NI-KOP Homework 1

Patrik Jantošovič

October 11, 2020

1 The Decision Version Of Knapsack Problem

The given is integer n as number of items, integer M as the capacity of the knapsack, integer B as minimal price of the knapsack and two finite sets W and C. W=$\{w_1, .., w_n\}$ being the set of weights of items and C=$\{c_1, .., c_n\}$ being the set of costs of items. The task is then to decide if it is possible to construct a finite set X=$\{x_1, ..., x_n\}$ where each x_i is either 0 or 1 so that:

$$w_1x_1 + w_2x_2 + ... + w_nx_n <= M \tag{1}$$

$$c_1x_1 + c_2x_2 + ... + c_nx_n >= B \tag{2}$$

2 Implementation

JAVA implementation was created to solve this problem and cross-checked with provided solutions. I managed to implement both a decision and constructive algorithm, difference being in stopping the algorithm once we decide we can not reach the minimal price in decision version of the problem. Which obviously does not apply to constructive version of the problem as we want to reach maximum possible price every time.

I also created a simple powershell script that was used to run the application somewhat automatically.

The full implementation can be viewed here: https://github.com/PatrikJantosovic/NI-KOP-Homeworks

2.1 BruteForce Algorithm

Using the powerset idea [1], the main code looks like this:

```java
double numberOfCombinations= Math.pow(2, bag.NumberOfItems);
for(int i=0; i<numberOfCombinations; i++)
{
  int combWeight=0;
  int combPrice=0;
  for (int j = 0; j < bag.NumberOfItems; j++)
  {
    if ((i & (1 << j)) == 0){
      continue;
    }
    combWeight=combWeight+bag.Items.get(j).Weight;
    combPrice=combPrice+bag.Items.get(j).Price;
  }
  if(combWeight<=bag.MaxWeight && combPrice>solution.Price
  && (combPrice >= bag.MinPrice || this.Constructive)){
    solution.Price=combPrice;
    solution.Weight=combWeight;
  }
}
```

2.2 Branch&Bound Algorithm

For branch and bound solution, recursive approach was selected. The algorithm is actually very similar as running the brute-force recursively generates a whole tree, but by adding the following conditions we can cut some branches:

```
if(node.Price + node.Bound < bag.MinPrice && !this.Constructive){
   return; // do minima nedojdem = len pri DECISION, pri constructive verzii ma to netrapi
 }
 if (node.Level == bag.NumberOfItems) {
   if (node.Weight <= bag.MaxWeight && node.Price > bestNode.Price
   && (node.Price >= bag.MinPrice || this.Constructive)) {
     bestNode = new Node(node);
   }
   return;
 }
 else if (node.Weight > bag.MaxWeight)
   return;
 else if (node.Price + node.Bound < bestNode.Price)
   return;

 //raz pridam item, raz nepridam item
 Node other = new Node(node);
 other.Level++;
 other.Weight += bag.Items.get(node.Level).Weight;
 other.Price += bag.Items.get(node.Level).Price;
 other.Bound -= bag.Items.get(node.Level).Price;
 solveRecursionNode(other, bag);
 Node next = new Node(node);
 next.Level++;
 solveRecursionNode(next, bag);
```

3 Results

Tests were run on Windows 10 64bit machine, with AMD Ryzen 7 PRO 2700U and Radeon Vega Mobile Gfx. Two data sets NR and ZR were provided and have been evaluated separately as requested.

3.1 NR Data Set

Looking at the computational time, we can see significant differences in the algorithms. In the Figure 1 we can see BF (blue) line being the Brute-Force algorithm and BB(orange) line being the Branch and Bound algorithm.

Figure 2 shows us that while computational complexity of Branch&Bound is smaller than the Brute-Force, the number of visited node grows exponentially depending on N.

The histogram (Figure 3) was created for 10 items. It shows that most of the *cutting* has happened at the start, which shows that there are a lot of instances when minimum price cannot be reached at all. Also, apparently, there are lot of instances when *cutting* occurred at the end or did not occur at all and whole tree was generated.

