1 Knapsack with Structured Weights: Part 1

The *Knapsack Problem* is a classic optimization problem in computer science. The most common version is as follows: consider a set of n items, numbered $1, ..., n$. Each item i has a positive weight $w_i > 0$ and a positive value $v_i > 0$. Furthermore, we're given an overall weight capacity C. The problem is to select a subset $S \subseteq \{1, ..., n\}$ that maximizes the total value of S, but keeps the total weight below C. Formally, we want to maximize $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq C$.

The knapsack problem is known to be hard: there are no algorithms known that find the optimal solution to all instances in better than worst-case exponential time (we'll study this concept more at the end of the course, when we consider NP-completeness).

However, for this tutorial question, we'll consider a variation of the problem. In particular, we'll set the capacity as $C = 1$, and restrict that all weights are either 1/2, 1/4, 1/8, or 1/16. An intuitive greedy approach you might think of to try first is by sorting the items in decreasing order of v_i/w_i, which is the "value density" of each item. Then, go down the list in decreasing order, selecting as many items as fit.

1. Give a small instance of the problem where this greedy algorithm achieves an optimal solution.

   SOLUTION:

   There are plenty of solutions, including really trivial easy ones like when there are no items, so there is one trivial solution consisting of no items. This is obviously optimal, as well as what is returned by the greedy algorithm, but it doesn't tell us too much about the algorithm.

   Now, suppose we have 4 items:

Item:	1	2	3	4
Value:	3	4	2	2
Weight:	1/2	1/2	1/4	1/4

   This greedy algorithm will pick items 2, 3, and 4 (all with value density 8), for a total value of 8 and a total weight of 1, which is optimal.

2. Give a small instance of the problem where this "value density" greedy algorithm does **not** give an optimal solution. What goes wrong with this greedy approach?

   SOLUTION:

   Again, there are tons of solutions. Suppose we have 3 items:

Item:	1	2	3
Value:	2	2	1
Weight:	1/2	1/2	1/16

Item 3 has the biggest value density (16), so the algorithm picks that first. Then, there's only room left for one of item 1 or 2, so the total value is 3. However, an optimal solution would avoids the high value density Item 3 and instead pick items 1 and 2, for a total value of 4.

We can see that what goes wrong with the value density approach is that getting a small, dense item can prevent us from fitting in a larger-valued (but less dense) item.

Again, the original Knapsack Problem is a "hard" problem and an efficient algorithm may not exist. However, thanks to the weight restrictions, there *is* an efficient greedy algorithm that gives an optimal solution to this version of the problem! We'll derive this algorithm and prove that it works.

To avoid special cases in our proofs, assume that we always have an unlimited supply of extra items whose value is 0 in every weight class.

3. Prove that there is always an optimal solution that uses an **even** number of 1/16-weight items.

   SOLUTION: Suppose not, such that you have an optimal solution that uses an odd number of 1/16-weight items. If you add up the weights of all items in the solution, and express the result as a fraction with denominator 16, note that the numerator must be odd. This is because for all objects that are not 1/16-weight, they will contribute 8/16, 4/16, or 2/16 to the sum, and we have an odd number of 1/16-weight items.

   Therefore, since $C = 1 = 16/16$, we can always add in (at least) one more 1/16-weight item, making an optimal solution using an even number of 1/16-weight items. This is where it's handy to always have an unlimited supply of 0-value items, in case the original optimal solution "used" up all of the original 1/16-weight items).

4. Prove that if an optimal solution uses c items of weight 1/16, it uses c of the highest value 1/16-weight objects.

   SOLUTION: We'll use an exchange argument. Suppose that solution S (a subset of $\{1, 2, ..., n\}$) uses c items with weight 1/16, but not the c highest-value 1/16-weight items. Then we can always swap in a higher-value 1/16-weight item, to get a better solution, so S could not have been optimal.

The preceding two parts suggest how a greedy algorithm might work. Suppose that we sort the 1/16-weight items in decreasing order of value, and then pair up the items in sequence (i.e. the 1st and 2nd items form a pair, then the 3rd and 4th items, then the 5th and 6th, etc.). The previous two parts imply that there is an optimal solution that uses the c highest-valued pairs, for some c. We'll continue this next time, but meanwhile think about how you might leverage this to get a greedy algorithm.

2 Knapsack with Structured Weights: Part 2

To be continued next time...