Antoni Solarski ID:148270, Nina Żukowska ID:148278

REPORT 5

Introduction:

In this report, three ways of solving the knapsack problem are analyzed, two of which generate optimal solutions (Exhaustive algorithm and Dynamic programming algorithm), while the third one -the greedy algorithm- is bringing forth a suboptimal one.

Dynamic Programming in this approach, as the name suggest – the problem is solved in a recursive manner, storing the intermediate results (maximum value for a given capacity) in a 2D array in order for the program to run faster (not wasting time on already computations already made).

Exhaustive Algorithm – since every optimization problem can be broken down into a multitude of decision problems, this approach does exactly that – it generates all possible solutions-exhausting all of the possibilities, propagating the best one out.

Greedy Algorithm – here the elements get sorted according to their value/weight quotient in a decreasing order. "Value-dense" objects are favored in the process of selection: the algorithm iterates through the array starting with lower indices and working its way down to the least "value-dense" objects, an item gets selected if it fits in the knapsack (together with the previously selected objects)

I Time complexity and time of execution.

When it comes to the execution time of all of the above listed methods of packing the knapsack, one has to make a clear distinction between the two optimal algorithms and the suboptimal one. Just as the names suggests, the greedy algorithm oftentimes does not provide nearly as good of a solution as the two previous ones(algorithms), keeping this disclaimer in mind one can further examine and compare the execution times.

The most time-consuming method of solving is the exhaustive approach (**EA**) – as the name implies for each items a decision "take – not take" has to be made, leading to 2 different possible streams of outcome regarding each item – that is for n items 2^n different solutions have to be examined, and the best one (optimal) is being propagated through – thus the memory complexity is O(1). This way of conduct results in exponential (more precisely **O(n*2^n)**) execution time as n increases, making this way of approaching the problem highly inefficient as the bulk of operations is repeated over and over again. One can also point out that the capacity of the knapsack has no influence on the time execution, as a simple check for each solution is performed (whether the given selection of items fits into the knapsack).

The second method outputting an optimal solution is the dynamic programming (**DP**) approach, the complexity of which is **O(n*s)**, n being the number of items, s being the size of the knapsack. In the aforementioned array, fractional optimal solutions are being stored after being computed using the recursive approach, which provides a basis for the later generated ones. The array can be of dimensions n by s, but the memory complexity can be reduced by rewriting the newly generated row as the first one, making the algorithm work on a total of 2 rows (of length s) at a time. Contrary to the other approaches the complexity depends heavily on n and s, which is shown on *Chart 1*.

The last method – the greedy algorithm (**GA**) as mentioned prior is based on a sorted array of elements according to their value density – the sorting is of complexity **O(n*log(n))** and for each item only the condition of fitting in has to be checked – these factors contribute to GA being the fasted among the listed algorithms, although one can not compare them, because of the different outputs they provide (approximation of optimal solution).

