Homework 3 - Knapsack algorithms: experimental evaluation

Problem Statement

In this homework I tested different algorithms and observed their dependencies on various instance parameters. I did my evaluation by varying only one parameter at a time and fixing all others. During my experimental evaluation I tested two important values of algorithms: result quality (for approximate ones) and computational complexity.

Additionally, the robustness of the algorithms was evaluated to measure their sensitivity to the order of items by permutating them in each instance.

The instance parameters that I did my evaluations were:

1. instance size (# of items)
2. maximum cost
3. maximum weight
4. knapsack capacity to the total weight ratio
5. cost/weight correlation
6. granularity

Analysis and description of possible solutions

Brute Force – the algorithm iterates through all examples and branches out in two directions, either the item is taken, or it is not. It has one modification in a sense that when the capacity of the bag would become less than zero, that branch stops and returns its value.

Branch&Bound – this algorithm is similar to the Brute Force one with the addition of one more rule. Because it tracks currently best solution, it won't go to the branch where it knows that it cannot produce better result even if it takes all items. With that rule it speeds up the calculation by much but the complexity of the algorithm is still O(2^n).

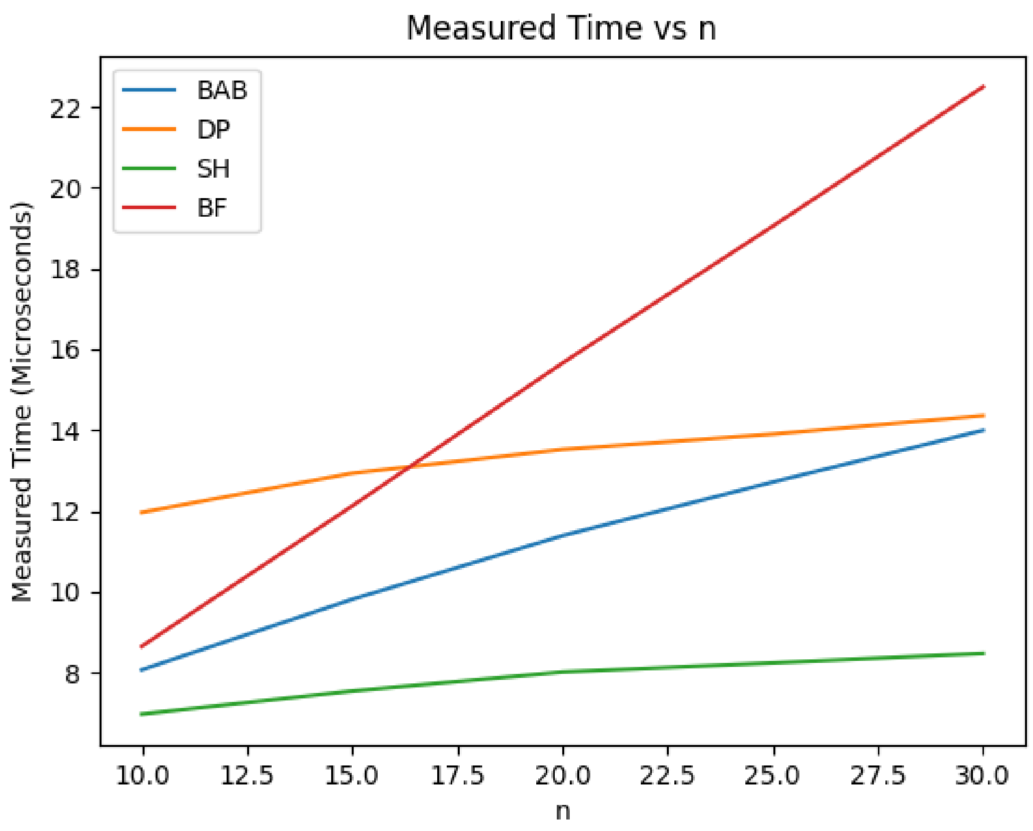
Dynamic programming – for DP I chose decomposition by total cost. It calculates the maximum achievable "cost" that fits within the given weight capacity. It iterates through items and cost states to update the table, ensuring the constraints of capacity and maximizing the cost.

Cost/weight heuristic – this algorithm sorts items by the cost/weight ratio and then inserts items, while the knapsack capacity is not exceeded.

