Exaktní Metody Řešení Rozhodovacího Problému Batohu

Úkol:

1) Experimentálně vyhodnoťte závislost výpočetní složitosti na velikosti instance u následujících algoritmů **rozhodovací verze** 0/1 problému batohu:

- **hrubá síla**
- **metoda větví a hranic (B&B)**

2) Otázky, které má experiment zodpovědět:

- Vyhovují nejhorší případy očekávané závislosti?
- Závisí střední hodnota výpočetní složitosti na sadě instancí? Jestliže ano, proč?

Řešení:

In order to solve the task I have designed and implemented a universal toolchain named KNAPSACKer. The precise process of building and execution is described in details in the README.md file. The very toolchain is adjustable and has a great potential of further improvements.

How does KNAPSACKer work?

KNAPSACKer reads all the files with knapsack data one-by-one and line-by-line from the directory specified by user. Each line is passed to and processed by a C++ executable (works with lines, not files) which computes the solution according to specified computational methods and type of the task, measures the computation time and prints the information to stdout.

The output is being redirected to the newly created directory, to the file with unique filename generated according to the time of file creation and computation parameters.

When the C++ executable has processed all the lines, the statistics about the file is being collected (min, max and average computation time) and passed with the exact file contents to the visualization tool that generates a plot of computational time distribution which is stored in the file directory. This allows user to process the same file in different time with different computational preferences and store the previous info to further comparison.

What was done?

After developing the tool, I have run it four times simultaneously for all of NR and ZR sets, using the both of Branch & Bound and Brute Force methods for each one. The tool was running for almost 12 hours (11th Gen Intel© Core™ i7-1165G7 @ 2.80GHz × 4), however, the only computation that was fully completed was the “NR set using B&B method”. All other configurations (NR & ZR using Brute Force, and ZR using B&B) were still in process of computation (size of current set was 35, 37 and 40 correspondingly). All the generated data is located in the project directory.

