Combinatorial Optimization
Homework 3 – Experimental Measurements

Matyáš Skalický
skalimat@fit.cvut.cz

November 25, 2021

Contents

1 Measured Algorithms

Baseline exact method is the **brute-force** algorithm that recursively iterates explores all possible solutions without any speedups or pruning.

The **branch&bound** extends the brute-force with 2 speedups. We stop when the current candidate already exceeds the capacity of a bag. Also, we don't recurse further when the cost sum of the items that can be added into the bag is lower than the best found solution.

The dynamic programming approach is based on the **decomposition by cost**. This solution is based on memoization by the recursive function calls by returning the maximum value from the tested branches on return.

The **greedy** heuristic simply adds the items with the highest cost/weight ratio until the capacity of the bag is reached. This is the only algorithm that doesn't always result in the optimal solution.

2 Measured Variables

For the pilot experiments, I've chosen to benchmark all algorithms on the instances of size 24. The generator's maximum weight W is set to 3000 as well as maximum price C.

In the following experiments, we will try to measure how time and the solution quality depend on the generator inputs. Also, we will look how robust each algorithm is against permutations in the algorithm inputs.

- Ratio of bag capacity to summary weight. (cw_ratio $\in [0, 1]$)
- Correlation between price and weight. (pw_corr $\in \{uni, corr, strong\}$)
- Granularity and distribution of weights. (w_dist $\in \{light, bal, heavy\}$)

3 Experiments

We will measure the effect of each variable separately. We will generate 10 problem instances for each algorithm with each weight.

3.1 Robustness

First, we will try to answer the robustness (invariance to order of input items) of each algorithm. We will use default generator values mentioned above. We will generate 500 instances and calculate 10 permutations for each algorithm. We will calculate the variance in the elapsed time across the 10 permutations and take a mean across all generated instances.

method	mean runtime	mean runtime variance
greedy heuristic	0.00006964	0.00000001
branch&bound	0.00642724	0.00001817
dynamic (cost)	0.38064328	0.01843123
bruteforce	9.22061772	1.95676123

Table 1: Mean of runtime (seconds) and runtime variance over 10 permutations

It is not surprising, that the heuristic is extremely stable as it is basically just one pass over the items in the bag. I have expected that the algorithms such as branch&bound and dynamic would be less robust as especially the first one depends on the order of items while pruning. Yet the dynamic algorithm seems unstable (not robust to the item premutations) as well.

It is surprising that the bruteforce algorithm shows such huge mean variance in the measured runtime. But it is also worth noting that the mean runtime for bruteforce was 9.22 seconds while it was only 0.38 and 0.006 seconds for dynamic and branch&bound respectively.

I would say, that since the measured metric was the runtime and the experiments were performed on a 40-core CPU in parallel, it could be caused by uneven load across CPU cores as different tasks were running on the CPU as well. But I still find it very interesting.

3.2 Pilot Experiments

We will not measure the permutations in the pilot experiments. Figure 1 contains the absolute Pearson correlation between the runtime in seconds and changing parameters in the instance generator.

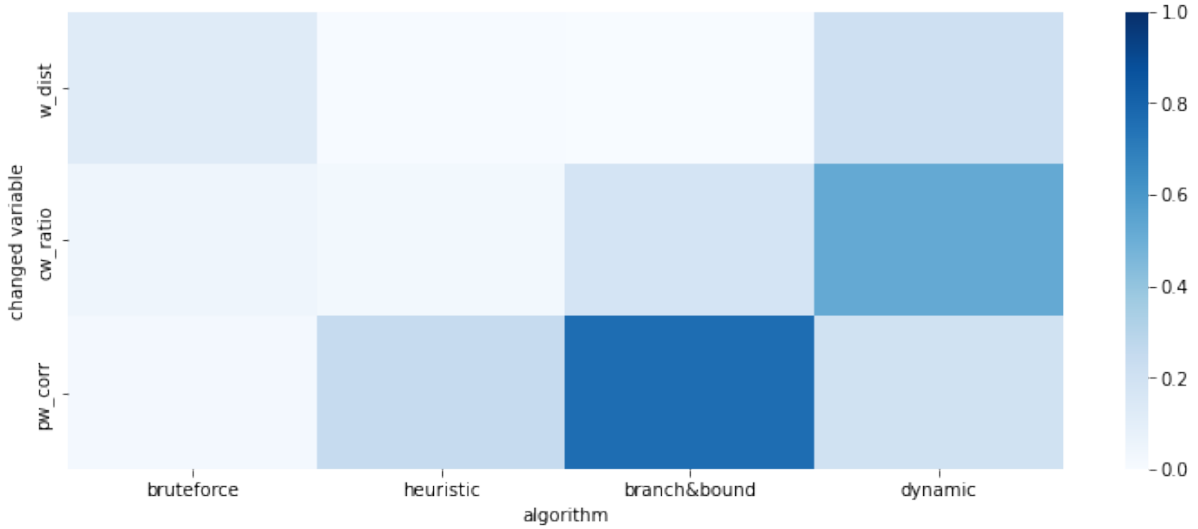

Figure 1: Correlation (absolute) of the algorithm runtime to the change in parameters

