Lab 2 – In search of the best knapsacks

Group: 4, “Forsberg, Linus”, “Falkenström, Maximilian”

INTRODUCTION

In this assignment we were tasked with implementing algorithms for finding solutions to the multiple knapsack problem. The algorithms implemented were a basic greedy algorithm and neighborhood search. We built a system of knapsacks and items including a test program to run the algorithms. The platform used for this assignment was IntelliJ IDEA by JetBrains.

DESIGN AND IMPLEMENTATION

Greedy

First we sort the items according to their value by weight (value / weight). We then iterate through all the items that should be placed in the bags. For each item we then iterate thru all the bags to find the best suitable bag to put the item in. We find the most suitable bag by first checking if the capacity of the bag is greater than the weight of the item and then checking which bag has the least space left.

The result was as expected as it did not give an optimal solution. There was a bit of space left in all of the bags but it gave us a good starting point for the neighborhood search. We also tried to place the items in randomly picked bags but decided to go with the first solution.

Pseudo code

```
sort items from high relative value to low relative value
for each (item in items)
    selectedBag = null
    for each (bag in bags)
        if ((bagCapacity > itemWeight) &&
                (selectedBag == null || selectedBagCapacity > bagCapacity))
            selectedBag = bag
    if (selectedBag != null)
        add item to selectedBag
    else
        add item to list of unused items
```

Neighborhood

For the neighborhood search algorithm we started by using the greedy algorithm we had created to get a fairly good solution to start out with. Once we had this solution we sorted all the bags by capacity available and picked the one with the most capacity available. We then iterated through the other bags, starting with the one with the second most capacity available and tried to move as big of an item as possible between the two bags, in order to maximize the available capacity in one of them. At that point, we could simply iterate through the list of unused items, sorted by their relative value (value by weight), and add them to the bag with the most capacity as long as there was enough capacity in the bag to hold them.

It worked out fairly well, all bags were at full capacity or close once the neighborhood search was finished in the tests that we have run. We defined the neigborhood as the bags and the items in them. The unused items could increase the local optima of the neighborhood by joining one of the bags. The termination criteria we chose was to simply cycle the algorithm 100 times. The number was picked fairly randomly, it showed to run fast enough for our tests, while making sure to find a good solution.

Pseudo code

```
Sort unused items
Loop 100 times:
   if (unused_items_list is empty):
      break
   sort bags by available capacity
   bigBag = bag with most capacity

   for (i = bags.size - 2; i >= 0; i--):
      secondBag = bags.get(i)

   for (j = 0; j < secondBag.availableCapacity; j++):
      if (bigBag.findItem(weight)):
         bigBag.remove(item)
         secondBag.add(item)
         break

      if (secondBag.findItem(weight)):
```

```
        secondBag.remove(item)
        bigBag.remove(item)
        break

    if (items were moved):
        for (item in unusedItems):
            if (item.weight <= bag.availableCapacity):
                unusedItems.remove(item)
                bag.add(item)
```

CONCLUSION AND FUTURE WORK

We built a simple test program consisting of two bags and a few more items, where we could manually find an optimal solution. Following this we implemented the greedy algorithm and compared the results. Then we implemented the neighborhood search algorithm and compared the results from this one with the ones from the greedy to make sure it was better and with our manual solution to make sure it was good enough. Once satisfied with the result, we increased the number of bags and items a bit to see how the results came out between the greedy and the neighborhood search algorithms. The neighborhood search got close enough to max capacity that we thought it was good enough.

Looking back, we could have implemented a more advanced algorithm that took into account an item from two or three different bags at the same time and switched or shuffled these to optimize the available capacity of a bag in order to fit more items from the list of unused items.

We have expanded our knowledge of greedy algorithms, neigborhood search and found new ways of thinking about problems and solutions. One thing we found out as we were discussing the problem together is the necessity of drawing things out on paper as these things are difficult to visualize or put into words.

Number of words: 750