Lab 2 – The grand solving of the knapsacks of the Knapsacksolver 3000

Group: 1, Andreasson Fredrik, Rockström Truls

INTRODUCTION

The multiple knapsack problem is a simple problem that deals with filling some number of sacks with the highest value of items taking their weight into account. We used two algorithms for this. Our first one was the greedy algorithm that just fills the knapsacks to the brim with the highest comparative value items with no respect to the weight until the sack is full. Our other algorithm was the neighbourhood search which used the greedy algorithm as a base and then optimized the space in the knapsacks to get more value.

DESIGN AND IMPLEMENTATION

Our greedy algorithm was based on the description in the document where it starts off by creating random items and then sorting the list. The list is sorted based on the comparative value of the items. The greedy algorithm starts off by filling one knapsack with the “best” items until it is full and then it moves on to the next knapsack and repeats the same behaviour. This just fills the knapsacks with the best items with no regard for the remaining size in the other knapsacks.

Our neighbourhood definition is defined as after a knapsack has switched an item with another knapsack that “state” is defined as a neighbour. If that neighbour has a smaller remaining capacity it is a better state for the knapsacks because there is then potential to add another item to another knapsack in turn increasing the total value of all knapsacks.

The algorithm terminates when no items can be added from the list and no productive swaps can be made.

A pseudo code implementation of the greedy algorithm is as follows:

1. Create items
2. Create knapsacks
3. Sort items based on comparative value
4. Fill all knapsacks with items going from best comparative value to the worst.

Pseudo code for the improving search is as follows:

1. Fill bags with the greedy algorithm.
2. Swap items between bags to open up space from first to last.
3. If something was changed execute from step 1 again

CONCLUSION AND FUTURE WORK

Our algorithms did a really good job of filling the knapsacks with the weight being extremely close to the capacity limit. From our tests with our manually defined items and knapsacks the algorithm always found the best solution in the test we have displayed here in the document [1] there is another “optimal” solution where Item 9 could take the place of Item 8 and Item 10 and thereby saving 1 weight and remaining at the same value but this weight change doesn't count for anything to our algorithm so the change does not need to be made.

I believe that the neighborhood search could have been changed by trying to minimize the weight of the knapsack while maintaining the highest value possible. There could be cases where that saved weight introduces another item that wouldn't otherwise fit into the knapsack.

Test data

The following is the output to the console from the algorithm.

Item number 0 Value 6, Weight 1, Comparative Value 6

Item number 1 Value 5, Weight 2, Comparative Value 2.5

Item number 2 Value 2, Weight 1, Comparative Value 2

Item number 3 Value 4, Weight 2, Comparative Value 2

Item number 4 Value 6, Weight 4, Comparative Value 1.5

Item number 5 Value 3, Weight 2, Comparative Value 1.5

Item number 6 Value 4, Weight 3, Comparative Value 1.33

Item number 7 Value 5, Weight 4, Comparative Value 1.25

Item number 8 Value 5, Weight 4, Comparative Value 1.25

Item number 9 Value 6, Weight 5, Comparative Value 1.2

Item number 10 Value 2, Weight 2, Comparative Value 1

Item number 11 Value 1, Weight 2, Comparative Value 0.5

Item number 12 Value 2, Weight 5, Comparative Value 0.4

Filling knapsacks

Contents of the knapsacks:

Knapsack 0 The knapsack carries 7 items weighing 15, a total capacity of 15 and with a total value of 30

Knapsack 1 The knapsack carries 4 items weighing 12, a total capacity of 12 and with a total value of 13

Total value 43, total weight 27, comparative value 1.59

Remaining items

Item number 0 Value 6, Weight 5, Comparative Value 1.2

Item number 1 Value 2, Weight 5, Comparative Value 0.4