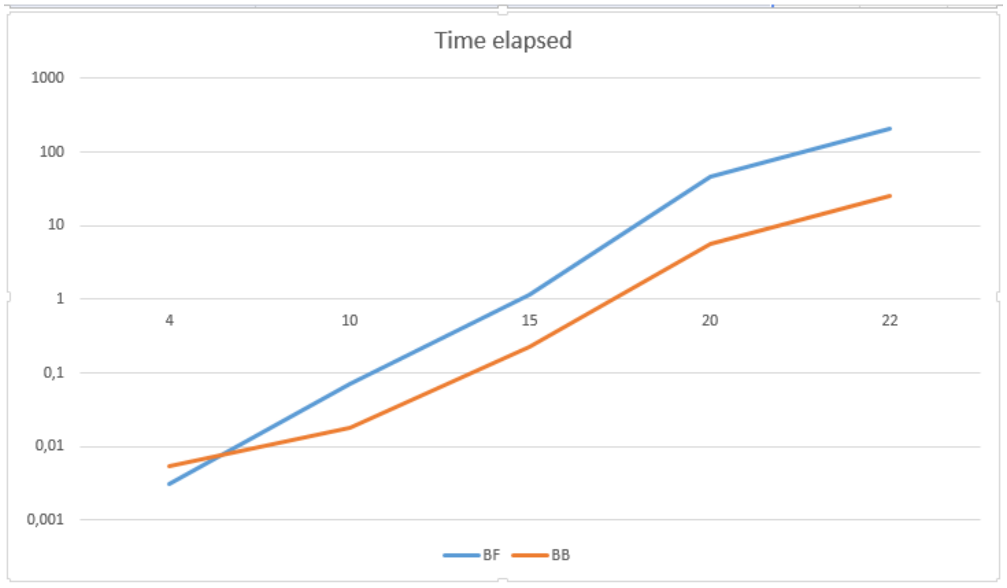

Figure 1: Time elapsed for NR data set (in seconds)

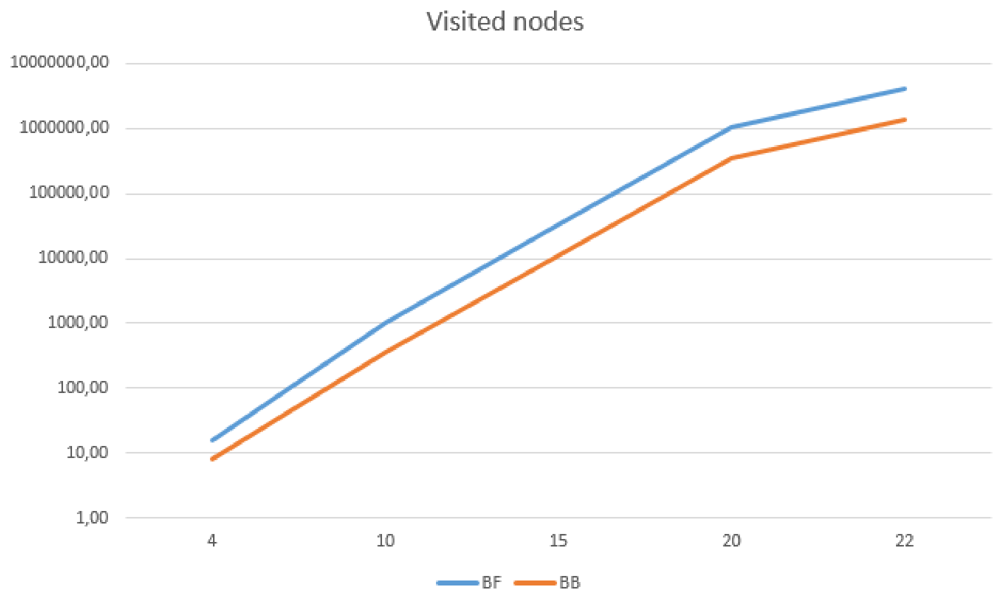

Figure 2: Average Computational complexity of NR data set (in nodes visited) for given N

