NI-KOP Homework 2

Patrik Jantošovič

October 28, 2020

1 Assignment

Implement and compare different algorithms to solve constructive version of Knapsack Problem. The goal is to observe the computational complexity and error of heuristic algorithms such as Greedy, Greedy Redux, FPTAS and compare them to Brute-Force, Branch&Bound and Decomposition algorithms.

Full text of assignment can be viewed here: https://moodle-vyuka.cvut.cz/mod/assign/view.php?id=89697.

2 Implementation

JAVA implementation was created to solve this problem and cross-checked with provided solutions.

I also created a simple powershell script (`Run2.ps1`) 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 & Branch&Bound Algorithm

I already described both the decisive and constructive version of my implementation in previous assignment so I am going to skip this and move on to what is new.

2.2 Greedy Algorithm

Using greedy heuristic we sort the items by price to weight ration and we add item to the bag every time we can. This obviously does not guarantee the most optimal result.

```
itemsSorted.sort(new Comparator<Item>() {
   public int compare(Item i1, Item i2) {
     if (i1.Price/(double)i1.Weight > i2.Price/(double)i2.Weight) return -1;
     if (i1.Price/(double)i1.Weight < i2.Price/(double)i2.Weight) return 1;
     return 0;
   }});
 int combWeight=0;
 int combPrice=0;
 for(int i=0; i<bag.NumberOfItems; i++)
 {
    if(combWeight+itemsSorted.get(i).Weight<=bag.MaxWeight) {
      combWeight = combWeight + itemsSorted.get(i).Weight;
      combPrice = combPrice + itemsSorted.get(i).Price;
    }
 }
 solution.Price=combPrice;
 solution.Weight=combWeight;
```

2.3 Greedy Redux Algorithm

We compare the result of previously described greedy algorithm with the most expensive single item we can take and select the higher price as our solution.

```
for(int i=0; i<bag.NumberOfItems; i++)
{
  if(bag.Items.get(i).Weight<=bag.MaxWeight && bag.Items.get(i).Price>bestPrice) {
    bestWeight = bag.Items.get(i).Weight;
    bestPrice = bag.Items.get(i).Price;
  }
}
if(bestPrice>greedySolution.Price){
  solution.Price=bestPrice;
  solution.Weight=bestWeight;
}
```

2.4 Dynamic Algorithm

I implemented the decomposition by price as explained on the tutorial and by following the instructions from course website [1]. I at first fill the first two columns of the dynamic table and then compute the rest. After the table is computed, I look through the last column from top to bottom looking for the first value (sum of weights) that fits in the bag. Then I collect the items by comparing values in `n-th` and `n-th - 1` column.

```
// spocitam zvysok tabulky
 for(int i = 2; i <= n; i++){
   for(int j = 0; j <= T; j++){
     if(j>=items.get(i-1).Price && dynamicTable[i-1][j - items.get(i-1).Price] != Integer.MAX_VALUE)
     {
       dynamicTable[i][j] = Integer.min(dynamicTable[i-1][j],
           dynamicTable[i-1][j - items.get(i-1).Price] + items.get(i-1).Weight);
     }
     else{
       dynamicTable[i][j]=dynamicTable[i-1][j];
     }
   }
 }
```

2.5 FPTAS Algorithm

FPTAS algorithm was implemented with parameter `e` that I used to calculate the scaling factor `K` instead of bit neglecting method [2]. Item prices are then divided by `K` and rounded down and Dynamic algorithm with decomposition by price is used on this new set of items.

After that, dynamic algorithm tells us which items were taken from modified set, so we can find the same items in original set of items and calculate the real price of the bag.

Later, I noticed that mainly on ZKW files the algorithm behaved suspiciously and I realized that sometimes I have created items with 0 price by rounding down and the final price might therefore be zero. I disregarded these cases and did not calculate error on them.

```
double K = bag.MaxItemPrice*e/bag.NumberOfItems;
if(e==0) K=1; // test ze to funguje spravne -> je to defacto obalka na dynamicsolver
              //s odhodenim tazkych itemov
int totalPrice = 0;
List<Item> items = new ArrayList<Item>();
```

```
for (Item i: bag.Items) {
  if(i.Weight>bag.MaxWeight) continue;

  Item item = new Item();
  item.Weight=i.Weight;
  item.Id=i.Id;
  item.Price= (int) Math.floor(i.Price/K);
  totalPrice+=item.Price;
  items.add(item);
}

dynamicSolution=dynamicSolver.Solve(items, items.size(), totalPrice, bag.MaxWeight);
```

3 Results

Tests were run on Windows 10 64bit machine, with AMD Ryzen 7 PRO 2700U and Radeon Vega Mobile Gfx.