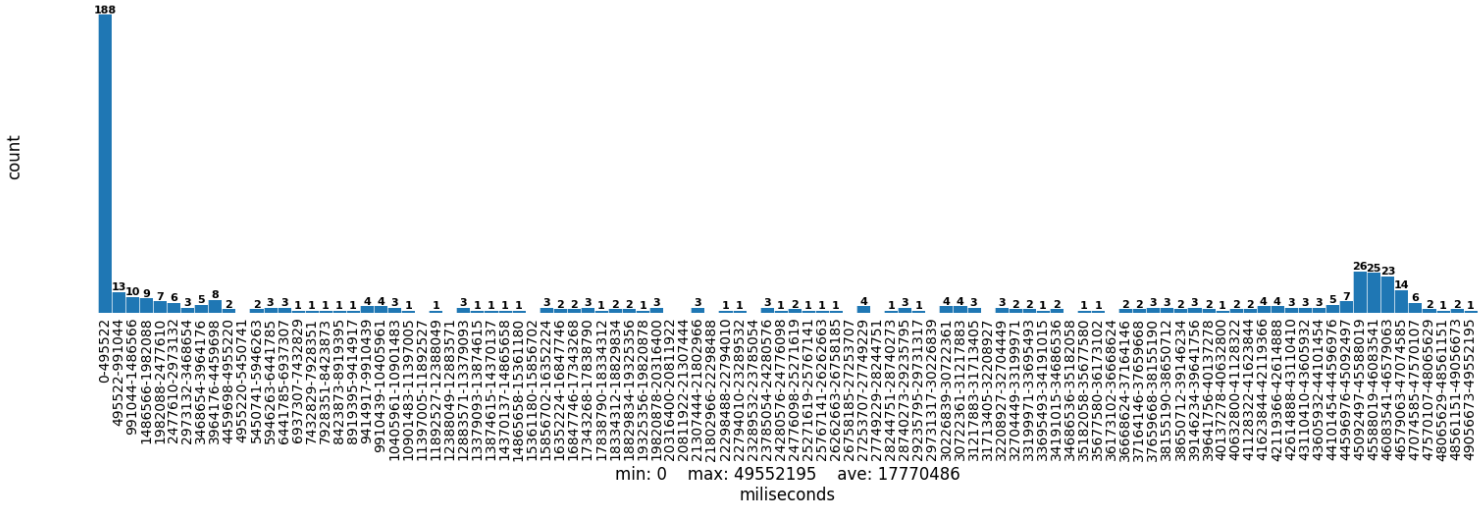
Results

This section consists of general description of used methods, observations analysis and proper conclusions. All the plots and source codes can be found in the project folder.

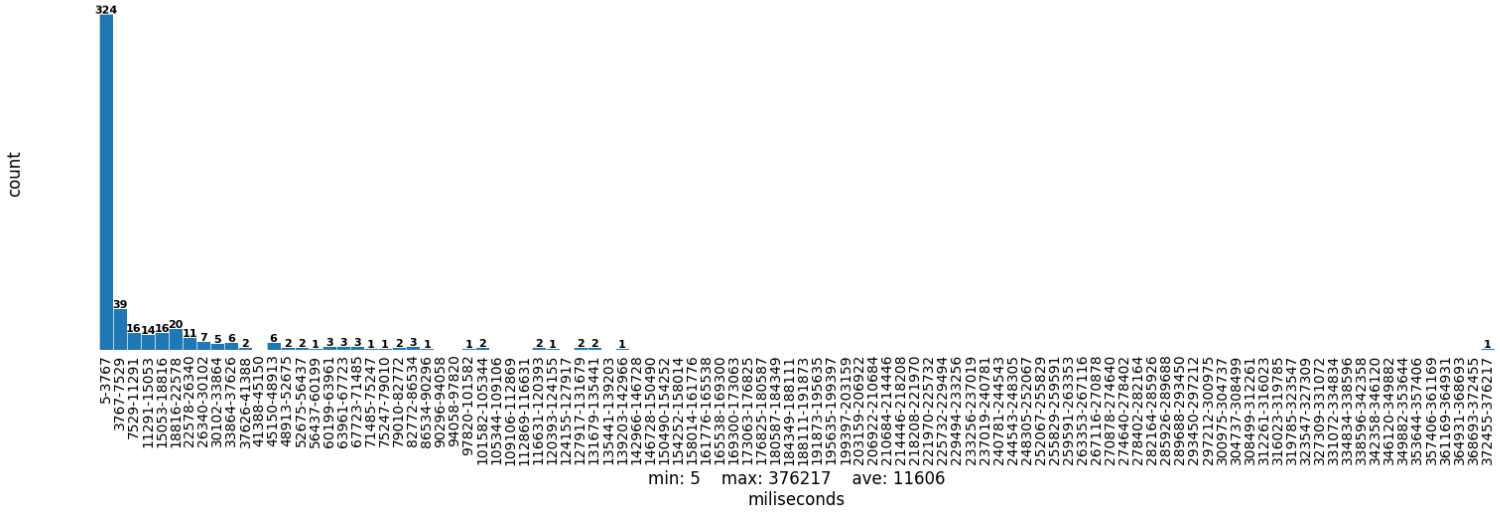
Observations:

1) The distributions of all of the obtained computational time of NR sets resemble Exponential Distribution in no dependence on the computational method (B&B or BF) or size of a set. However, the BF method produces much more bias in the density distribution of results (shift of records to the right).
CONCLUSION: the numbers in the set were generated artificially according to some predefined generation function, since all the sets match the same pattern. In the case of randomly generated sets, the distributions resemble Standard Normal Distribution.