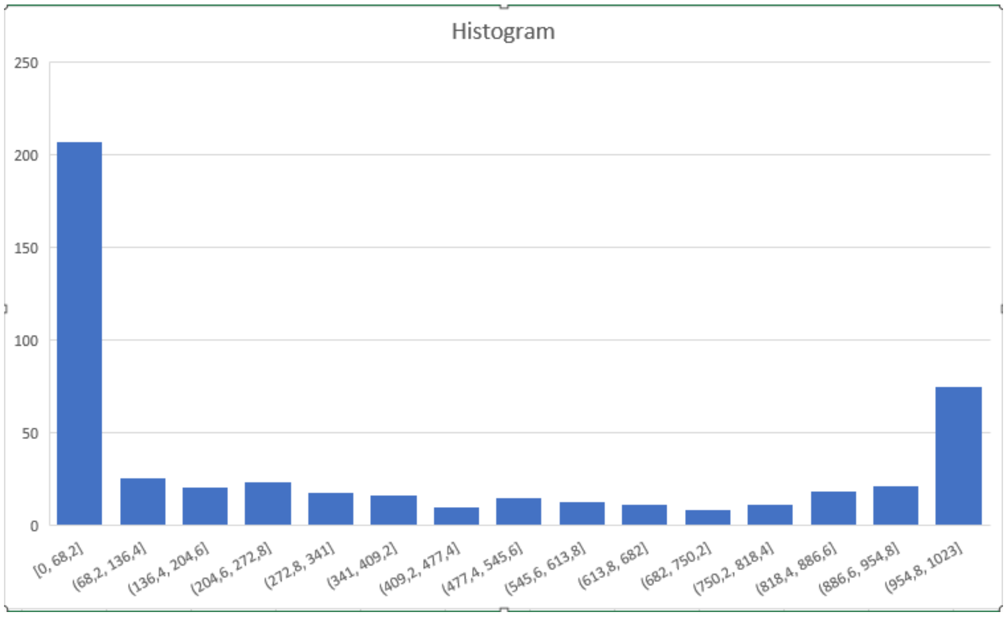

Figure 3: Histogram of NR data set for N=10

3.2 ZR Data Set

Not surprisingly, maliciously generated set caused Branch&Bound algorithm to perform worse, both in computation complexity (Figure 5) and time (Figure 4).

Moreover, the histogram (Figure 6) shows that the decrease in performance is caused by inability to perform branch cutting effectively. Please note, that the histogram starts at **745**!

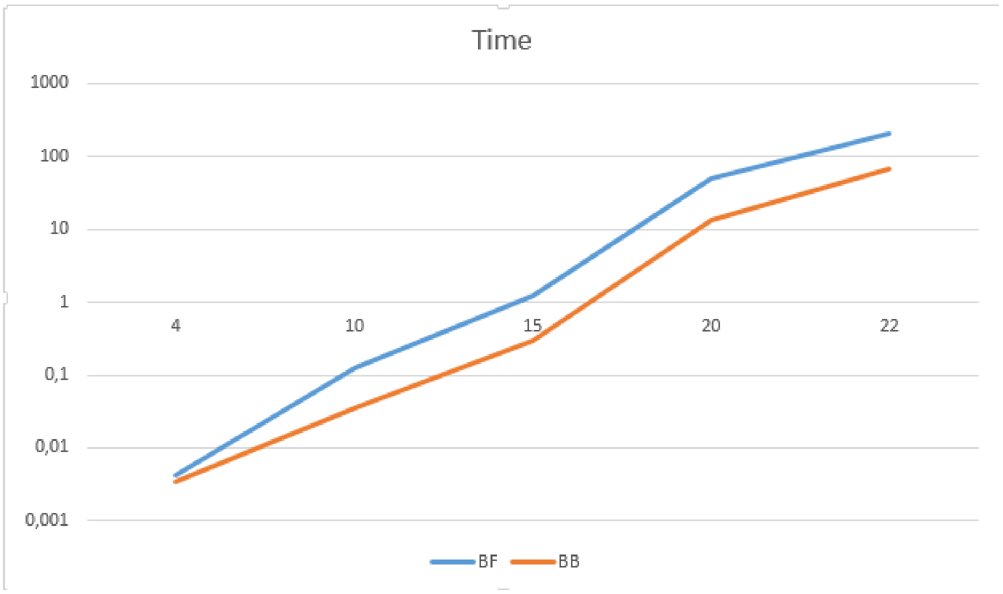

Figure 4: Time elapsed for ZR data set (in seconds)