Three data sets NK, ZKC and ZKW were provided and have been evaluated separately as requested. On FPTAS graphs, FPTAS e=0 can be seen, which is obviously the Dynamic algorithm without items that are too heavy on their own. This is plotted just for comparison.

3.1 NK Data Set

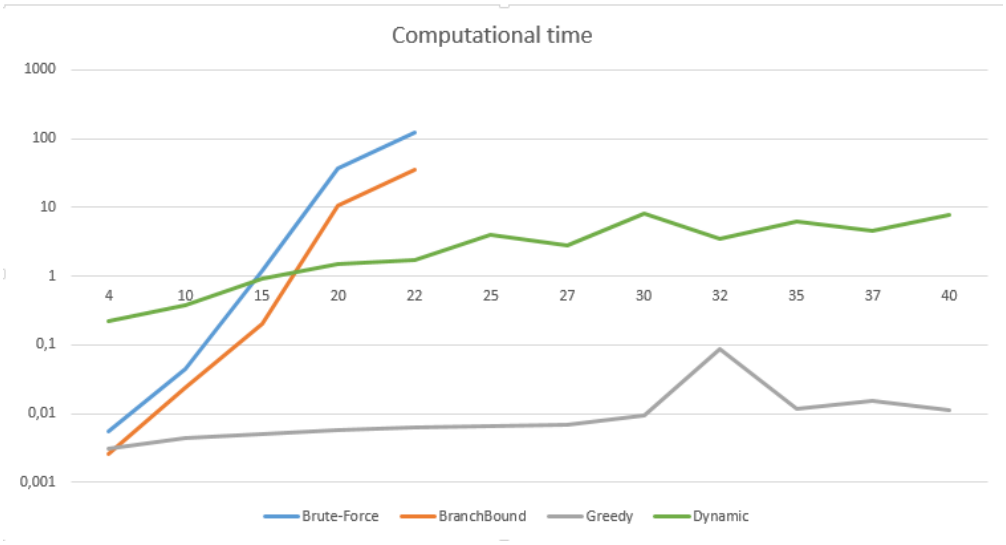

Figure 1: Time elapsed for NK data set (in seconds)

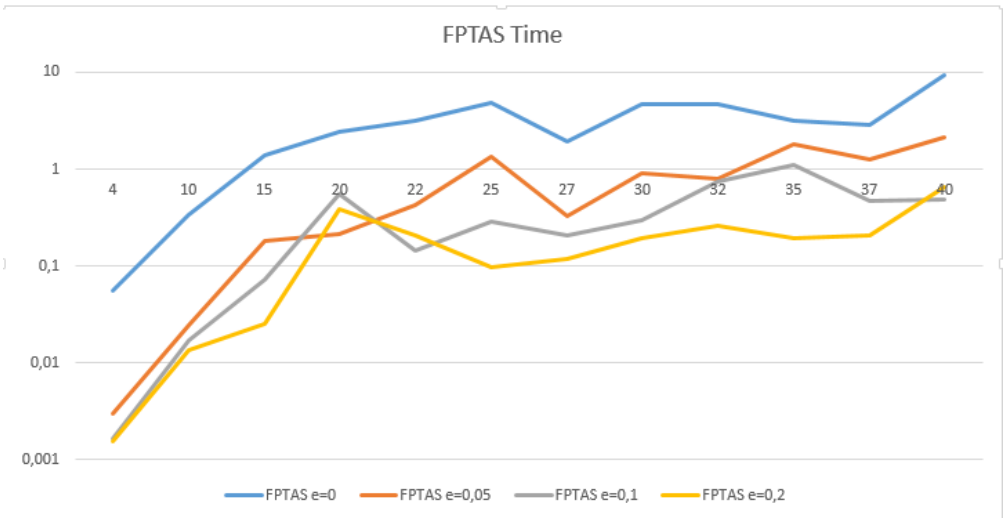

Figure 2: Time elapsed for NK data set (in seconds) comparing paramter e in FPTAS