3.3 Detailed Experiments

Based on the Figure 1, we can see that there is a strong correlation between *dynamic* and cw_ratio. We will further inspect the relation of *dynamic* and w_dist. For the *branch&bound*, we will inspect the variables cw_ratio and pw_corr as they has shown a small correlation with the algorithm runtime.

All variables were tested separately, 100 samples per each testing instance size.

3.3.1 Weight Distribution (w_dist)

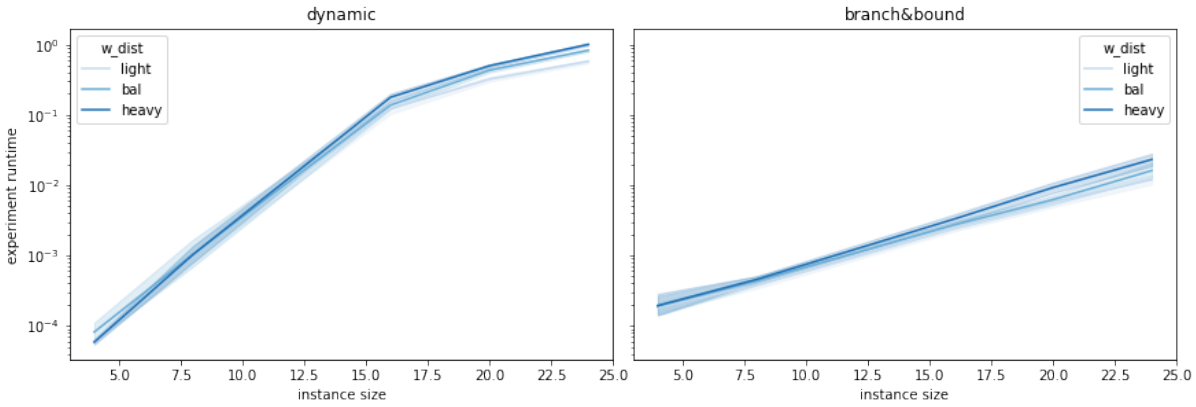

Figure 2: Weight distribution [light, bal, heavy]

Weight distribution affects whether the majority of the items are light, heavy, or balanced. As indicated by the pilot experiments, the weight distribution (as seen in Figure 2) does not heavily the experiment runtime.

3.3.2 Price and Weight Correlation (pw_corr)

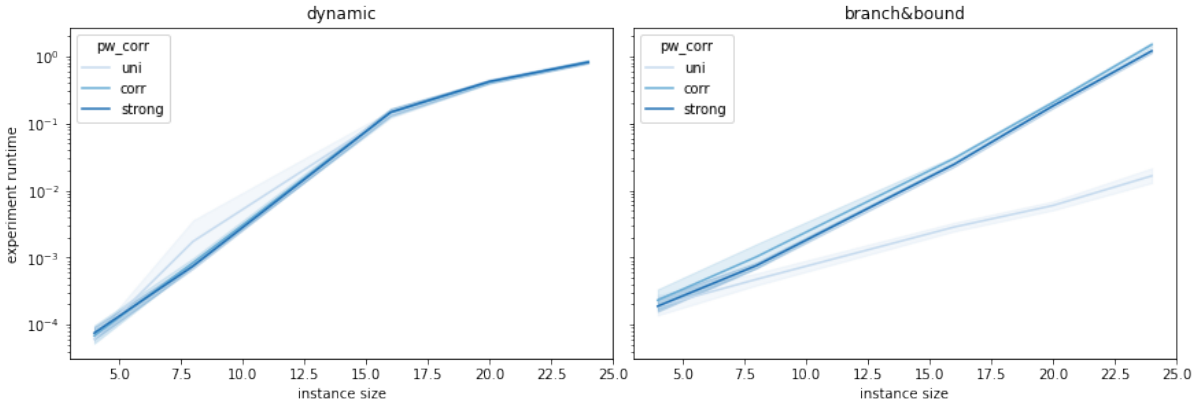

Figure 3: Price and Weight Correlation [uni, corr, strong]

As shown in the Figure 3, dynamic programming isn't affected by the changing correlation between price and weight (pw_corr). On the other hand, we can see that branch&bound is affected strongly as the processing speed of the bags with correlated items is much slower. This is likely due to the pruning being less effective.

