COMP3010 Week 13 Submission

Ameer Karas

44948956

A - Sort the items according to their value, then add items to the set S_0 starting from the highest value item as long as adding the item does not exceed the weight limit. Continue until we cannot add any more item.

B - Sort the items according to their value density, i.e. v_i/w_i, then add items to the set S_0 as before starting from the highest value item until we cannot add any more item.

Inputs:

- Set S of n items, with numbers from 1 to n representing the items
- The weight of each item, w_i
  - Say set Wt holds the weights for each item
- The value of each item, v_i
  - Say the set V holds the values for each item
- Maximum capacity, W

a) Assume:
W = 8

S	**1**	**2**	**3**	**4**	**5**
Wt	2	3	4	7	10
V	8	10	12	15	30
V/Wt	4	3.33	3	2.5	3

Algorithm A would choose item 4 with a weight of 7 and a value of 15. This is not optimal, as a better solution would be to choose item 2 and item 3 for a total value of 22.

b) Assume:
W = 10
S = {1, 2, 3, 4, 5}
Wt = {1, 3, 7, 8, 4}
V = {1, 8, 20 ,24, 4}
V/Wt = {1, 2.66, 2.86, 3, 1}

Algorithm B would choose item 4 and item 1 for a total value of 24 + 1 = 25. The optimal solution is to choose item 3 and item 2 for a total value of 28.

c) Because in the week 12 lecture slide it says, and I quote, “in this unit, we will consider only approximation algorithms… that runs in polynomial time”. 😉

Nah but seriously, for both algorithms we have to sort the items (let’s assume merge sort for O(nlog(n)), and then iterate with a loop:

- With A, this loop iterates while added items don’t exceed the weight limit
- With B, this loop iterates until no more items can be added to S’

Using the example input from part (a) for both, we’d have:

S	1	2	3	4	5
Wt	2	3	4	7	10
V	8	10	12	15	30
V/Wt	4	3.33	3	2.5	3

$V_a = 15$; algorithm adds to the set S_0 starting from highest value.
In this example, there are 2 comparisons: one at item 5 ($W >= w_5$) and one at item 4.
Therefore, this algorithm completes in 2 iterations.

Sorted for B, we have:

S	4	3	5	2	1
Wt	7	4	10	3	2
V	15	12	30	10	8
V/Wt	2.5	3	3	3.33	4

$V_b = \{1, 2\} = 10 + 8 = 18$
Similarly to algorithm A, there are two iterations here.

There is one more comparison to decide which of V_b and V_a is larger, for a total of 5 computations. Assuming that we're calculating Big O, the merge sort is dominant so this algorithm runs in polynomial time.

d) In our example in part (c), v_k would be item 3 (ignoring item 5 which would not fit regardless due to the choice of W) with a value of 12. This item's value is $12 <= V_a = 15$ because with algorithm A, we are starting with the highest value item that can be added to the knapsack and then moving down the line (while there is available weight); V_a will return the most valuable items while there is remaining weight. These items/the most valuable available item will be more valuable than v_k.

e) In part (a), we've defined V* as 22. Using the previous parts, we can define v_k as item 3 with a value of 12, and we can define V_k as items 1 and 2 with a value of 18. With items 1 and 2, we have a remaining weight of 3. To take a fraction of v_k to remain less than or equal to W = 8 would be to take ¾ of v_k. This would result in a total value of 18 + 9 = 27, which is greater than V*.

f) As in part (e), $V_b + v_k = 27 > V^* = 22$.

g) From the previous parts, $V_a + V_b = 33 > V^* = 22$.

h) To be a 2-approximation algorithm, the algorithm mustn't produce an algorithm worse than twice the optimal solution. As this approximation algorithm produce a result better than 22/2 = 11, it is a 2-approximation algorithm.