Homework 1 - Solving the decision knapsack problem exactly

Problem Statement

Knapsack problem is a problem in which a thief needs to decide which items he should put in the bag. He wants to steal the most value possible with the items so that they can still fit inside the bag.

The only type of combinatorial problem we are doing for this homework is a decision problem. Which means we are only looking for combination of items which fit the bag and are exceeding the required minimum cost.

Analysis and description of possible solutions

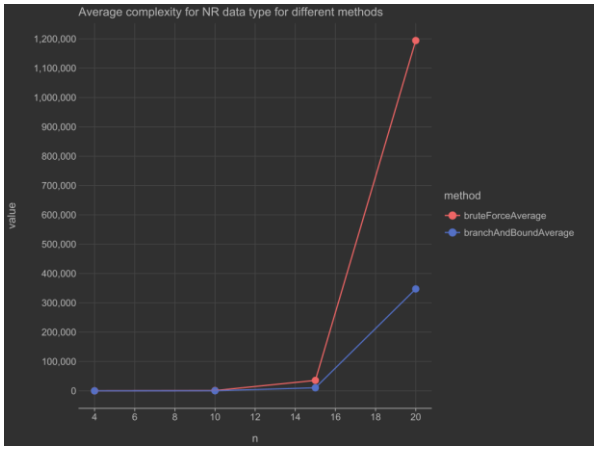
In combinatorial problems the important thing is that it has finite domain and finite number of configuration variables.

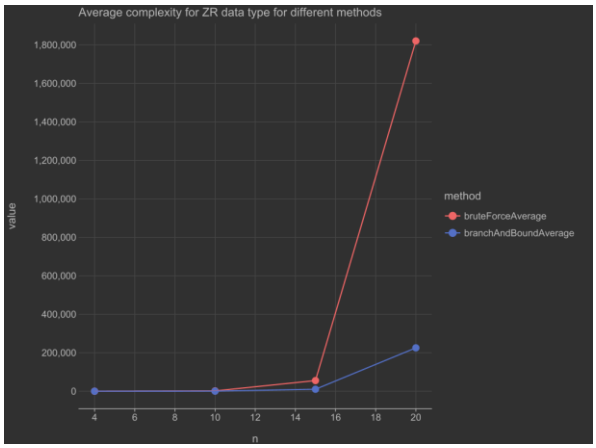
The most basic approach to solve the problem then would be to try every single possible combination of variables (in knapsack that would be every single combination of items that could be brought). For this solution I used recursion method which goes through each item one by one and makes two recursive method calls (one with the item and one without) until it goes through each possible combination. One way to optimize the search space was to stop the search when the weight of the bag was already exceeding the capacity.

The other solution to this problem was to use the branch & bound method which prunes the search space significantly more. It is like the brute force approach, but the catch is that it also uses pruning based on the cost function. For each current state of the search, we know exactly how much cost we currently have in the bag and what is the sum of the cost of all the other items we haven't decided by now. If we sum those numbers and they are less than the minimum required cost, we know for sure that even if we take all the remaining items, we for sure won't meet the required cost so we can abandon these possible combinations and continue the search elsewhere. My implementation is similar to the brute force one with the addition of comparing upper bound with the desired cost.

Results and interpretations

After conducting evaluation on decision versions of instance sizes 4, 10, 15 and 20 I got these results.