3.3.3 Capacity and Weight Ratio (cw_ratio)

The cw_ratio is defined as the ratio of max knapsack capacity to total weight. The pilot experiments had indicated a correlation between the runtime and this variable. Figure 4 sheds more light to this hypothesis. Both dynamic and branch&bound algorithms are affected by the changing cw_ratio.

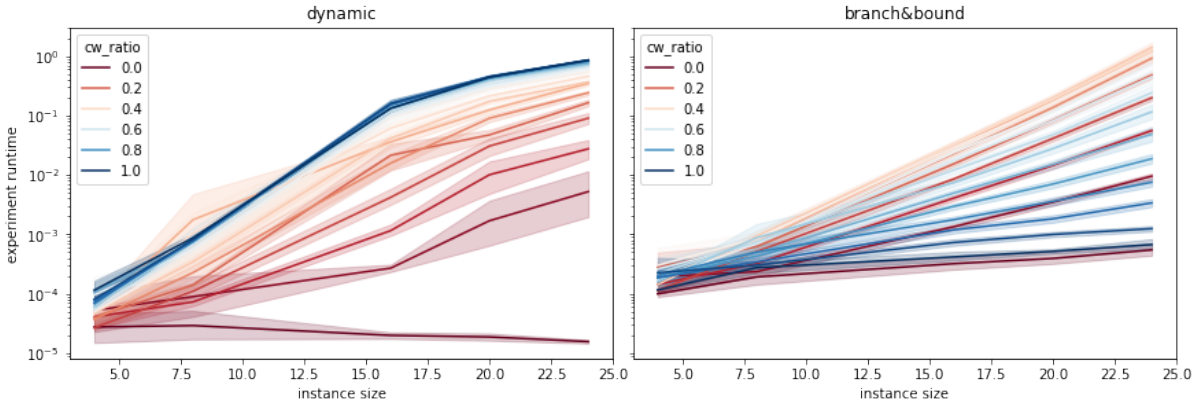

Figure 4: Ratio of max knapsack capacity to total weight

As seen from the Figure 5, the dynamic algorithm is strongly affected if cw_ratio is closer to 1. The relationship for the brach&bound isn't linear. The algorithm is the fastest both, for bags which were on the sides of the spectrum.

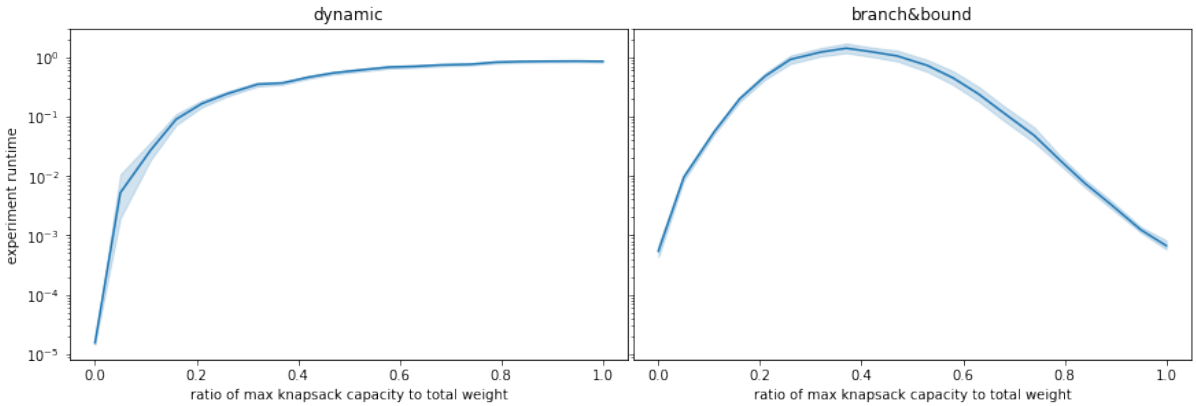

Figure 5: ratio of max knapsack capacity to total weight for instance size 24

4 Discussion and Takeoffs

I've first ran the pilot experiments on the small data samples to detect potentially interesting relationships between generator hyperparameters and the experiment runtime. Based on this, I've explored the effects of the parameters on larger data samples as mentioned above to see the relationships in a better detail.

One of the interesting takeoffs for me was the large variance in the brute-force algorithm when permuting the items. As the brute-force algorithm always searches through the full searchspace, the time taken should be very similar. My intuition about this is that this must be due to uneven CPU thread scheduling.

Overall, I would say that this homework took a long time to create, yet I wish I was doing anything useful. Instead of just replicating the experiments that were done 1000 times before and don't benefit anyone in doing again in our homeworks.