Zuzanna Gawrysiak ID: 148255
Agata Żywot ID:148258

REPORT 5 – Knapsack Problem

Introduction

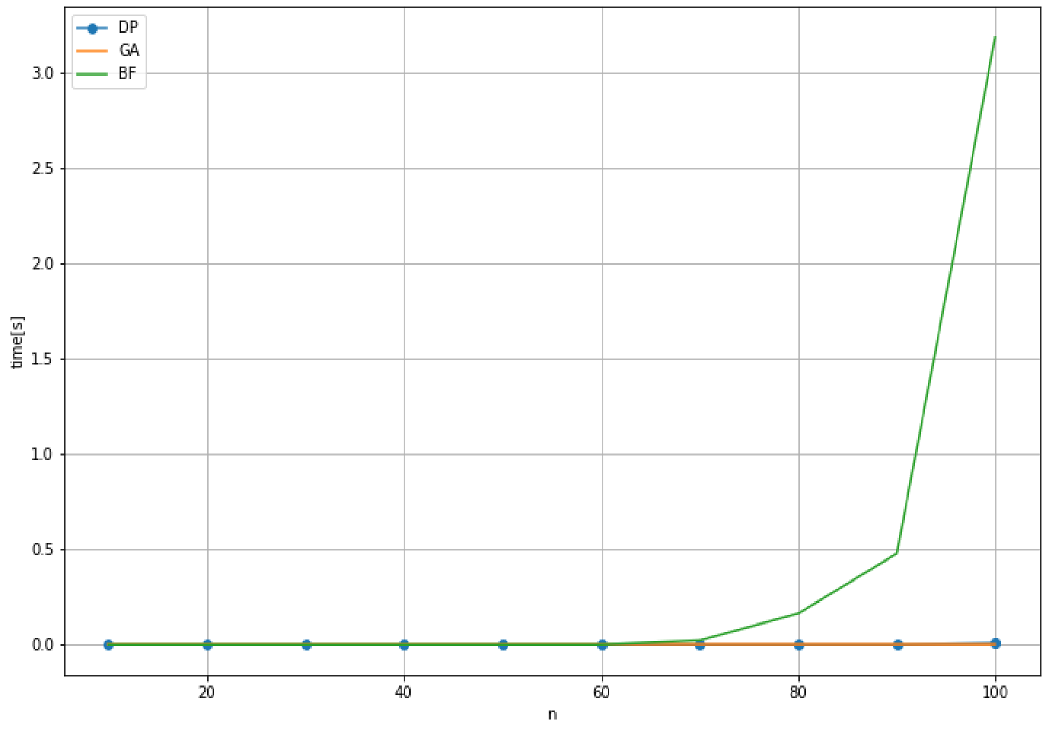
In this report, the comparison of three algorithms considering the Knapsack Problem(KP) will be performed. These approaches include: dynamic programming (DP), greedy algorithm (GA) and brute force (BF). GA was performed with respect to the profitability coefficient, i.e. value/weight. The data was randomly generated with instance sizes: N=number of items, W=total capacity, appropriate to the considered characteristic. In every experiment, except for the *special case* (see below in the section Quality), weights *w* were randomly taken from range [10, 1000] and values *v* from range [100, 10000].