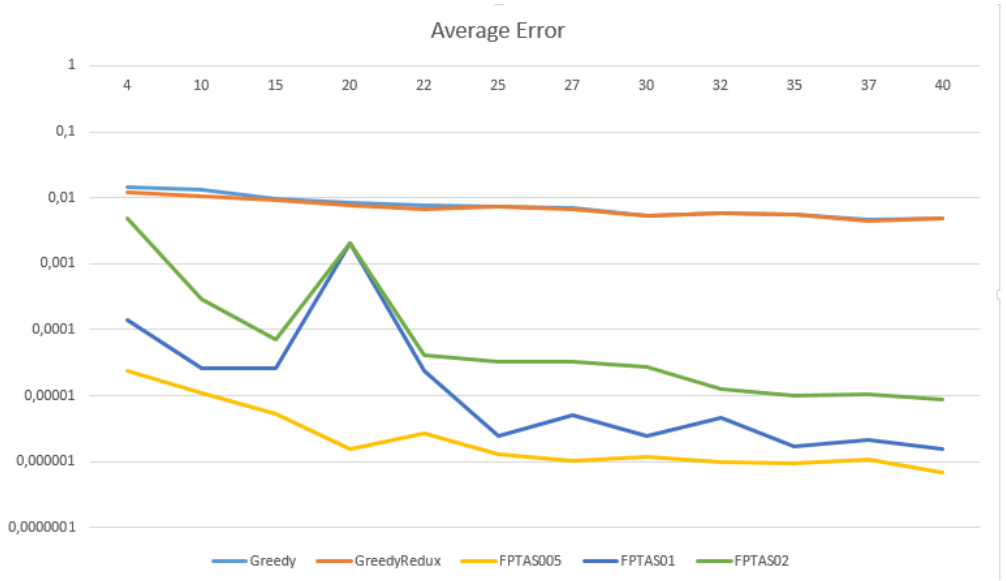

Figure 3: Average error for NK data set

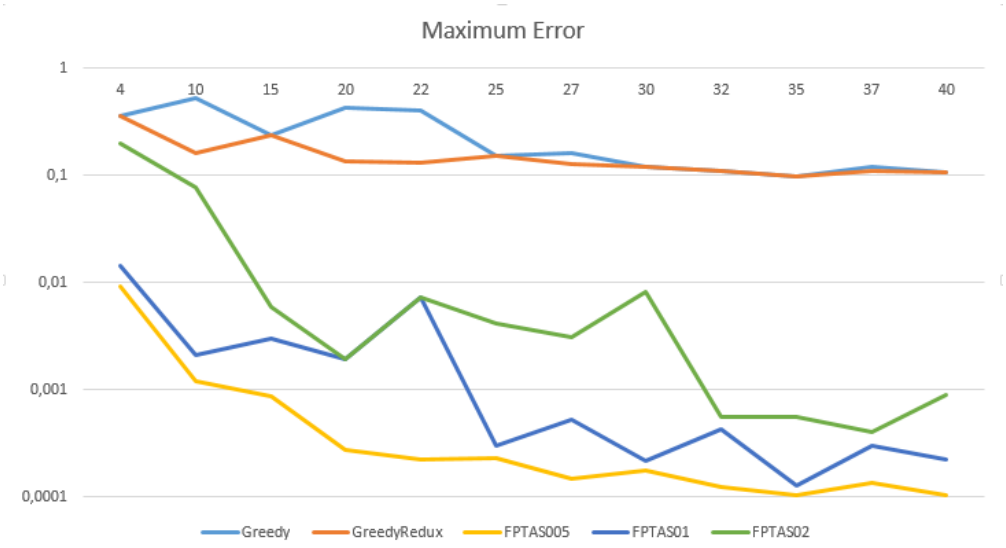

Figure 4: Maximum error for NK data set

3.2 ZKC Data Set