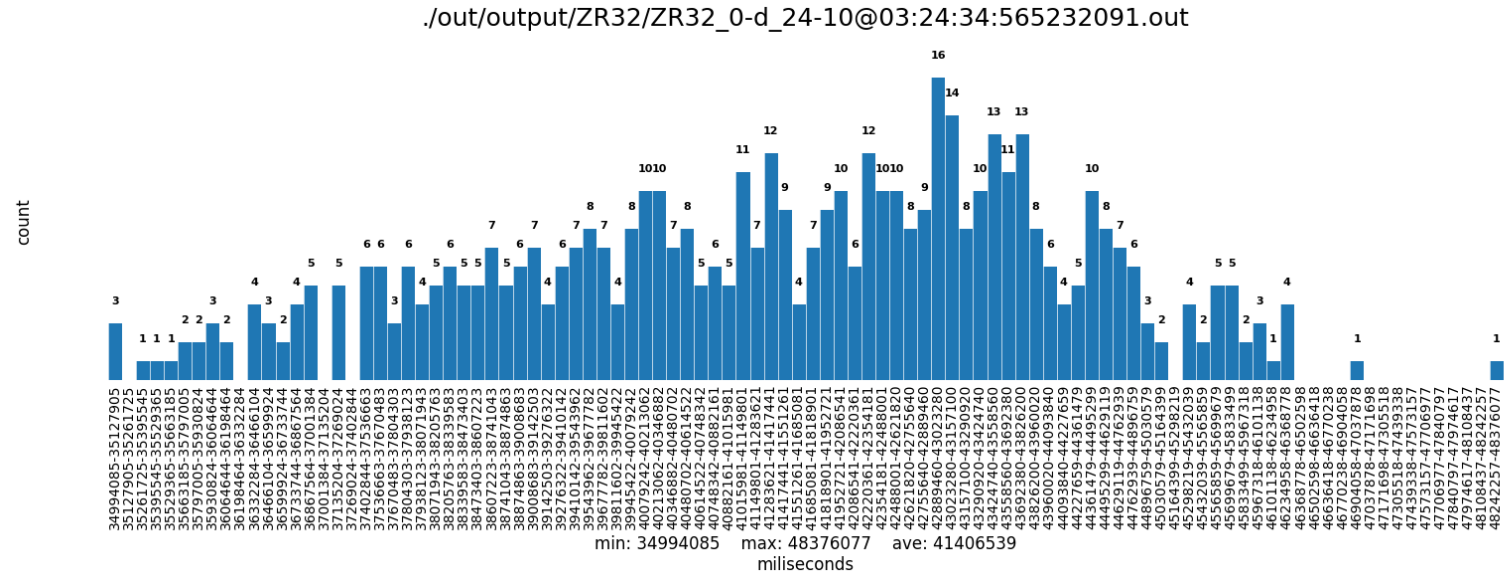
./out/output/NR32/NR32_0-d_24-10@02:23:51:317205694.out

