Knapsack Problems

The 0/1-Knapsack Problem is defined as follows: given a container of capacity k (we will assume k is a limit on the total mass that can be placed in the container – but it could just as easily be a measure of volume) and a set A of items $\{a_1, \ldots a_n\}$, each of which has mass m_i and value v_i, we call a subset $S \subseteq A$ ***feasible*** if $\sum_{a_i \in S} m_i \leq k$. Our goal is to find a feasible subset S* that maximizes $\sum_{a_i \in S*} v_i$. In other words, we want to find the most valuable combination of items that will fit in the container.

This is a profoundly practical question – NASA has to solve it every time they put together a load of supplies and equipment for launch to the ISS. Not only that but every one of us solves something similar on a daily basis. Given that there are only 24 hours in the day and a huge collection of activities competing for our time (each activity requiring a certain number of hours for completion), we decide on a subset of activities to fill our day. The question is, of course, how do we decide on the values? I suspect that the better we are at solving this problem – consciously or subconsciously – the better we feel at the end of each day.

The bad news is that unless P = NP, there is no polynomial-time algorithm to solve the 0/1-Knapsack Problem ... which means in practical terms that we will never find a Greedy Algorithm that is guaranteed to find the optimal solution to all instances of this problem. (See below for a quick proof of the difficulty of the 0/1-Knapsack Problem.)

So what is this uber-difficult problem doing in our discussion of Greedy Algorithms, except making us feel depressed about the complexities of daily life? Well, as with a lot of difficult problems, if we modify the problem we can find a version that is solvable.

Here we're going to do something a bit counter-intuitive. We are going to change the problem so that the set of feasible solutions becomes infinitely large ... surprisingly, this makes finding an optimal solution easy!

Greedy Algorithm for Fractional Knapsack Problem

The Fractional Knapsack Problem is defined as follows: given a container of capacity k and a set A of items $\{a_1, \ldots a_n\}$, each of which has mass m_i and value v_i, find the most valuable combination of objects that will fit in the container, **allowing fractions of objects to be used,**

where the value of a fraction of an object is the same fraction of the value of the object.

More formally: Given k and a set of n pairs of the form (v_i, m_i) find a set of values $\{ p_1, p_2, \ldots p_n\}$ such that $0 \leq p_i \leq 1 \quad \forall i$ and $\sum_{i=1}^{n}(p_i \cdot m_i) \leq k$ and $\sum_{i=1}^{n}(p_i \cdot v_i)$ is maximized

In this formulation, the p_i values are the fractions.

Our goal is to find a greedy algorithm to solve this problem. As with all greedy algorithms, our first task is to sort the objects. We can experiment with a variety of sorting criteria but the one that leads to success in this case is to sort the objects in decreasing $\frac{v_i}{m_i}$ order.

```
Greedy FKS:

     Sort the objects  in decreasing  $\frac{v_i}{m_i}$  order

     While k > 0 and there are still objects to consider:
          Take as much of the next item as possible
          Reduce k by the mass amount just added to the knapsack
```

That's about as simple as an algorithm can get, and it clearly runs in O(n log n) time (the sort is the longest part of the algorithm).

Proof of Correctness:

In class I gave a simple outline of the standard proof by induction, the details of which are so similar to the previous algorithms that there is little point reproducing it here.

Instead, we will look at a different method for proving correctness of Greedy Algorithms: a technique we can call "eliminate the differences".

WLOG we will assume that object o_1 has the highest $\frac{v_i}{m_i}$ ratio, object o_2 has the second highest, etc. ... in other words, we have performed the sort and renumbered the objects.

Let S_A be the algorithm's solution, and let O be an optimal solution. If S_A and O are identical, then S_A is optimal.

Suppose S_A and O are not identical. In fact, suppose S_A contains more of o_1 than O does. That means O contains some other objects in at least the same amount as the amount of o_1 that O leaves out. But since $\frac{v_1}{m_1}$ is $\geq$ all the other $\frac{v_i}{m_i}$ ratios, we can replace some of the "other stuff" in O with the left-out o_1, and **the total value cannot decrease**. Call this new optimal solution O′. This means that S_A and O′ have one fewer difference than S_A and O did.

We can continue eliminating differences in this way until we reach a point where $S_A = O^{\cdots}$ (we don't know how many differences there were to start with, so I used "..." for the superscript after they have all been eliminated). We know there can't be more than n differences because there are only n objects. Since every time we eliminate a difference we get a new optimal solution, we eventually show that S_A is equal to an optimal solution ... so S_A is optimal.

As always, remember that every time we say "Let O be an optimal solution" we are not saying "Suppose we have found an optimal solution O". The "Let O be an optimal solution" is part of the argument we use to prove the optimality of S_A. At no point does the **algorithm** try to transform one solution into another.

Example of the algorithm in action:

k = 15

v_i	15	14	30	100	2	100000
m_i	6	7	20	80	2	100001

$\dfrac{v_i}{m_i}$	2.5	2	1.5	1.25	1	$\dfrac{100000}{100001}$

Ooooh, look at that last object – it's worth 1000000 ! Surely we should take 15 units of that! Well no, it's easy to see that if we take *any* of that object we could improve our value by removing that and replacing it with any of the other objects.

The algorithm takes all of o_1, all of o_2, and 2 of the available 20 units of o_3 with a total value of $15 + 14 + \dfrac{2}{20} \cdot 30$ = 32

By the proof given above, there is no solution with a value > 32

Proof that 0/1-Knapsack Problem is Very Difficult

0/1-Knapsack is not an NP-Complete problem ... not because it is solvable in polynomial time, but because it is not a decision problem. However we can easy make it into one.

0/1KS is defined as (copied from above): given a container of capacity k (we will assume k is a limit on the total mass that can be placed in the container – but it could just as easily be a measure of volume) and a set A of items $\{a_1, \ldots a_n\}$, each of which has mass m_i and value v_i, we call a subset $S \subseteq A$ ***feasible*** if $\sum_{a_i \in S} m_i \leq k$. Our goal is to find a feasible subset S* that maximizes $\sum_{a_i \in S*} v_i$. In other words, we want to find the most valuable combination of items that will fit in the container.

To convert this into a decision problem (which we will call 01KSD) all we need to do is fix a target value T. Everything in the problem definition remains the same except the sentence " Our goal is to find a feasible subset S* that maximizes $\sum_{a_i \in S*} v_i$ " becomes

"Does A have a feasible subset S* such that $\sum_{a_i \in S*} v_i \geq$ T ?"

Note 01KSD trivially reduces to 01KS.

01KSD is easily shown to be NP-Complete, by reducing Subset-Sum to 01KSD :

Let S, m be an instance of Subset-Sum. (S is the set $\{s_1, \ldots s_n\}$, m is the target value.)

Construct an instance of 01KSD with $m_i = s_i$ and $v_i = s_i \quad \forall \quad i$, k = m, T = m

Here $\frac{v_i}{m_i} = 1 \;\; \forall \; i$. It is easy to see that the answer to this instance of 01KSD is "Yes" if and only if the answer to the original instance of Subset-Sum is "Yes".

So if 01KS could be solved in polynomial time, then so could 01KSD ... which would mean P = NP. We conclude that 01KS cannot be solved in polynomial time.