Results and interpretations

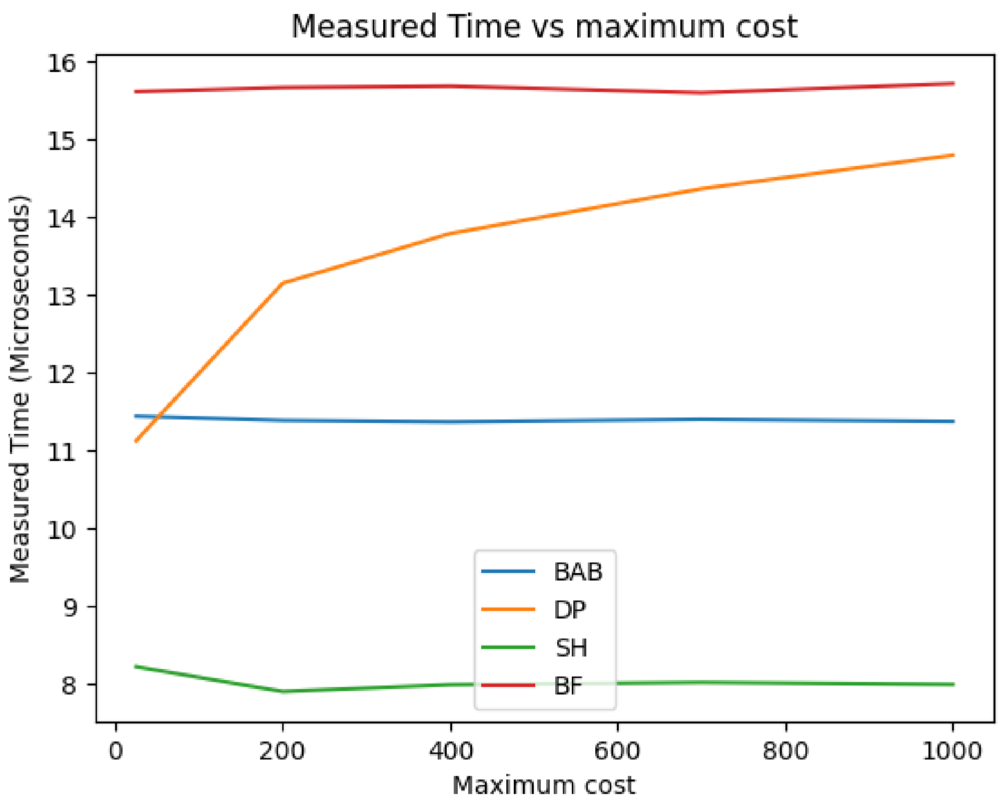
Computational complexity dependency on various parameters

Instance size



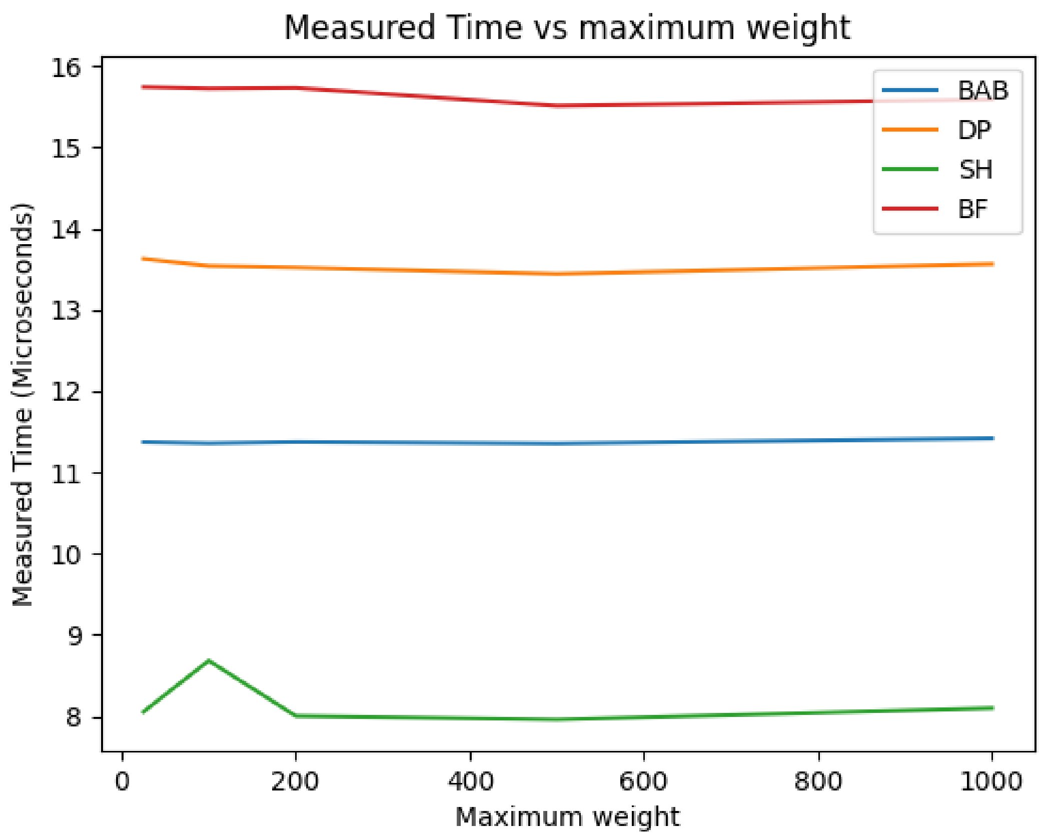
We can see that the most impact on instance size has brute force algorithm, as expected. Also, branch and bound has some impact. Dynamic Programming and heuristic avoid exploring all possible subsets so that is why they are not impacted a lot.