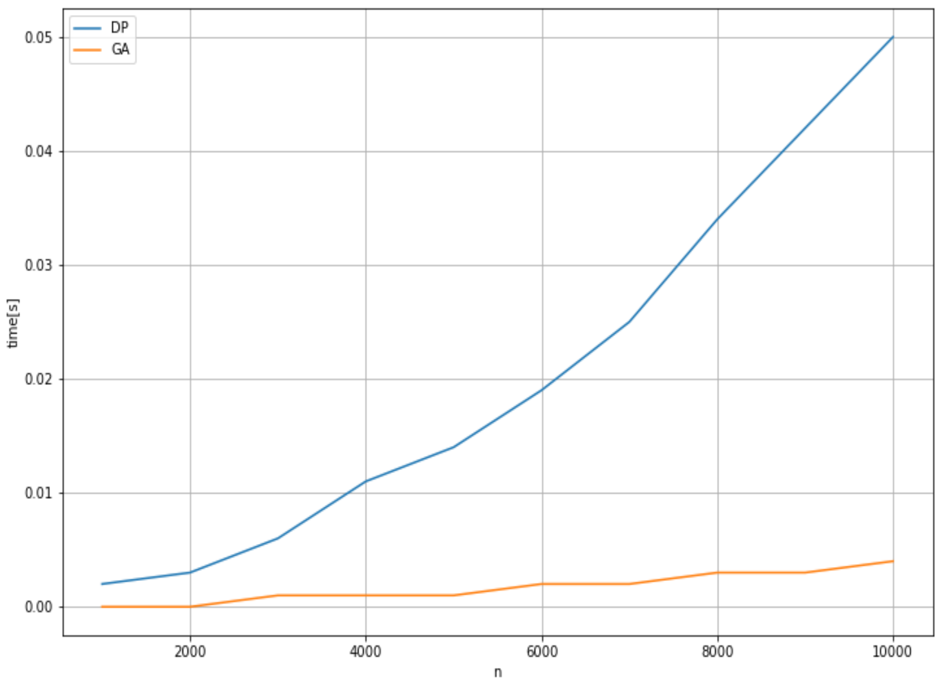
Effectiveness

Method

For the comparison of three algorithms (chart 1), the data was generated for n from range [10, 100] with step=10. The capacity was increased simultaneously with n, starting at W=20. The second chart excludes BF to compare DP and GA. It is based on data from range [1000, 10000] with step=1000. The 3D chart representing DP performance is created by measuring time for every pair from the cartesian product N x W, where N=W={10, 50, 100, 500, 1000, 2000, 5000, 10000, 15000, 20000, 25000, 30000}.

Results

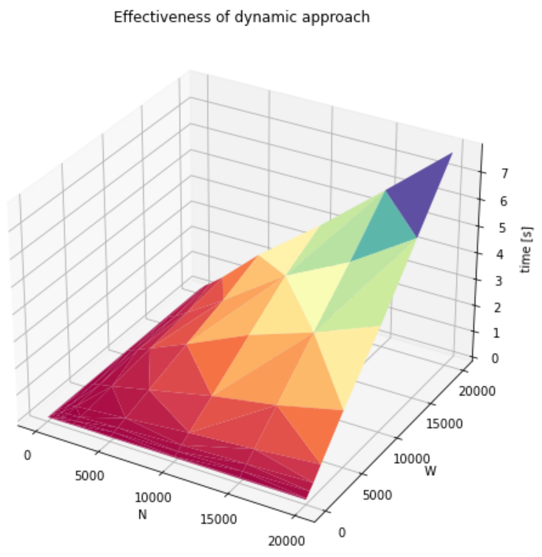




<u>Observations and conclusions</u>

BF has time complexity O(2^n), since in the implemented algorithm all subsets of items (whose total weight is smaller than W) are considered and there are two cases (included/not included) for every item. GA complexity is based on the sorting algorithm used, therefore in this case O(nlogn). The complexity of DA is dependent on two variables – O(N*W), since in the implementation a matrix NxW is constructed. Generally, taking only time into consideration, GA is the best choice. However, in the case when W < logn, DP will perform better. Moreover, a greedy approach does not ensure finding the optimal solution. The problem of quality is investigated in the next section.

The 3D graph was constructed to show that DP is linearly dependent on W and N.