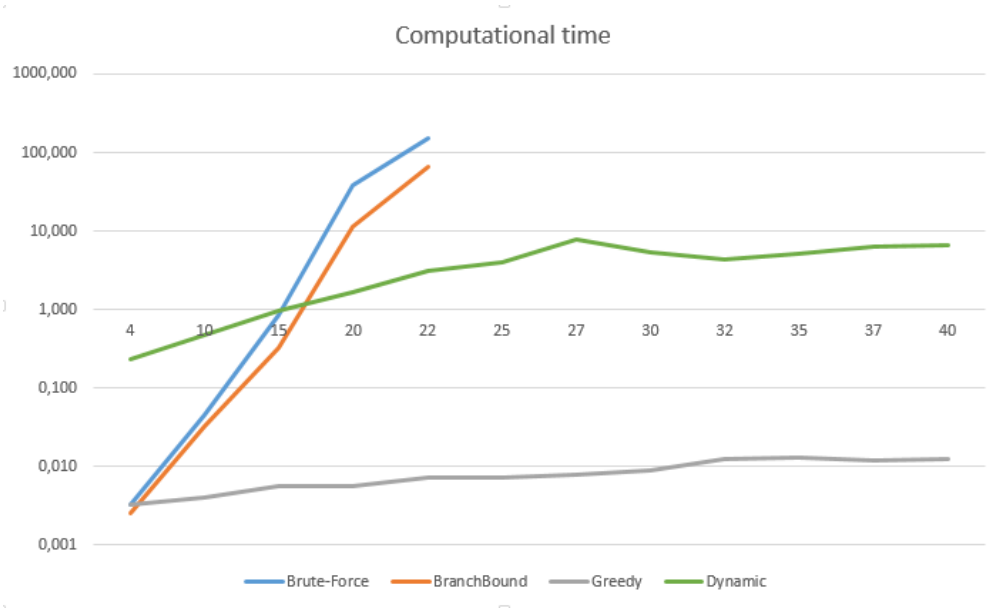

Figure 5: Time elapsed for ZKC data set (in seconds)

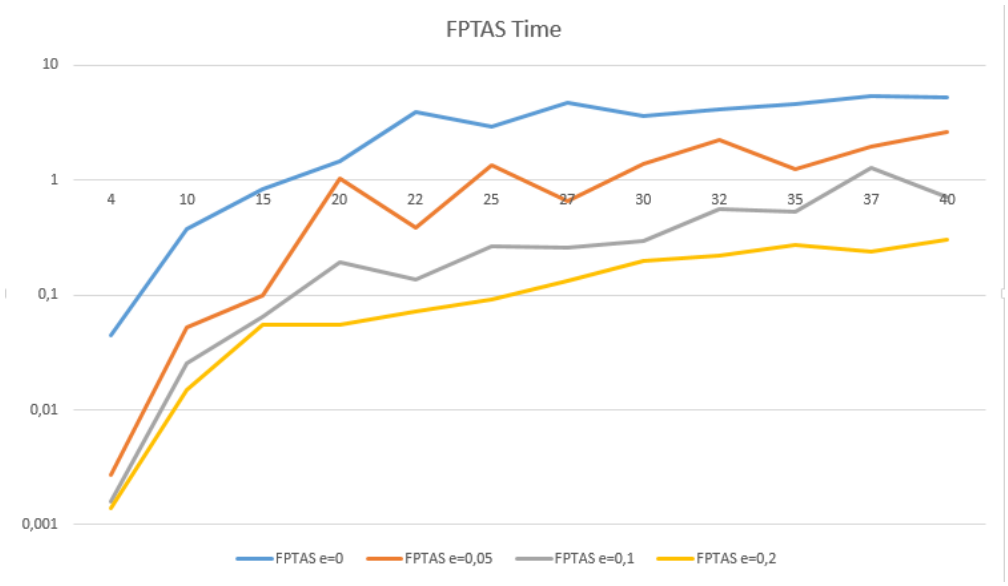

Figure 6: Time elapsed for ZKC data set (in seconds) comparing paramter e in FPTAS