Maximum cost



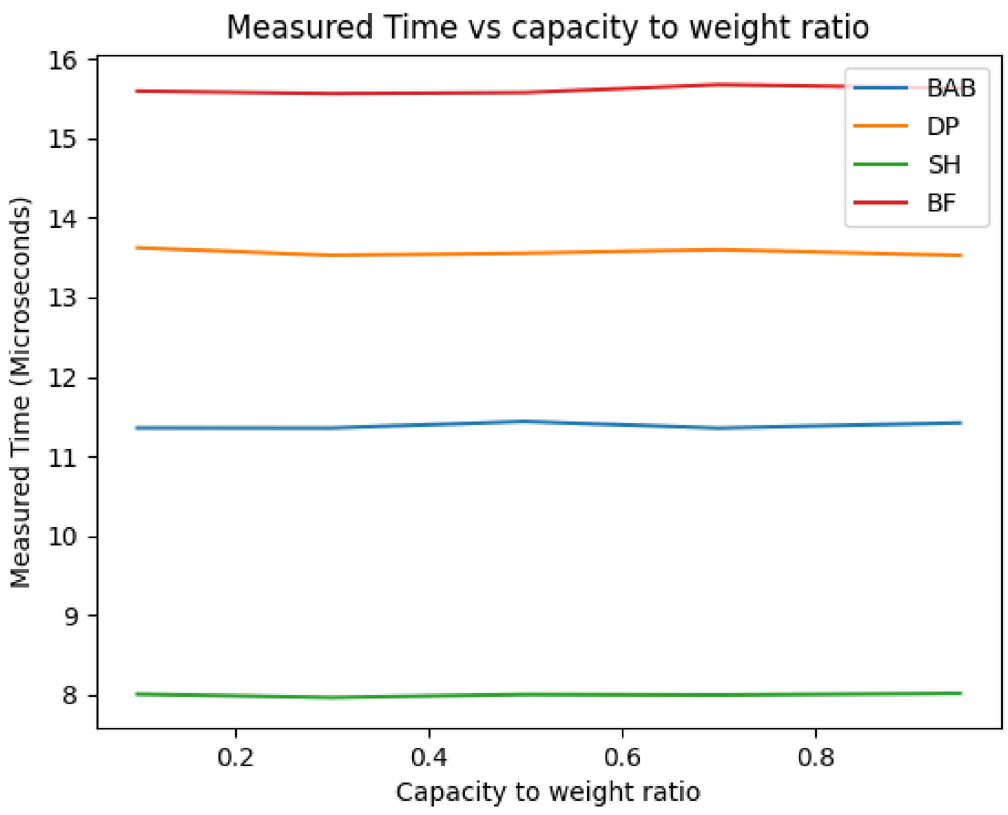
We can see that only dynamic programming algorithm is sensitive to setting maximum cost. It makes sense since its complexity depends on the maximum cost item. In contrast, other algorithms like Brute Force, Branch & Bound, and cost weight heuristic do not use the cost range directly in their computation.