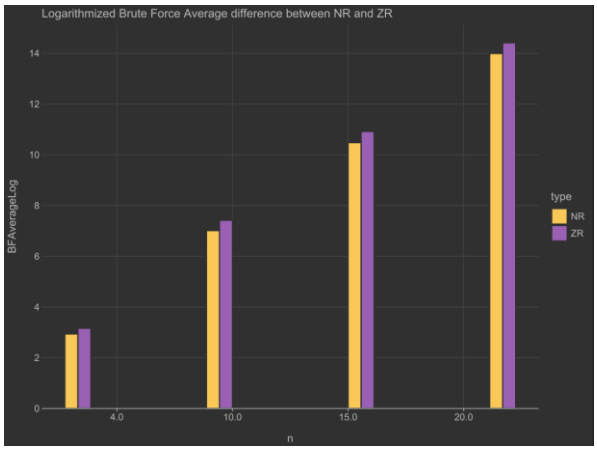


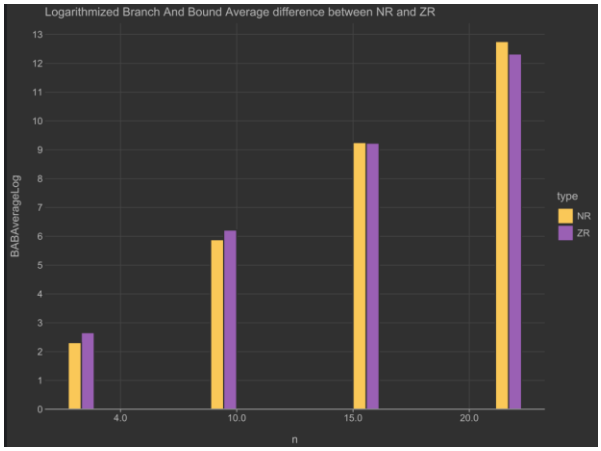


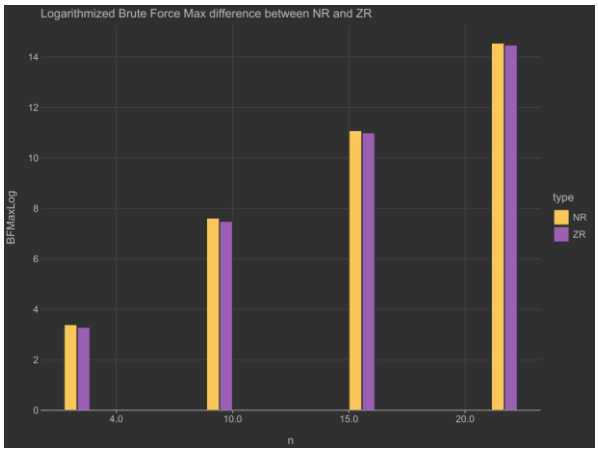
We can see that for both of data types (NR and ZR) complexity rises much faster as the instance size increases. This proves our first theory that the brute force complexity of knapsack problem is 2^n (each item can either be included or not be included so the overall search space would be 2^n). We can also conclude that branch & bound method prunes the search space significantly because it also uses cost function to optimize the search.

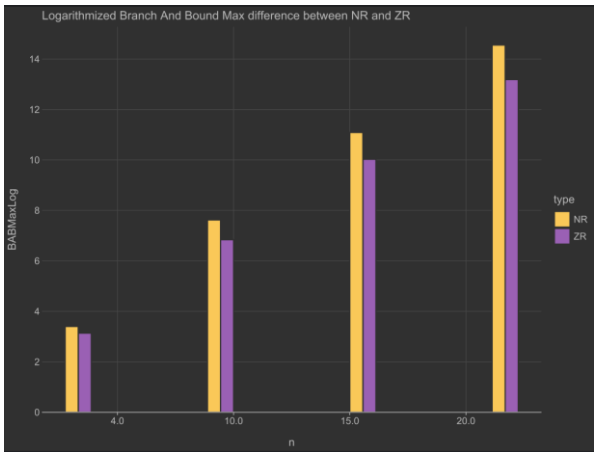
n	type	bruteForceAverage	branchAndBoundAverage	bruteForceMax	branchAndBoundMax
4	NR	18.862	10.122	30	30
10	NR	1110.862	358.978	2046	2046
15	NR	35575.604	10442.412	65534	65505
20	NR	1194365.86	347480.336	2097150	2097066
4	ZR	23.614	14.296	27	23
10	ZR	1667.156	501.64	1799	934
15	ZR	55321.29	10217.664	60077	22645
20	ZR	1820483.232	225418.574	1940038	533831

One interesting thing we can see from this table about maximum complexity of the brute force method is that it is always for two smaller than the 2^{n+1} (e.g. $30 = 3^5 - 2$)





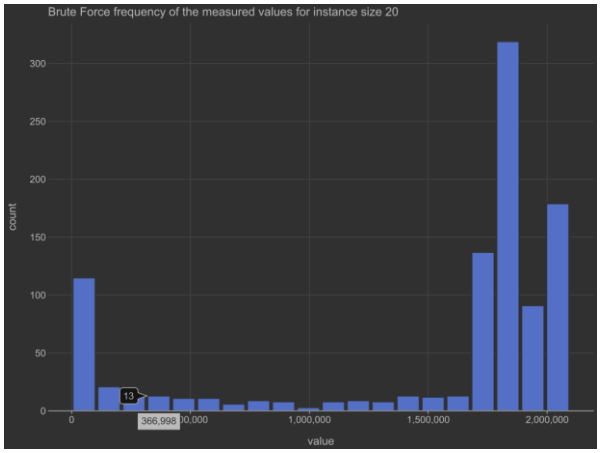


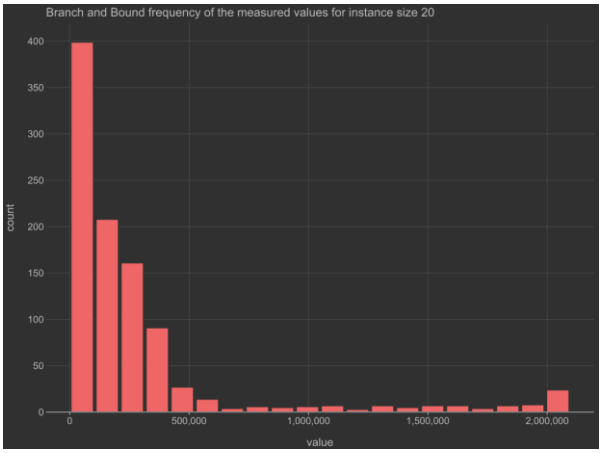


After conducting research on NR and ZR data types independently, we can see that in most cases average ZR types of complexity is bigger than the NR with an exception in brach & bound method for larger n. This is because ZR examples are made to be difficult to solve, they lead to more “edge case” examples.

Interestingly, maximum complexity is always bigger for NR data type than for the ZR. While ZR examples are carefully made, NR examples are randomly generated so they can from time to time generate extremely difficult cases that need exploring almost entire search space.

All the values in last 4 graphs have been logarithmically scaled to fit graph for all instance sizes. For real values refer to the table above.





Comparing these two histograms for instance size 20 it is obvious that branch & bound method is much less complex than brute force approach. In brute force histograms most of the values fall between 1500000 and 2000000 with an exception at low values while for branch & bound method we can see that most of the values fall between 0 and 500000.