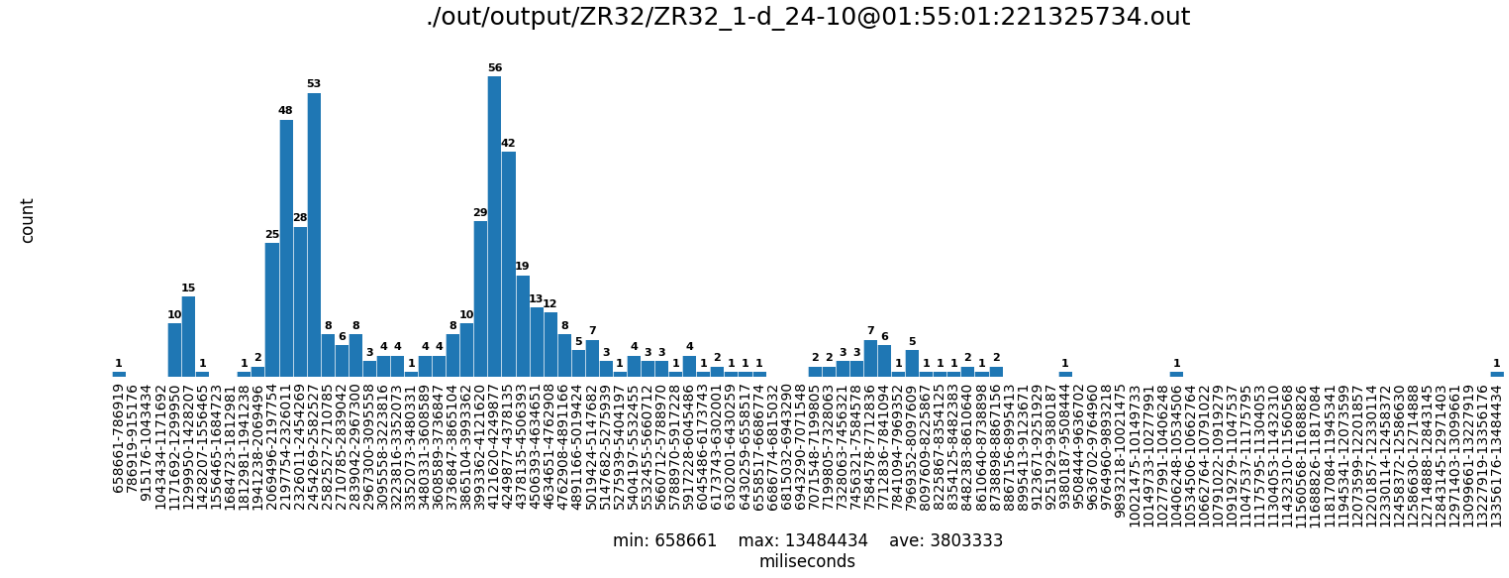


./out/output/NR32/NR32_1-d_24-10@01:40:55:525341405.out



2) The distributions of all of the obtained computational time to ZR sets also seem to match predefined distribution pattern. However, those computational times achieved by a BF method resemble mostly Standard Normal Distribution as the size of the set increases. Which might be sign of a random generation. Distributions of those, which were computed using the B&B method within the all sizes of sets (>10) consist of 3-4 regions with the highest distribution densities.
CONCLUSION: the sets were probably generated randomly, since the distributions might resemble Standard Normal Distribution pattern.





3) The majority of the worst cases do not match the expected computational time according to the average per this size of set and distribution of other computational times in this set, but they do match with the maxim possible according to the method of computation. In the majority of sets, only 1-6 cases of 500 form the worst ones, having a significant divergence with the results of the same set.
CONCLUSION: all the majority of the worst cases were probably added artificially not to match the common distribution pattern (applies for both of NR and ZR sets).

Brute Force method:

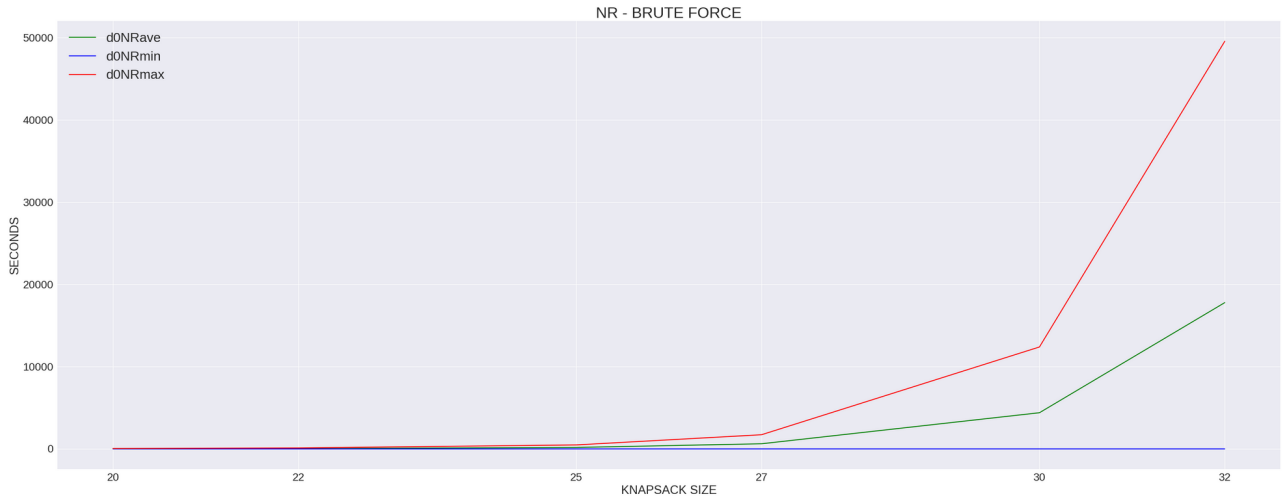
The method of Brute Force consists in going through all the possible combinations of items in the backpack and finding the combination with the highest price. An impermissible combination is one where the sum of the weights of the items exceeds the carrying capacity of the backpack. This method of solution is computationally very demanding, because in the general case (all combinations allowed) the number of combinations is **2^n**, where n is a total number of items.

