Combinatorial Optimization
Homework 1 – Knapsack Problem

Matyáš Skalický
skalimat@fit.cvut.cz

October 14, 2021

Contents

1 Implementation

The brute-force solver of the knapsack problem was implemented using the recursive approach. The solver was implemented in Python. To speed the experiments up, they were executed in parallel to fully utilize the CPU. The datasets were subsampled from 500 to 250 samples to speed up the experiments.

The naive brute-force search recursively enumerates all possible solutions until a feasible solution is found, or the solution space exhausted.

An optimized branch&bound solver was also implemented and compared to the naïve brute-force approach. It utilizes several speedups:

exceeded weight Weight of the bag already exceeds the capacity of the bag. Do not recurse further.

residuals mincost Sum the cost of remaining items that can be added into the bag. Do not recurse further if (cost + residual) $<$ mincost where *mincost* is the decision version requirement.

2 Experiments

2.1 Instance Size

First, let's compare the algorithms on a mixed dataset (half from the NR and half from the ZR dataset). Figure 1 shows that the number of performed operations grows exponentially with the size of the instance.

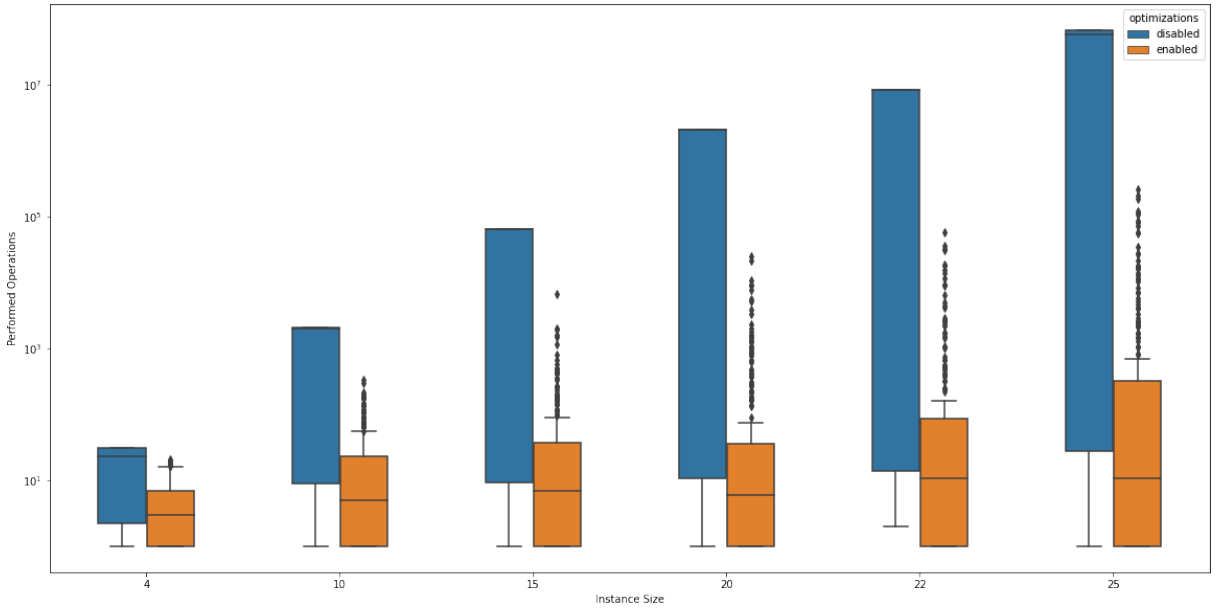

Figure 1: Instance size vs. number of performed operations

Let's break down the number of operations by the dataset. Figure 2 compares the required number of operations by dataset and also by solver. We can see, that when branch&bound is disabled, the complexity is the same for both datasets. When we enable the branch&bound, the speedup is most notable for the *NR* dataset.