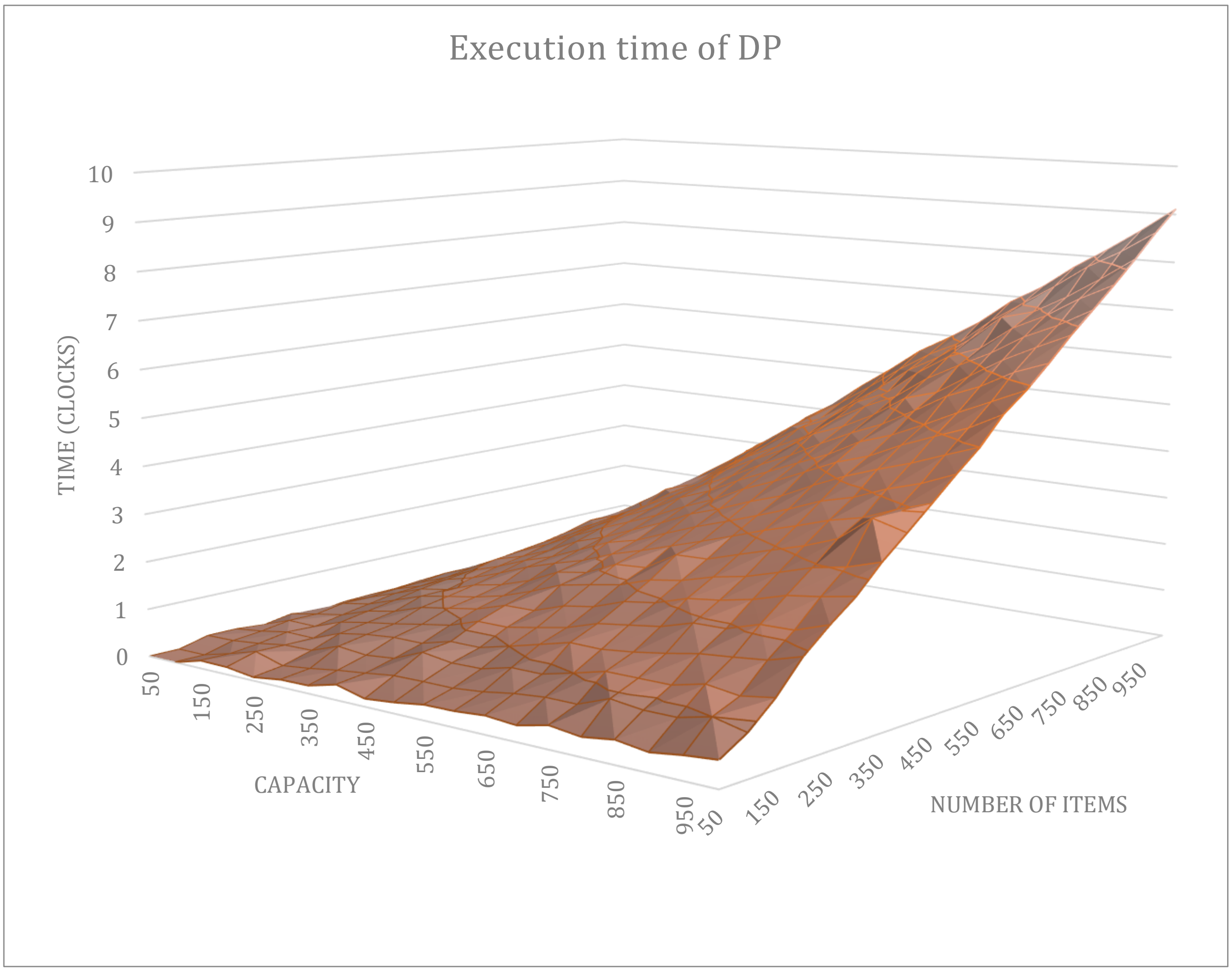


Chart 1

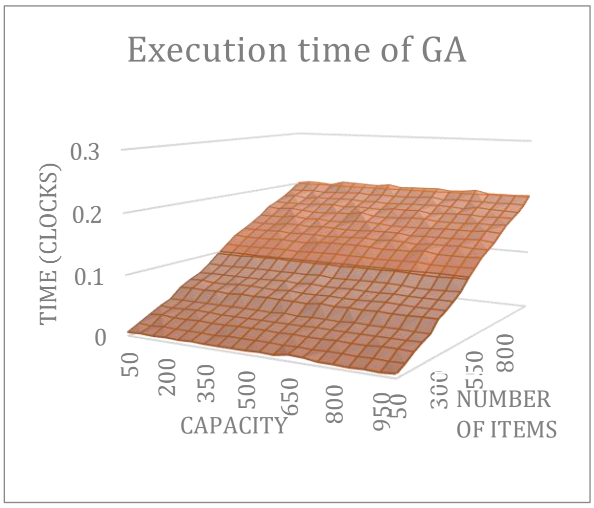


Chart 2