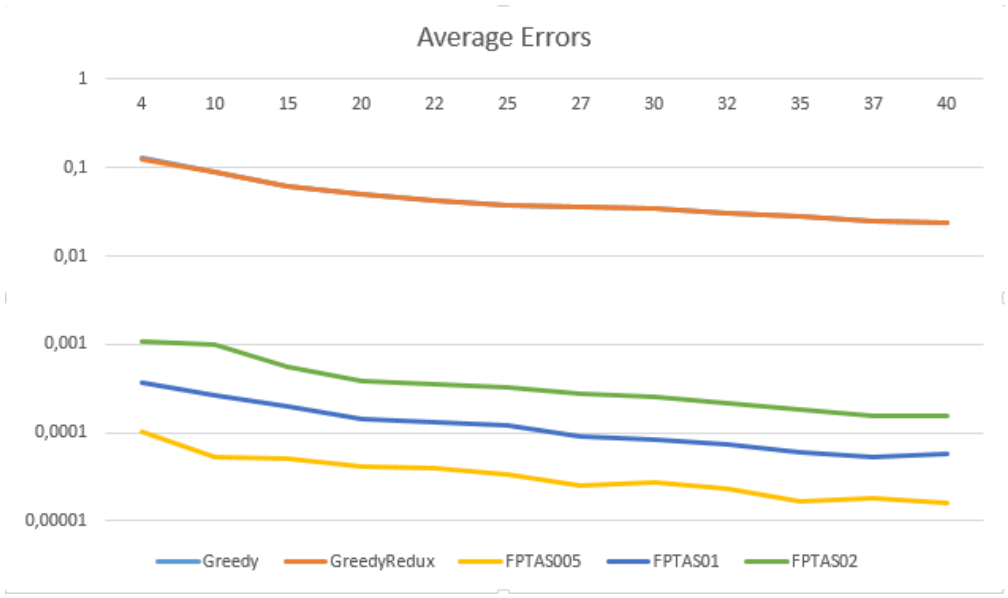

Figure 7: Average error for ZKC data set

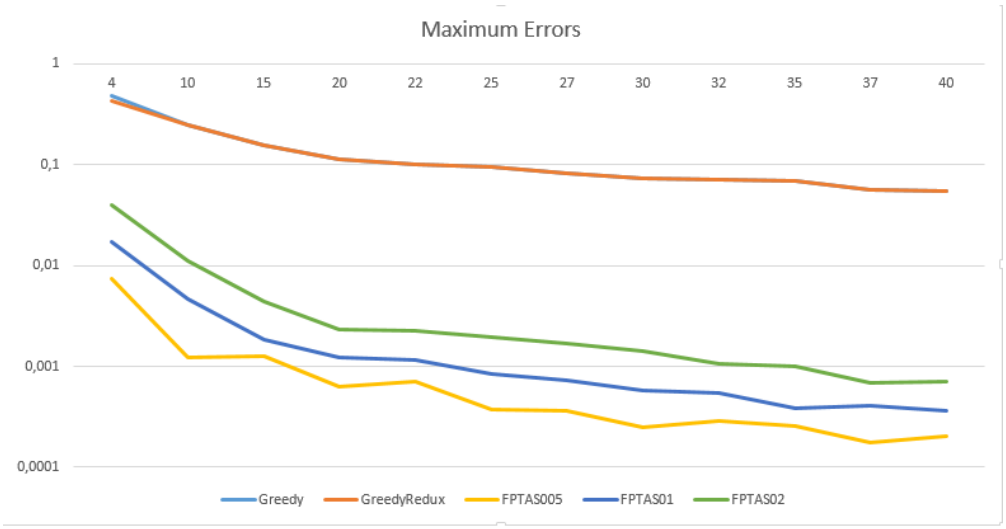

Figure 8: Maximum error for ZKC data set

3.3 ZKW Data Set