Maximum weight



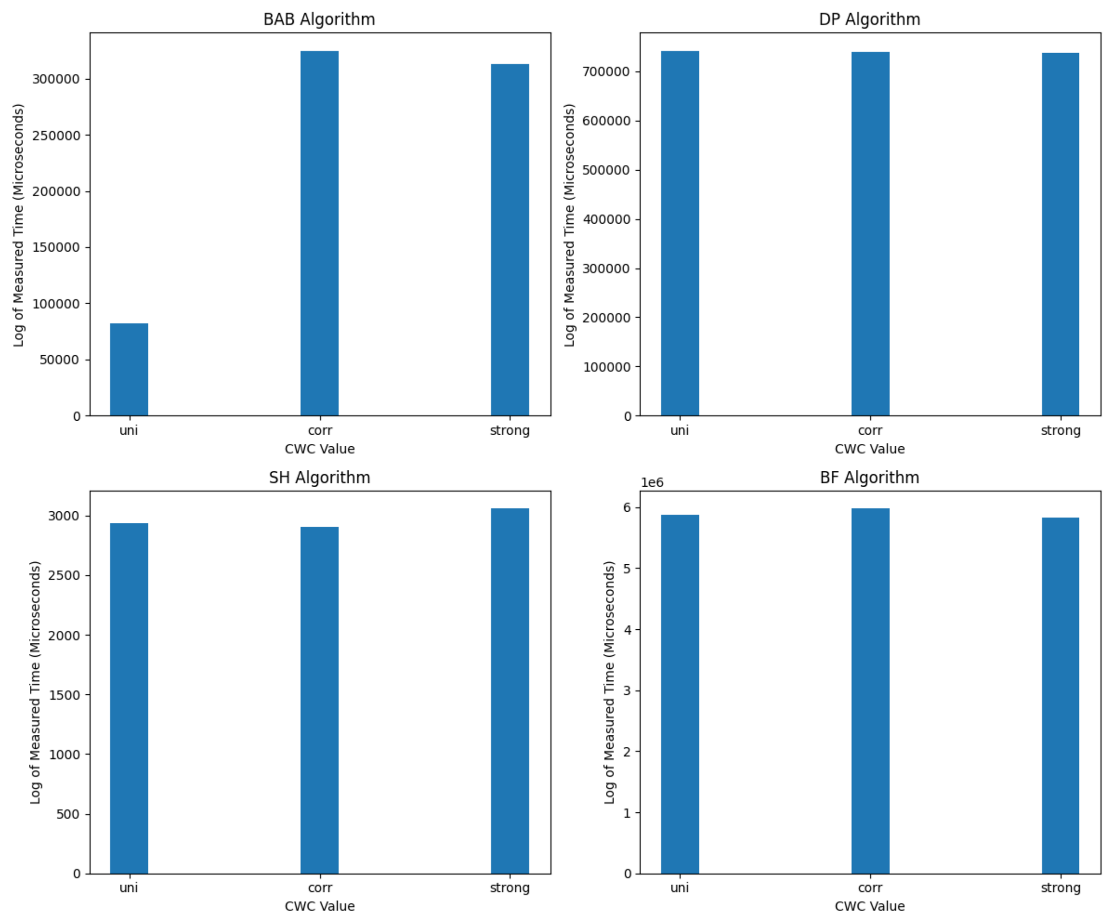
Maximum weight seems not to impact much any of the algorithms. Brute Force and Branch & Bound consider all subsets, regardless of individual weights, as long as they meet the capacity constraint. Dynamic programming does not rely on the magnitude of individual weights. Heuristics operate on cost/weight ratios, which is not affected by change in only weights (e.g., doubling all weights doesn't affect the order of ratios).

Capacity to weight ratio



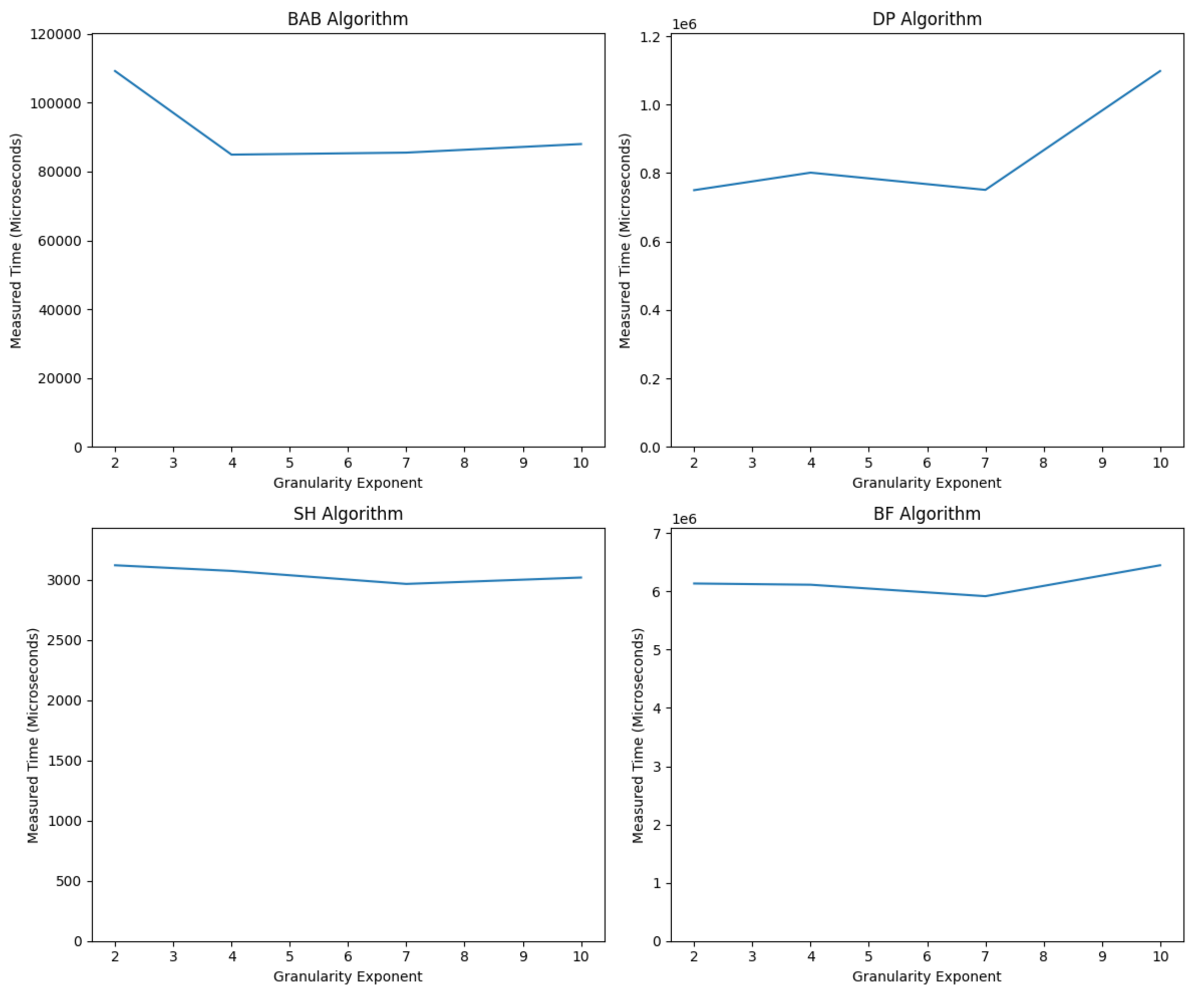
I could not find any dependency of any algorithm on capacity to weight ratio.