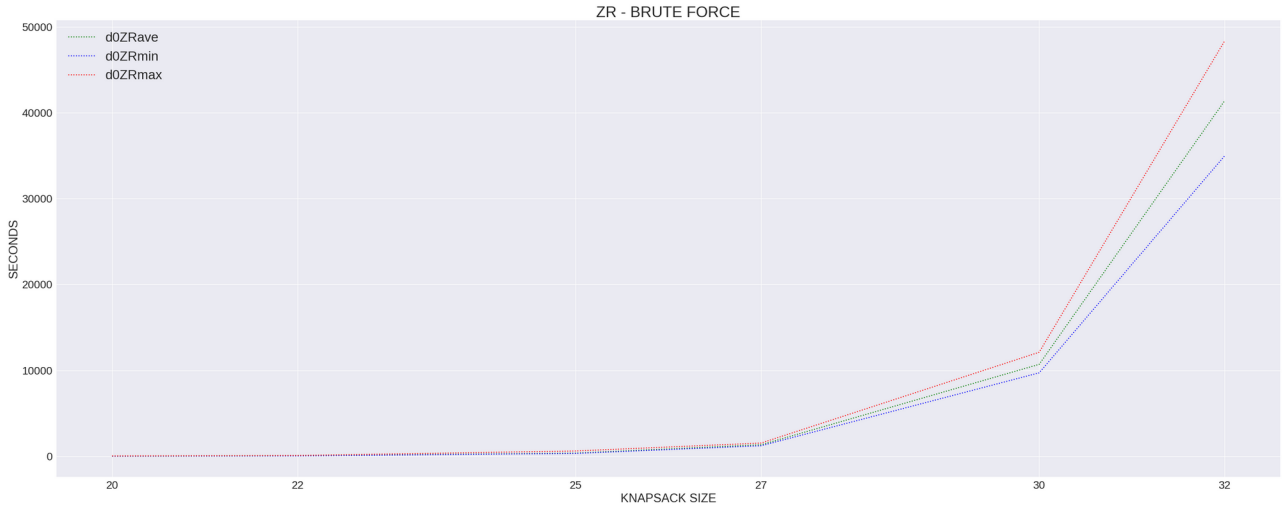
Table with average values for the both of NR and ZR set

Number of items (n)	**Time (s)**
4	immeasurebale
10	0.0004
15	0.0016
20	0.0411
22	1.520
25	11.63
27	54
30	417
32	1788
35	14002
37	56200
40	449807

Note: for the task with n = 4 the calculation time could not be measured because it is too small. Tasks with n larger than 32 for ZR and 35 for NR datasets could not be measured as the computational times would reach several hours. The times given were estimated on the basis of previous measured values and from knowledge of the complexity of the problem.

Dependence of the computation time on the number of items in the knapsack, using the BF method for both of NR and ZR sets is seen in the following plots correspondingly (n < 20 skipped due to insignificant values):





Branch & Bound method:

Branch and bound (B&B) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. Combinatorial optimization problems are mostly **exponential** in terms of time complexity. Also it may require to solve all possible permutations of the problem in worst case.