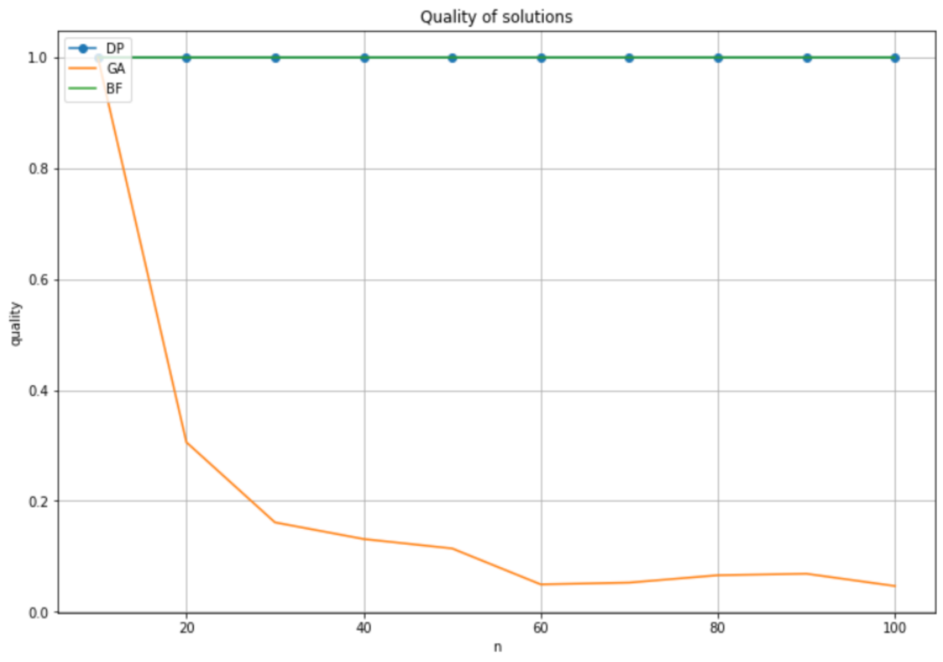


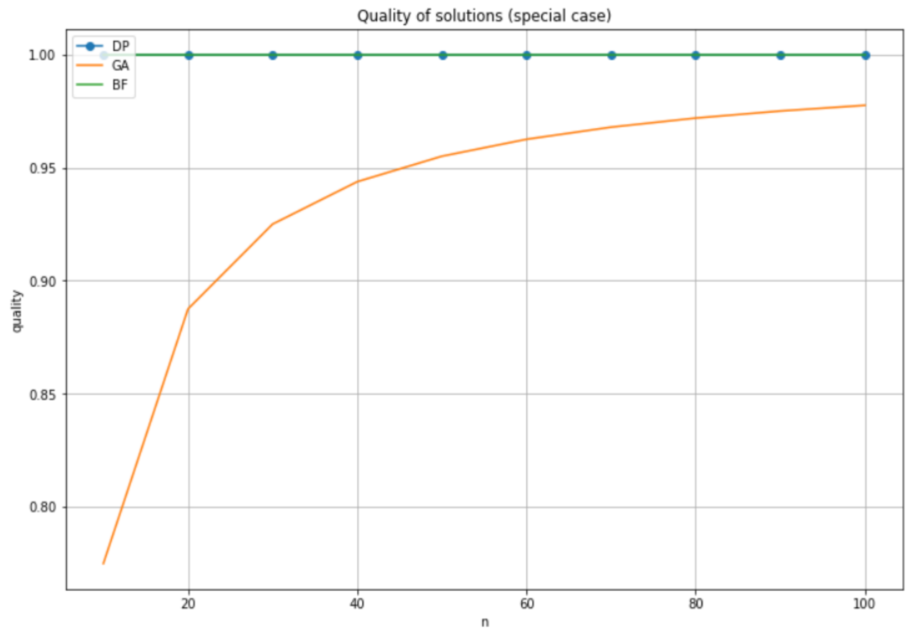
Quality

Method

As far as measuring the quality of all aforementioned approaches is concerned, both the range of n and W as well as the property of total capacity being changed are the same as before. The *quality* value provided on the chart is the ratio between the result obtained and the optimal solution (i.e. the result of exhaustive search, since it checks every possible subset and solves the knapsack problem). There is also a *special case* included which was designed to count against the greedy approach. The data was generated in such a way that all but one item had the value *v = 5* and *w = 10*. The *special* item had *v = 6* and *w = 11* which made its profitability coefficient better compared to others. GA always included it in the solution and eventually left some space but not enough to store one more item which resulted in a suboptimal solution.

Results





Observations and conclusions

The charts presented above immediately show that GA does not solve the knapsack problem. It gives only an approximation of the optimal solution that (in most cases) loses its quality with increasing n and W. It was an expected phenomenon since more items and greater capacity imply more possibilities of ‘packing’ a knapsack and therefore more unfavorable choices could have been made by GA. In every case the quality of DP is the same as the quality of BF (quality equal to 1) which points out that DP also solves the knapsack problem.

Considering the *special instance* of the problem, since for every n there is exactly one distinctive item, the negative impact on overall performance of GA decreases as W increases simultaneously with n, which can be observed in the chart. The purpose of implementing such an extreme case was to check the correctness of the algorithm and see what types of unfavorable choices can be made by GA.

Complexity class of the problem

Knapsack Problem can be solved using the exhaustive approach and the correctness of each solution can be verified in polynomial time which ensures KP belongs to NP. What is more, every NP problem can be reduced to KP which makes the problem NP-complete. Considering DP (which solves KP) having time complexity of O(nW) one may initially think that the problem can be solved in polynomial time. However, since the runtime of the algorithm is associated with both the size of the input (n) and the magnitude of the input (W; the value provided is binary), it means that to double the size of n is to double the *value* of n, whereas doubling the size of W implies its *binary length* being twice as large. Consequently, the *value* of W grows exponentially and the algorithm’s runtime is pseudo-polynomial. Therefore, Knapsack Problem belongs to a weakly NP-complete class.

Sources

https://stackoverflow.com/questions/3907545/how-to-understand-the-knapsack-problem-is-np-complete
https://stackoverflow.com/questions/4538581/why-is-the-knapsack-problem-pseudo-polynomial
https://www.youtube.com/watch?v=9oI7fg-MIpE