Combinatorial Optimization
Homework 3 – Experimental Measurements

Matyáš Skalický
skalimat@fit.cvut.cz

November 22, 2021

Contents

1 Measured Algorithms

Baseline exact method is the **brute-force** algorithm that simply iterates all possible solutions without any speedups.

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 $\in [0, 2]$)
- Correlation between price and weight. (pw_corr $\in [0, 1]$)
- Granularity and distribution of weights. (w_dist $\in [0, 2]$)

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 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 to be not very robust 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.

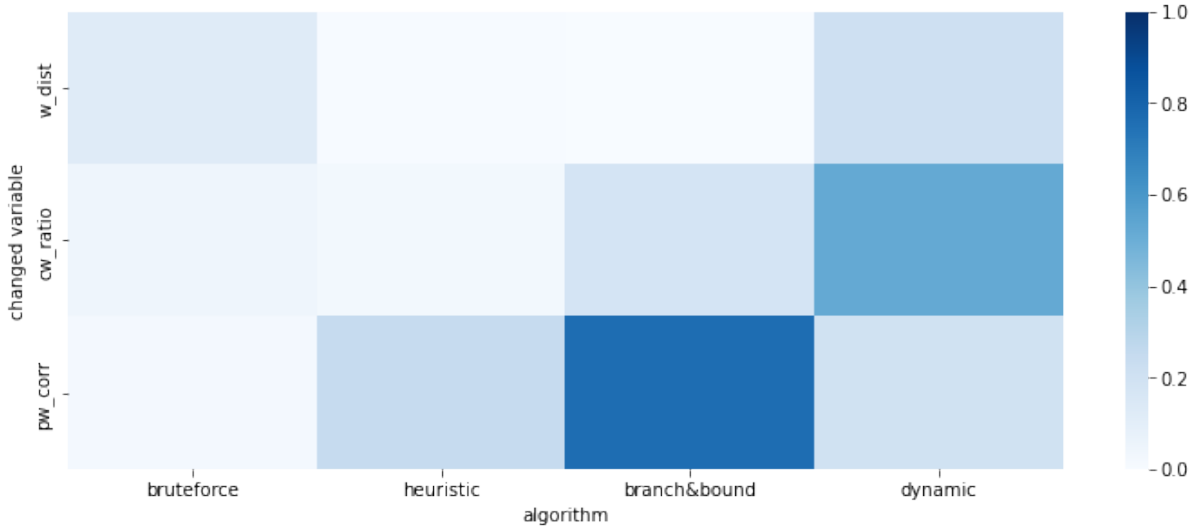

Figure 1: Correlation of the algorithm runtime to the change in parameters

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)

As expected by the pilot experiments, the weight distribution does not heavily affect the elapsed time.

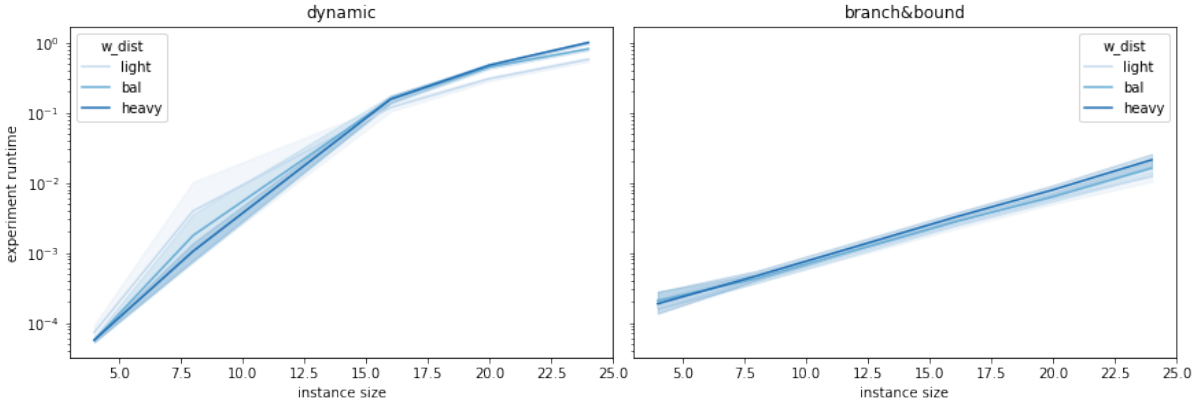

Figure 2: Weight distribution [light, bal, heavy]