Cost weight correlation



We can see that only branch and bound algorithm is greatly impacted by the cost weight correlation. It leverages upper bound (e.g., maximum potential value in a branch) to prune unnecessary explorations. Highly correlated cost and weight values reduce the ability of the algorithm to prune branches effectively.

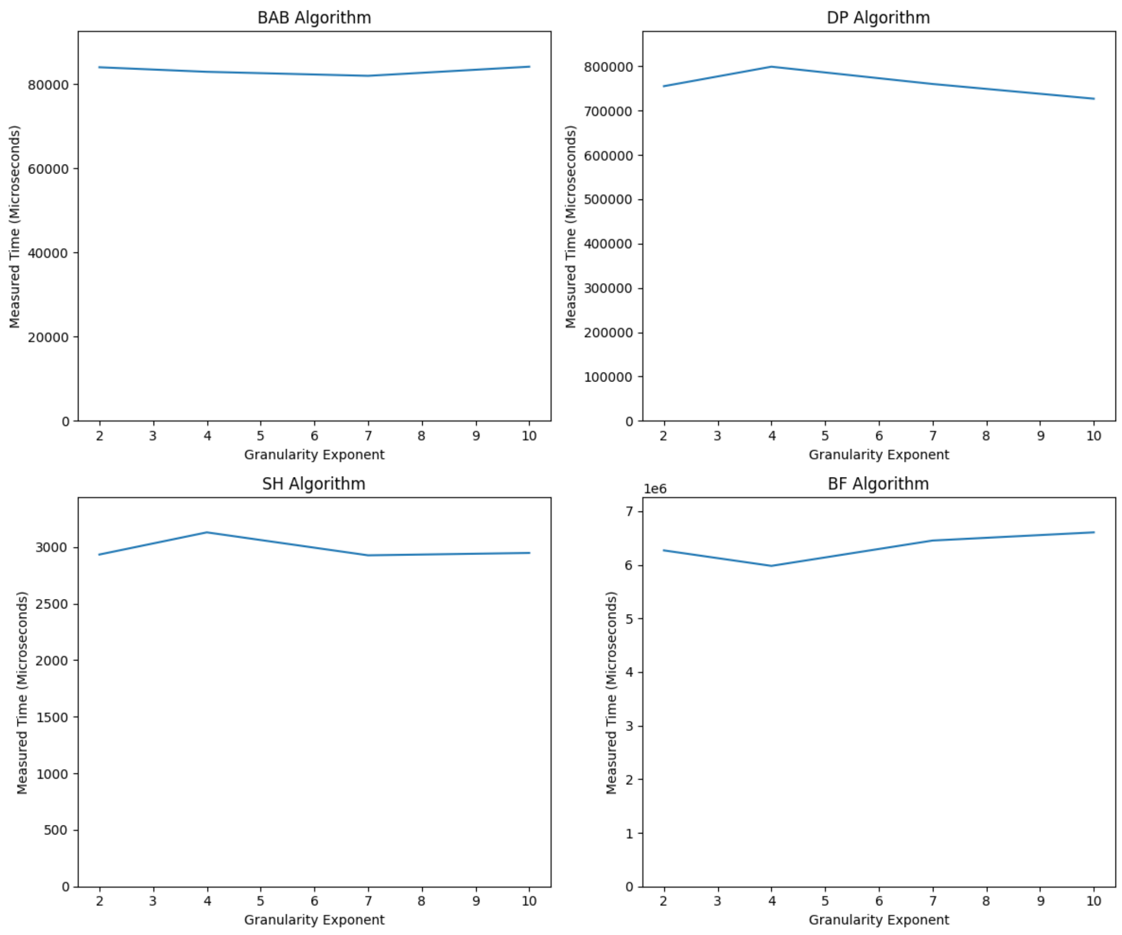
Granularity



This graph is for mostly heavy items. We can see that branch&bound and dynamic programming algorithms are affected by it.

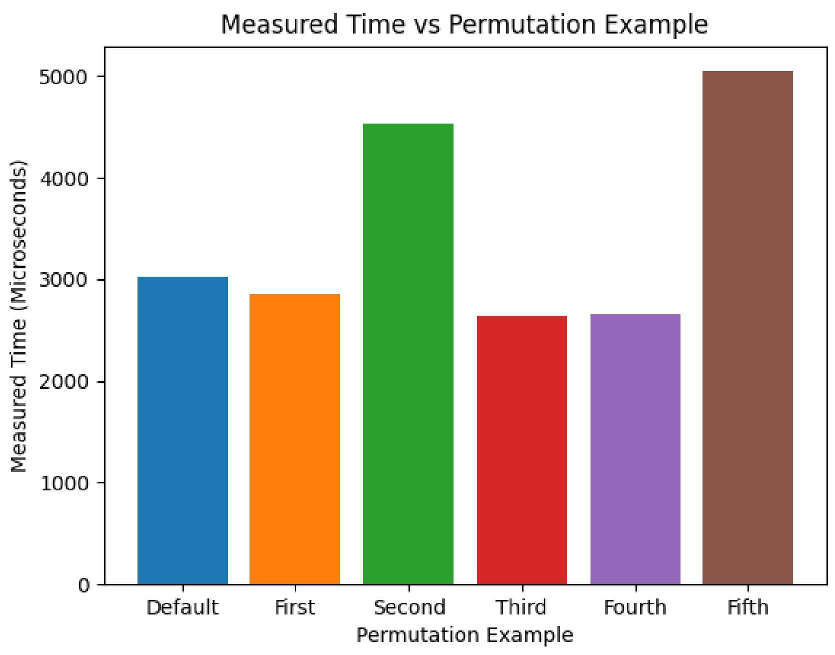