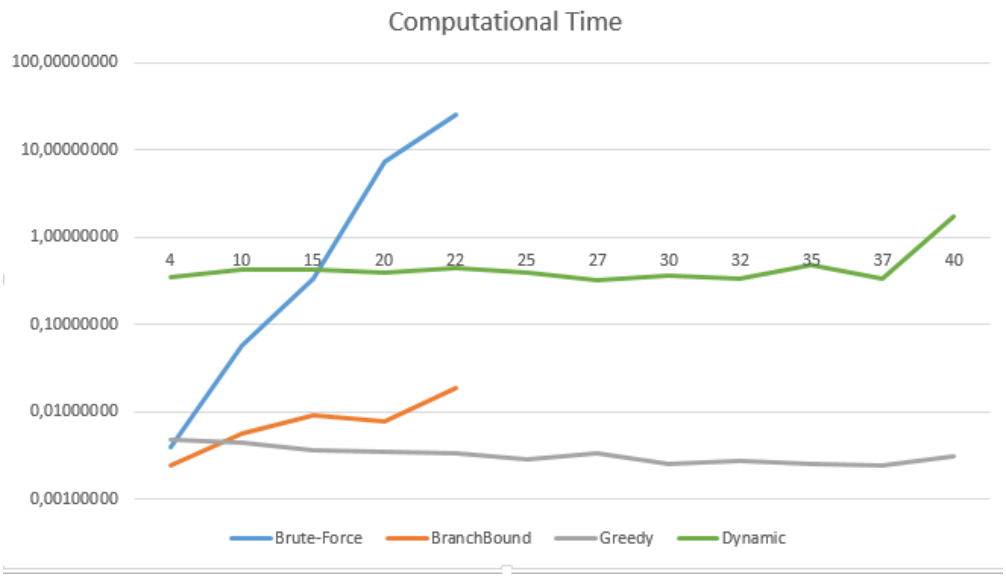

Figure 9: Time elapsed for ZKW data set (in seconds)

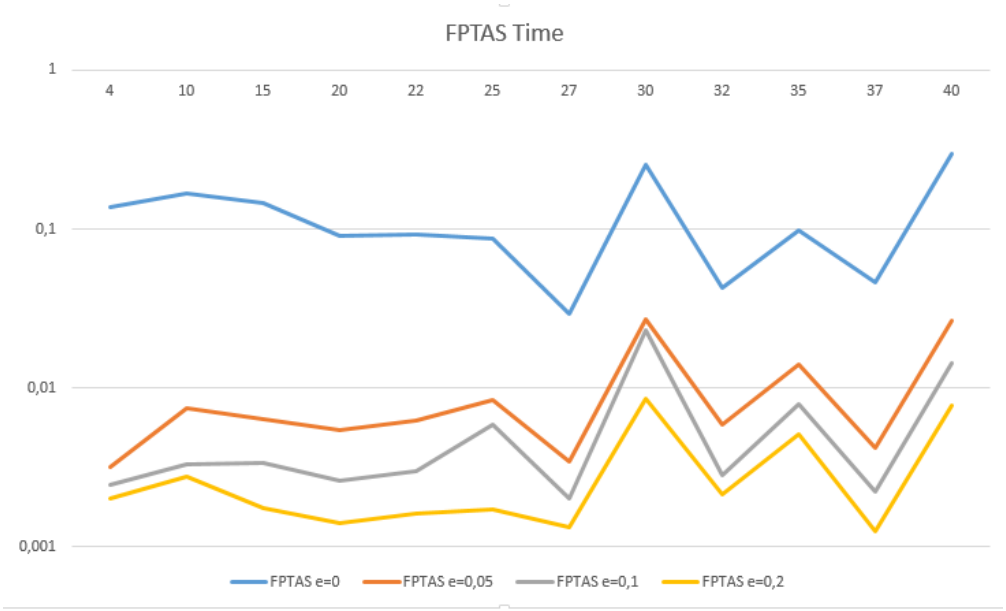

Figure 10: Time elapsed for ZKW data set (in seconds) comparing paramter e in FPTAS