3.3.2 Price and Weight Correlation (pw_corr)

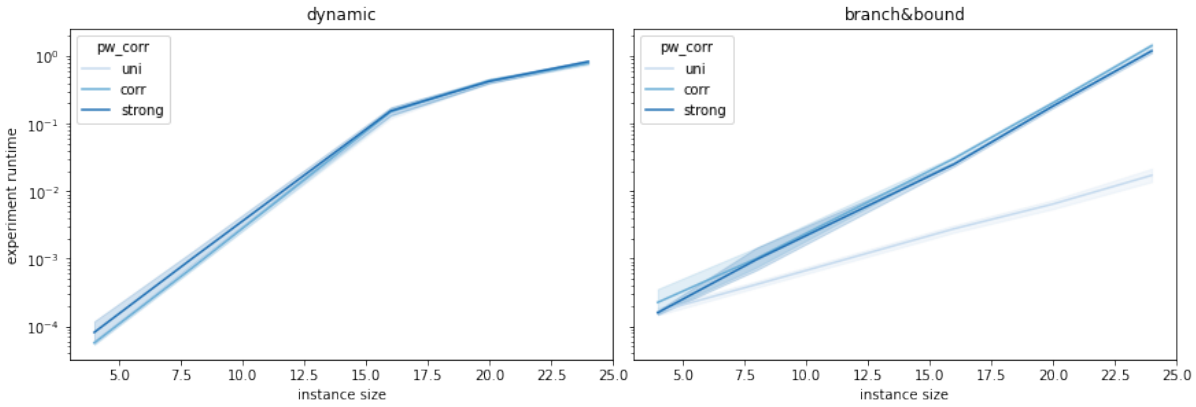

Figure 3: Price and Weight Correlation [uni, corr, strong]

3.3.3 Capacity and Weight Ratio (cw_ratio)

4 Discussion and Takeoffs

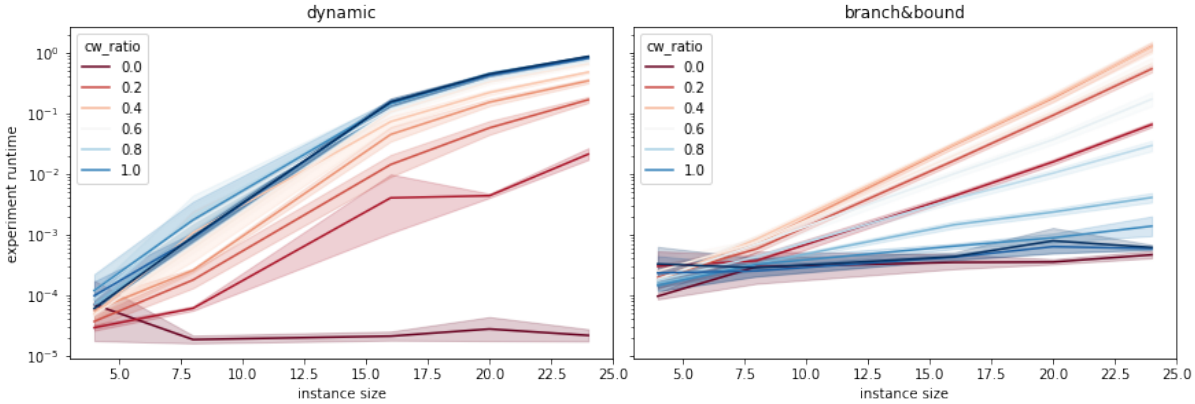

Figure 4: Capacity and weight ratio

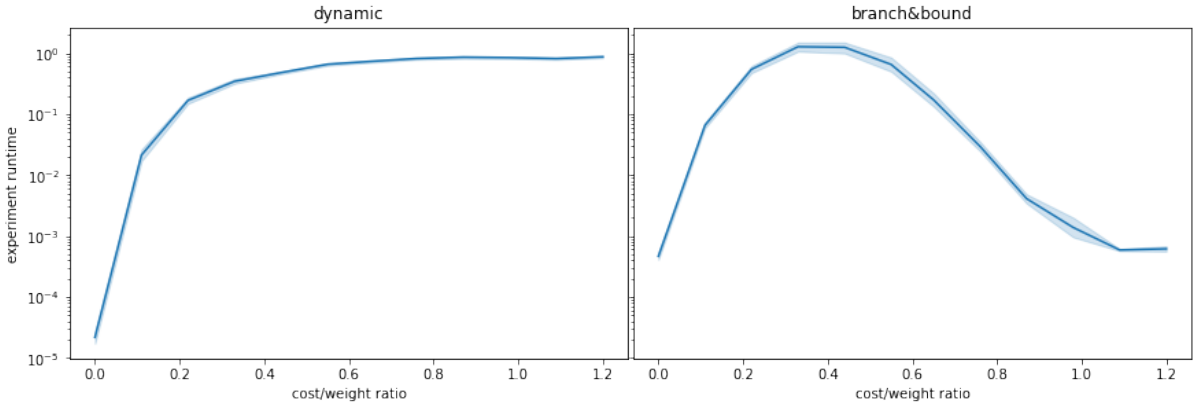

Figure 5: Capacity / weight ratio for instance size 24