For B&B it decreases as there exist more heavy items. It makes sense since it speeds up greatly and cuts down lots of branches from unnecessarily being explored.

For dynamic programming it increases runtime since DP table becomes bigger and bigger because it depends on the total weight capacity.



This graph is for mostly light items. There does not exist big impacts on the algorithms. With light items, the granularity of the weight states is higher, but each individual item has less impact on capacity. This reduces the chance for large, sudden changes in the state space for any algorithm.

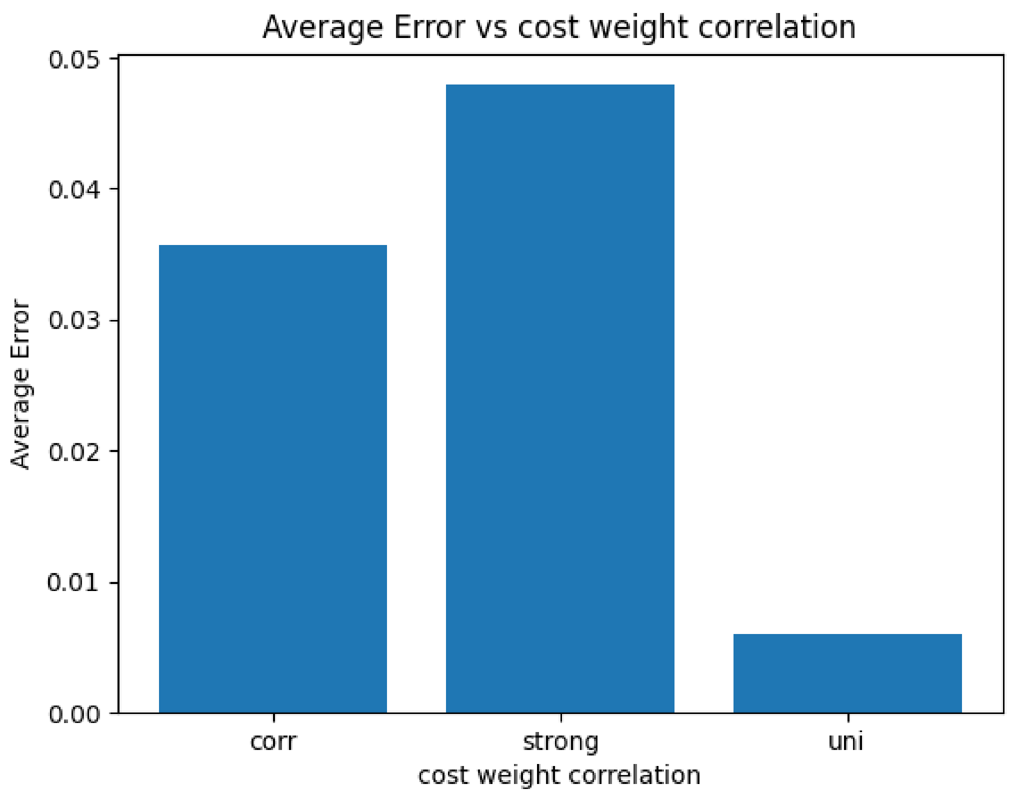
Robustness (permutations)