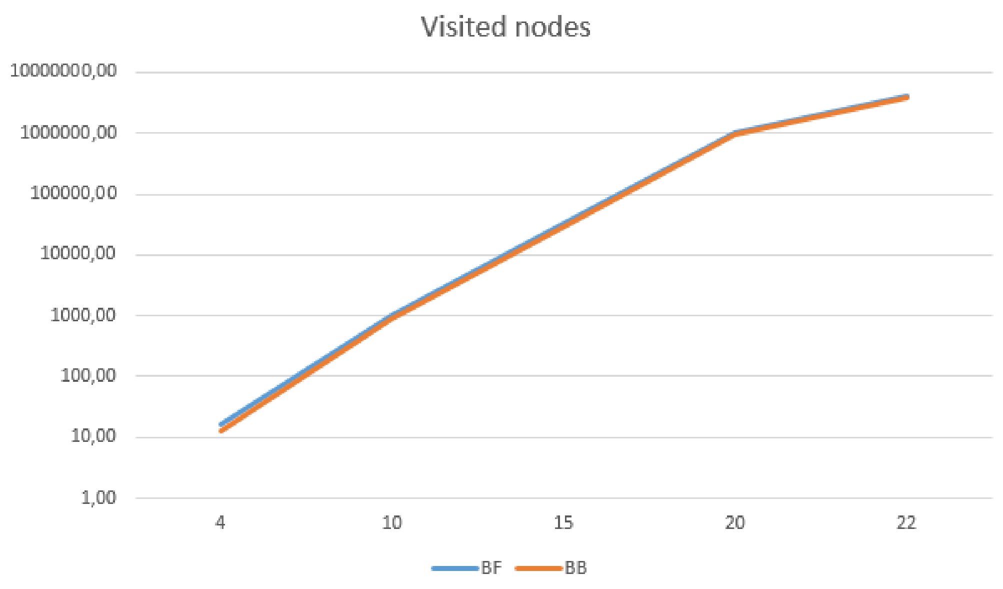

Figure 5: Average computational complexity of ZR data set (in nodes visited) for given N

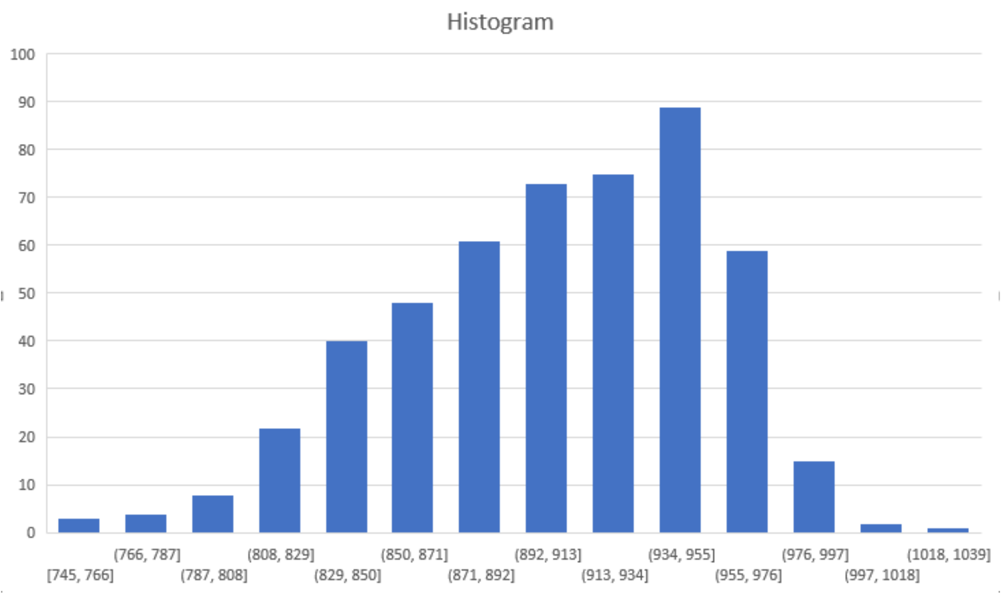

Figure 6: Histogram of ZR data set for N=10

3.3 Maximum Complexity

This subsection is added as I forgot to publish the results of maximum computational complexity on both data sets NR and ZR. The tables show that in the worst case scenario (maximum visited nodes for instance of N), Branch&Bound behaves exactly like Brute-Force algorithm and the number of nodes visited is 2^N on NR data set. Slight improvement was surprisingly seen on ZR data set. Table was intentionally chosen instead of graph as the difference is too small to be visible on graph.

N	Branch&Bound NR	Branch&Bound ZR	Brute-Force
4	15	16	16
10	1023	1019	1024
15	32767	31956	32768
20	1048575	1018174	1048576

4 Summary and Conclusions

As we observed, while computation complexity was only three times better on random sets (NR) and stayed exponentially dependent on N, Branch&Bound algorithm performed much better as it was able to cut the computational time significantly.

By comparing the two data sets (NR and ZR), we observed that maliciously generated set (ZR) caused Branch&Bound algorithm to perform almost as bad as Brute Force algorithm when looking at the computational complexity (number of visited nodes). This is clearly seen on histogram of ZR data set (Figure 6), as all of the *cutting of the branches* happened at the end or did not happen at all.

References

[1] *Powerset implementation.* URL: `https://www.techiedelight.com/generate-powerset-set-java/`.