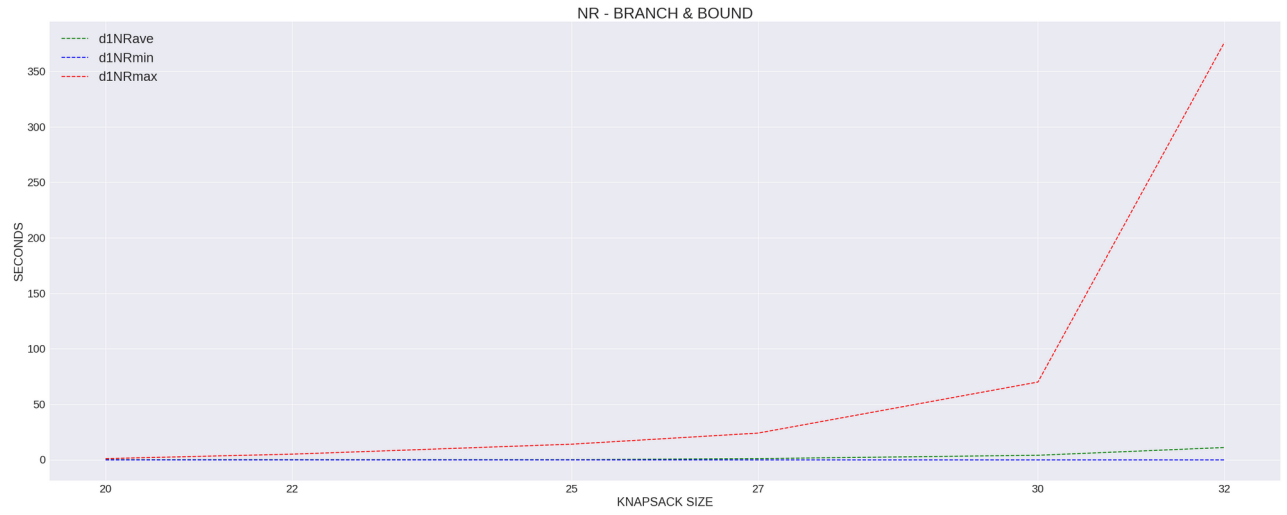
Table below compounds average computational times for the both of NR and ZR set. It worth mentioning, that in fact the computational time for ZR might be in tenth o times bigger than for NR.

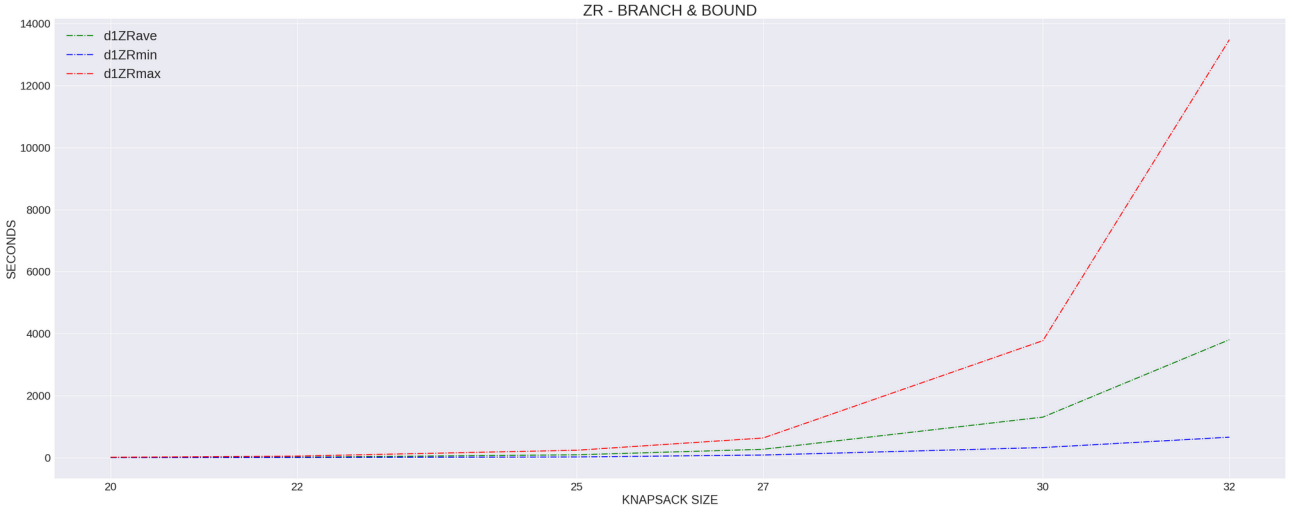
Note: for the task with n = 4 the calculation time could not be measured because it is too small. Tasks with n larger than 37 for ZR dataset could not be measured as the computational times would reach several hours. The times given were estimated on the basis of previous measured values and from knowledge of the complexity of the problem.