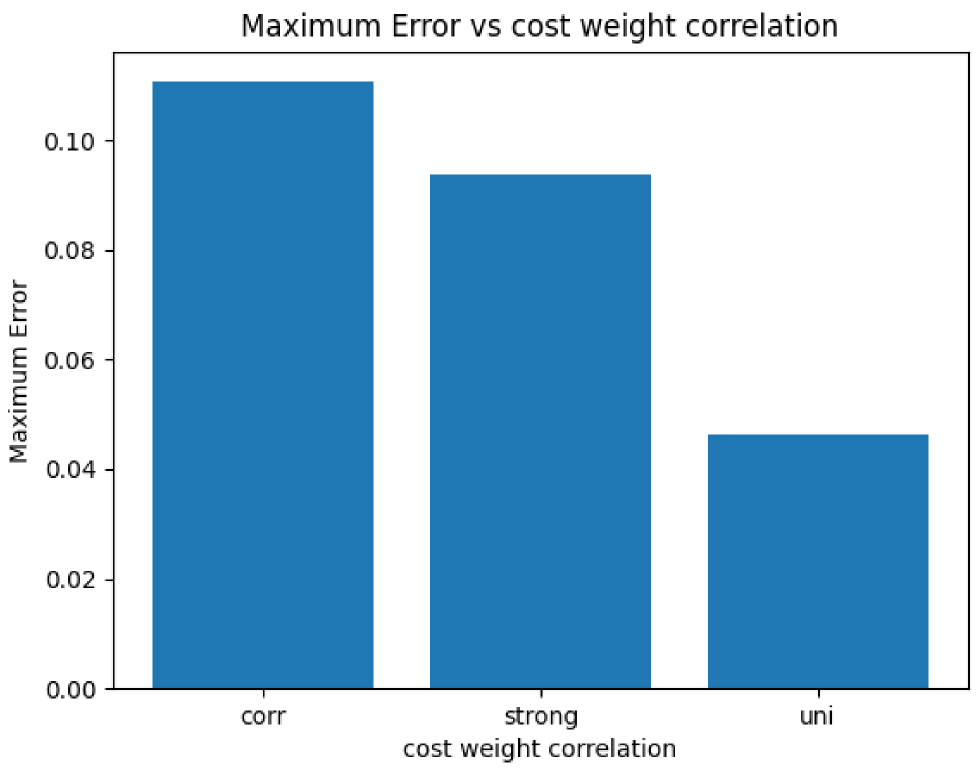


This graph shows different permutations of same instance for simple cost/weight heuristic algorithm. We can see that it can really change runtime based on the order of items.

Result quality (for cost/weight heuristic)