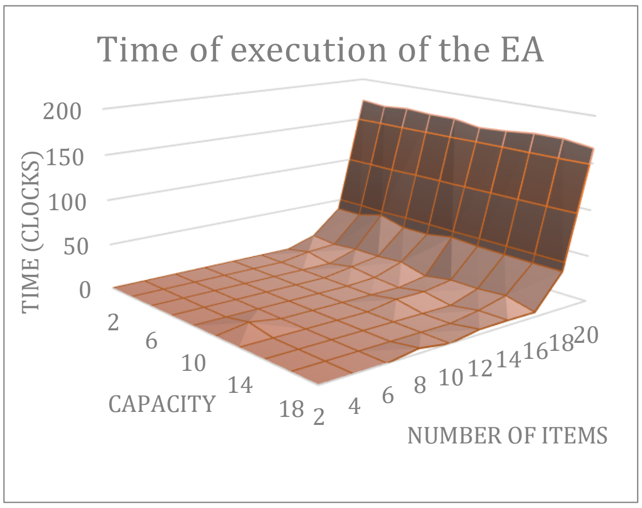
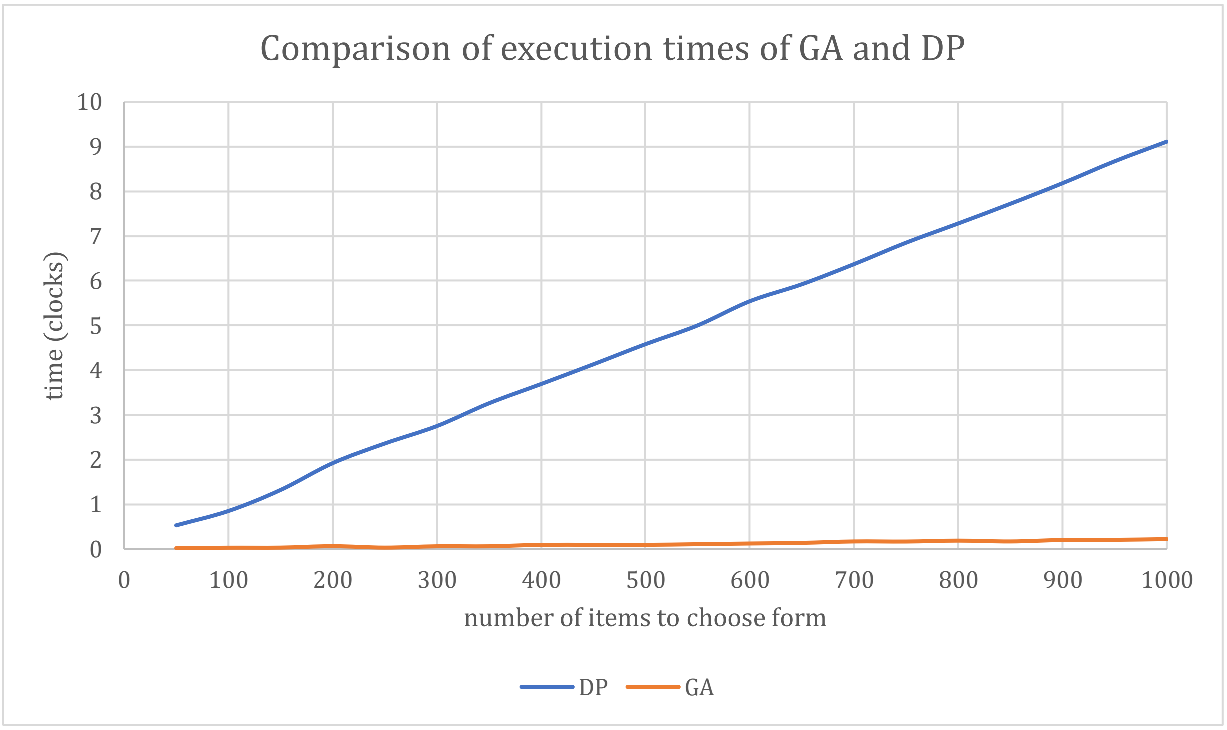
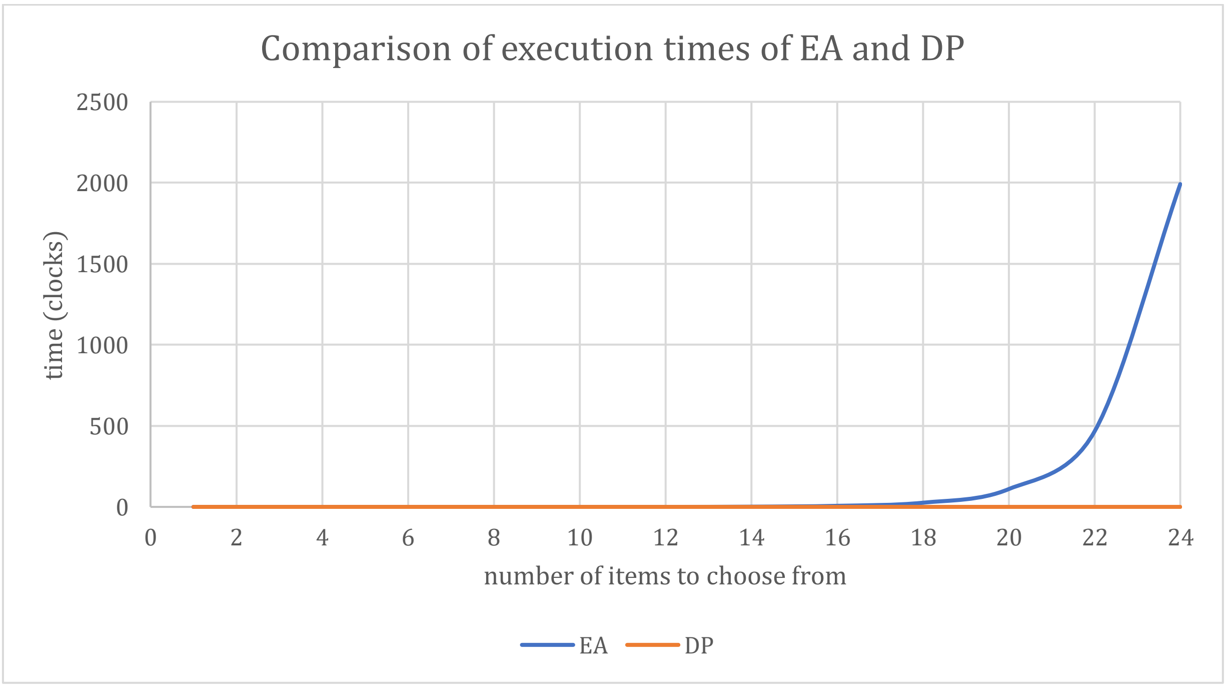