Number of items (n)	**Time (s)**

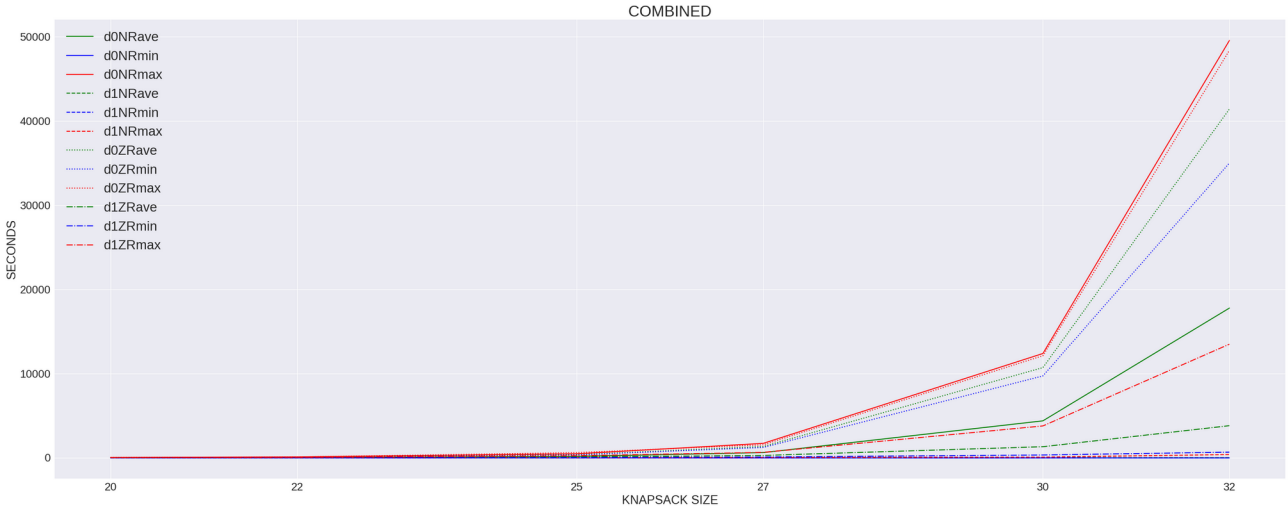
4	0.00015
10	0.0014
15	0.170
20	2.4
22	7.120
25	45.5
27	134.8
30	655.6
32	2788
35	19002
37	56200
40	749807

Dependence of the computation time on the number of items in the knapsack, using the BF method for both of NR and ZR sets is seen in the following plots correspondingly (n < 20 skipped due to insignificant values):





Plot, shoeing the dependence of computational time on the size of the set for each method:



The time changes depending on the size of the set due to the growth of the number of possible combinations that are to be checked and proceed.

Conclusions

During this experiment I have experimentally proved that computational time depends on the size of the set and chosen method of computation, that also were briefly explained. I have developed a toolchain for automatic processing of different knapsack problems. Moreover, I have collected and analyzed the data obtained after 12+ hours run of a KNAPSACKer, answering the questions set in the task.