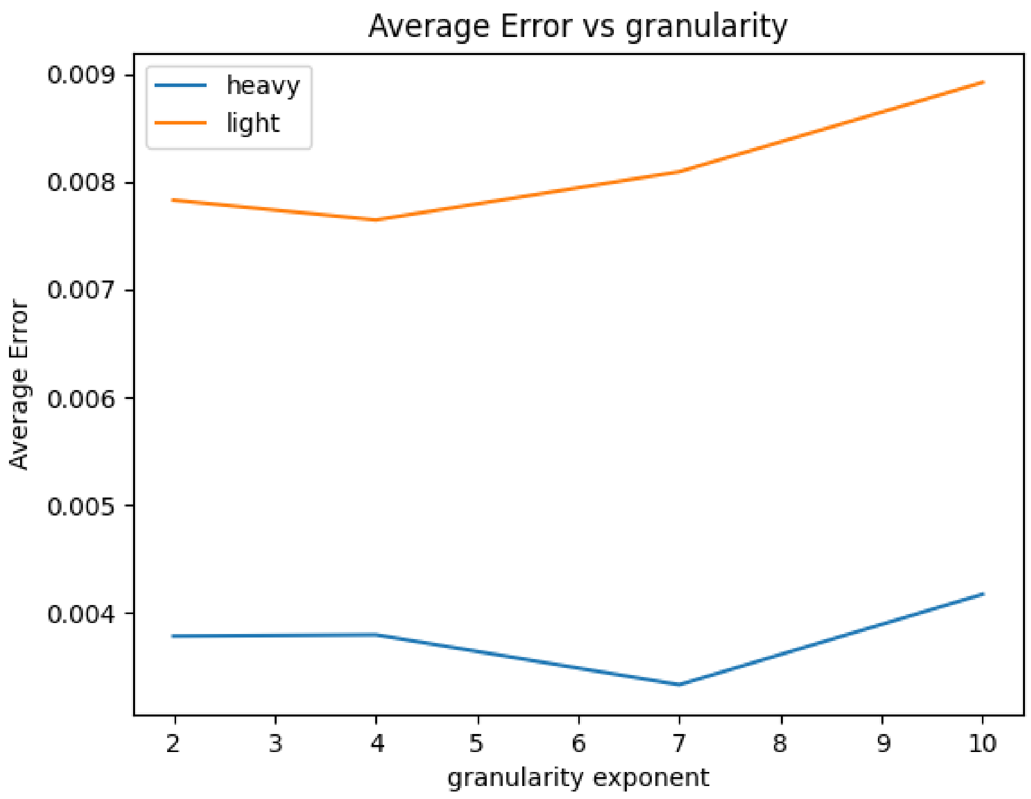
Cost weight correlation

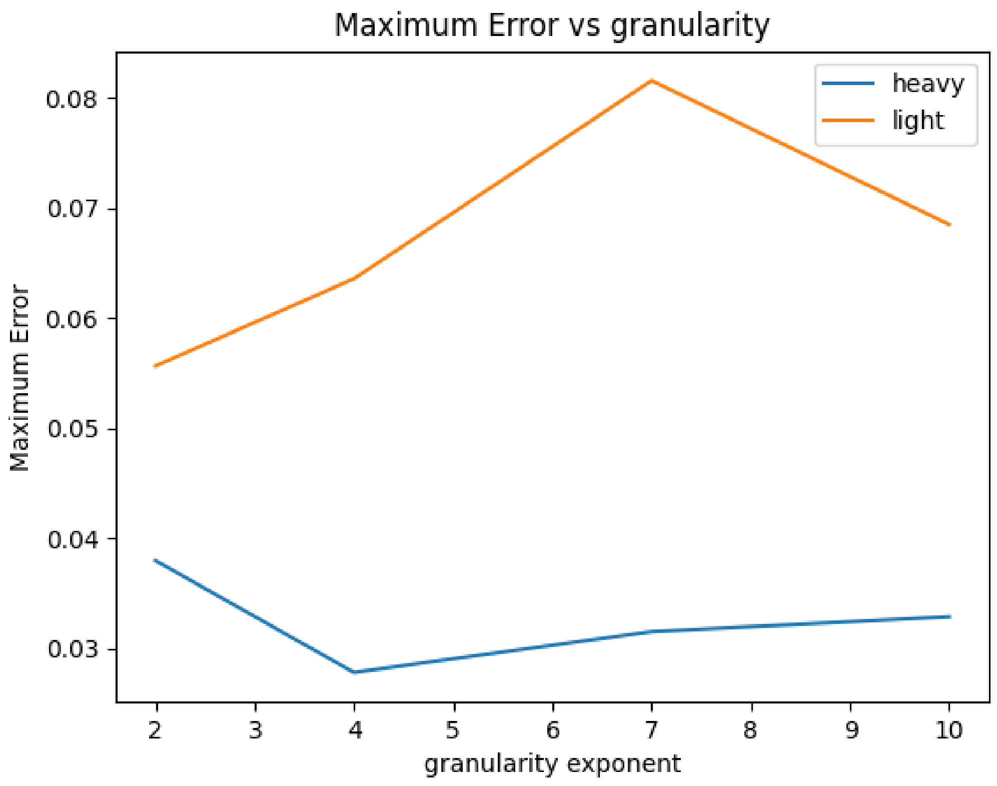




Average error increases greatly when we change cost/weight correlation to have either big or small correlation. When costs and weights are strongly, the heuristic is more likely to make greedy choices that approximate the optimal solution. When costs and weights are uncorrelated, the heuristic's greedy strategy often fails to prioritize the correct items, leading to greater error.

Granularity





We can conclude that both average and maximum error are bigger when there are majority of light items. For light items, the cost/weight ratios tend to vary widely. This variability exaggerates the heuristic’s reliance on ordering, leading to greater average and maximum errors when compared to heavy items, which have more stable ratios.