Chart 3

All of the above charts are not to be interpreted as comparisons, but as figures showing dependencies of the algorithm on s and n (or lack thereof).



Line chart 1 (constant knapsack size)



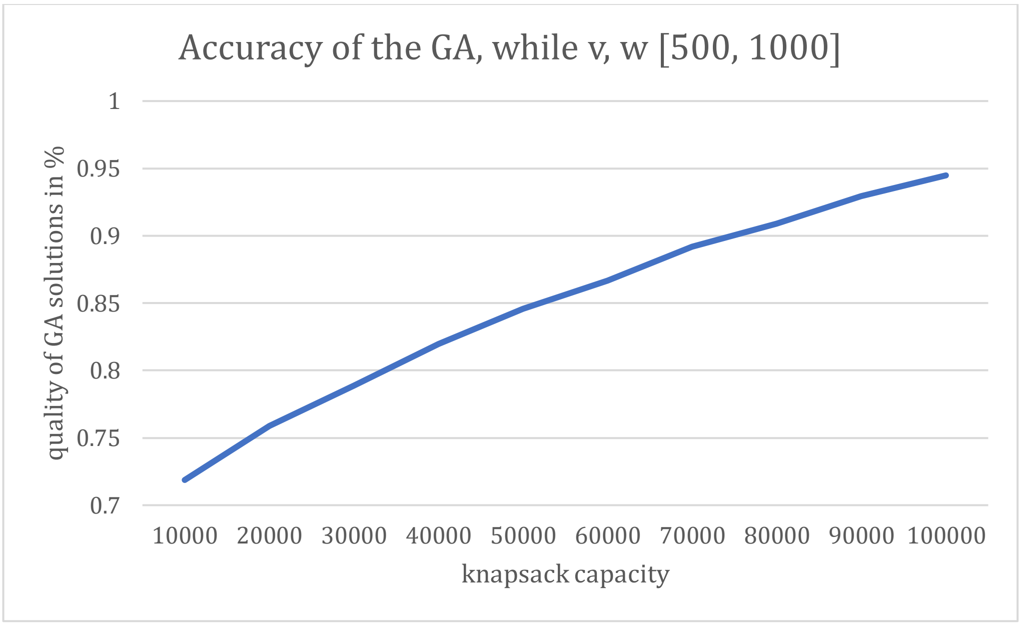
Line chart 2 (constant knapsack size)

As explained above – looking on *Line Chart 2* one can see that the time needed for the exhaustive algorithm to complete indeed increases exponentially – making it a far worse pick (way of solving the problem) than DP.

On line Chart 1, the execution times of DP and GA are compared, both of these output different types of solution, the latter one not guaranteeing an optimal result, but in problem case types where the GA performs well (such cases are mentioned in the second section) GA seems to be a decent and less time-consuming alternative.