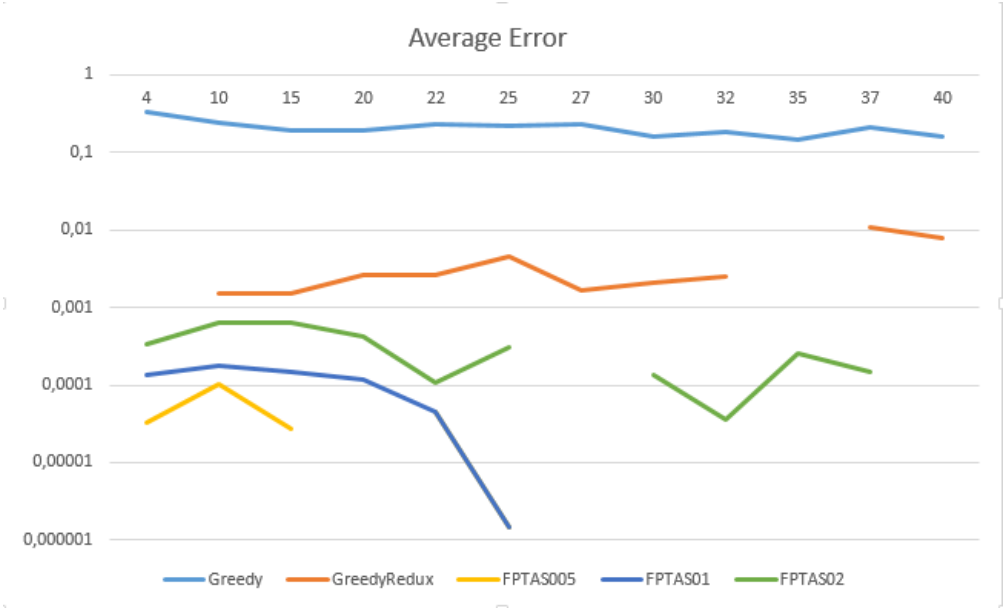

Figure 11: Average error for ZKW data set

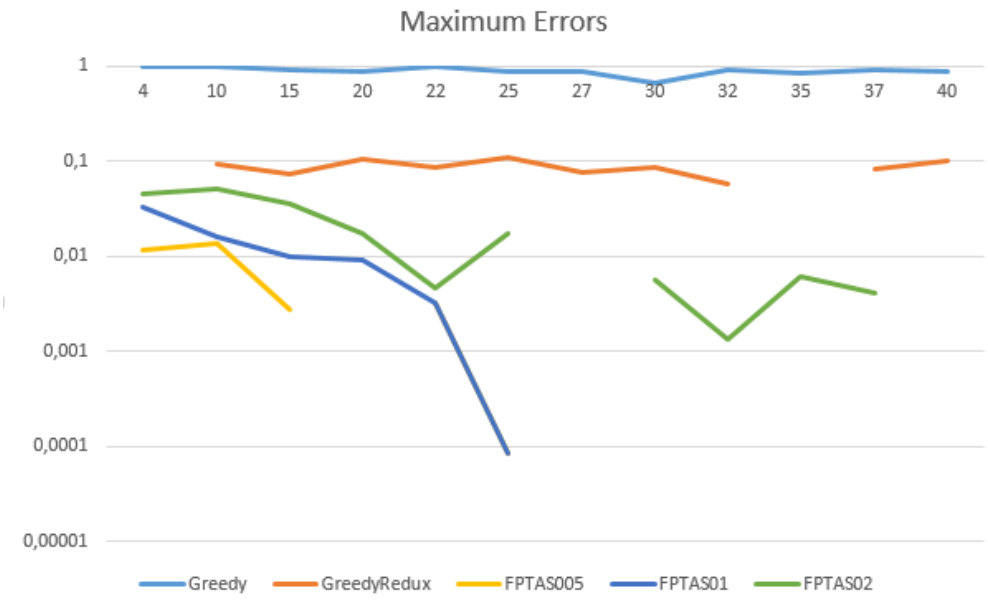

Figure 12: Maximum error for ZKW data set

4 Summary and Conclusions

While Greedy heuristics were fastest, they have been the ones with most errors. On the other hand, FPTAS with paramter `e = 0.05` and `e = 0.1` still performed in very good time and made far less mistakes. Looking at the parameter e, the most notable performance jump was seen between `e = 0` (which is basic Dynamic algorithm) and `e = 0.05`, and also between `e = 0.05` and `e = 0.1`. In other cases, the performance difference was not that notable, the algorithm just made more mistakes with growing `e`.

I also observed differences mainly in dataset ZKW when compared to the other two. The computational time for Branch&Bound and Dynamic algorithm was better, which is due to the nature of the data as a lot of cases were simplified because of heavy items. On the other hand, Greedy algorithms did not find correct solutions most of the time on these data sets.

Gaps in ZKW error graphs can also be seen, which is due to the fact that for some instances FPTAS found correct solutions for given n every time, and/or error was not calculated since FPTAS did not calculate anything because of 0 priced items.

References

[1] *Decomposition instructions*. URL: `https://moodle-vyuka.cvut.cz/mod/page/view.php?id=89682.`.

[2] *FPTAS instructions*. URL: `https://moodle-vyuka.cvut.cz/mod/page/view.php?id=89341`.