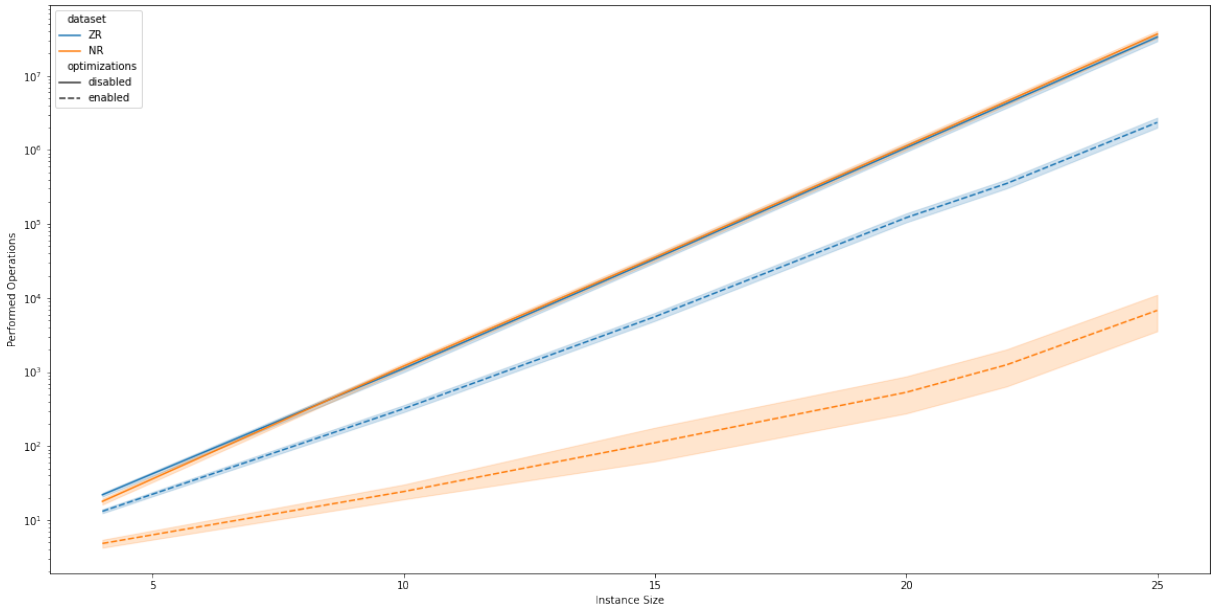

Figure 2: Instance size vs. number of performed operations by dataset

2.2 Histogram

Following Figure 3 highlights the finding, that the branch&bound algorithm speedup significance was accented mainly on the NR dataset.

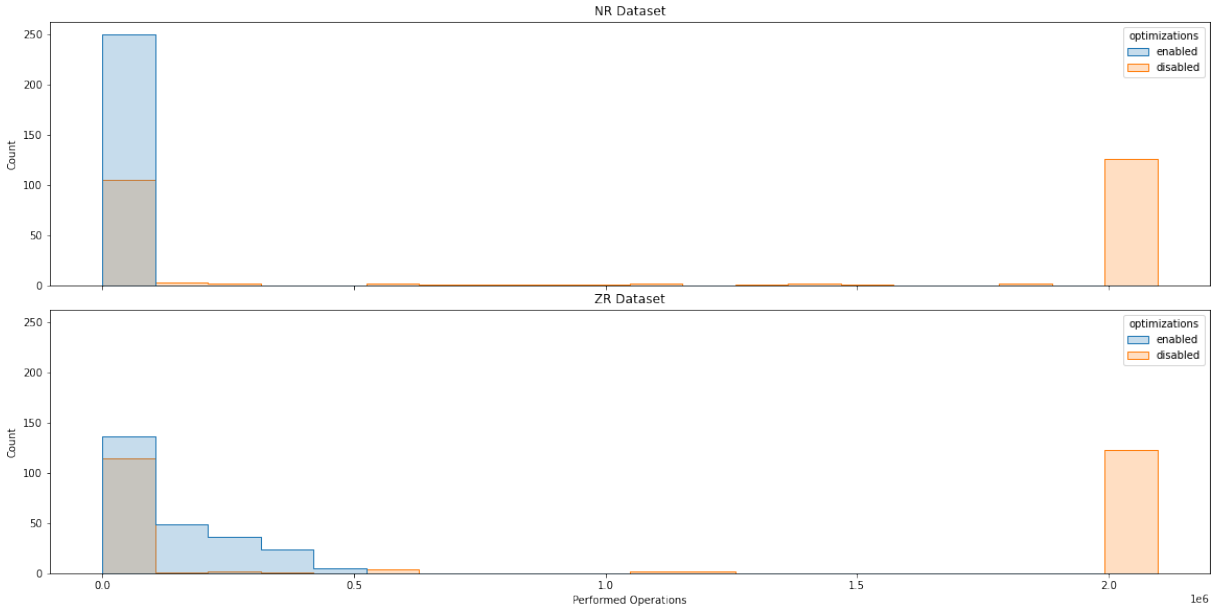

Figure 3: Histogram for number of operations; breakdown by dataset type and optimizations

3 Discussion

The *ZR* dataset is most-likely designed in a way to combat the usage of branch&bound speedups. My assumption was, that the branch&bound technique will be more effective, but overall, it should grow asymptotically with the brute-force approach.

When the optimizations were disabled, the computational cost was almost identical for the both methods (best seen on Figure 3). When using the branch&bound technique, there was a significant difference between the provided datasets as the *ZR* dataset forced the solver to often explore all possible solutions.

Since we are using a decision version of the knapsack problem, without branch&bound we need to explore all the possible solutions before saying that there isn't one. This is clearly visible on the right side of the Figure 3.