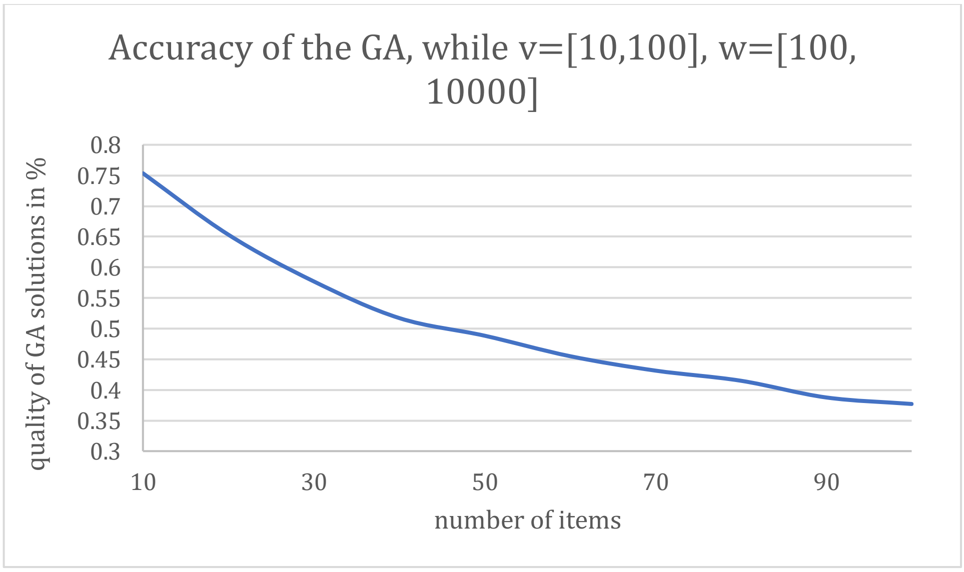
II Quality analysis

As stated, the Greedy algorithm performs better time-wise than both of the algorithms providing optimal solutions, in certain problem types - for example when the knapsack volume is great compared to the weights of the items, the greedy algorithm performs in an acceptable way. Line Chart 3 shows exactly this correlation – the bigger the knapsack, the more it is probable for it to fit more of the "top tier" items without a sacrifice in the form of picking a slightly better ratio-wise but heavier item and ending up with a lot of poorly packed space – this trap is shown on Figure 1.

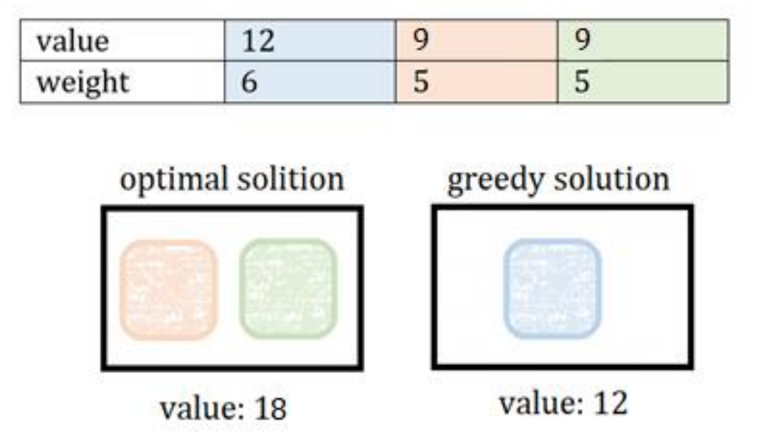


Line chart 3 – number of items is 400

On the other hand, if the weights of the objects are great and the value density does not fluctuate greatly – the greedy approach, performs poorly falling into many "traps" as described by Figure 1 – in this case as the number of items increases – the number of "traps" also does – causing a decrease in the quality of solutions.



Line chart 4 – knapsack capacity is 50000



value	12	9	9
weight	6	5	5

Figure 1 – the quality of GA's solution with the given items and maximum capacity 10 is 66%

In either of the cases presented, GA does not provide an optimal solution to the knapsack problem – it is only an approximation – in some online settings where an acceptable solution is needed the greedy knapsack could be an alternative, but as shown above this is greatly dependent on the number of items to choose from, their weights and the capacity of the knapsack.

Final remarks

The DP approach solves the knapsack problem in pseudo-polynomial time – depending on the knapsack capacity and the number of items, but a polynomial time algorithm is not known- which makes it clear that the problem belongs to the NP class (pseudo-polynomial algorithms have a polynomial running time in the *numeric* value of the input, but not necessarily in the *length* of the input). It is also to be stated that the knapsack problem is one of Karp's 21 NP-complete problems, and the subset sum problem which is proven to be NP-complete, can be reduced to it in polynomial time.