Knapsack problem algorithms execution

[1]Eduardo Pinhata,
[2]Rafael P Cassetta
[1,2]CMCC, Federal University of ABC,
Santo André, São Paulo, Brazil

e-mail: [1] eduardo.pinhata@ufabc.edu.br,
[2] cassetta.rafael@ufabc.edu.br

01/08/2025

Abstract - Report of the execution of knapsack instances for the algorithm provided by the Theory of Computer class, in 2025.

1 PROBLEM DESCRIPTION

The 0–1 Knapsack Problem (KP) involves selecting a subset of items to be placed in a knapsack with limited capacity, such that the total profit is maximized while the total weight does not exceed the capacity. Each item is characterized by a positive integer weight and profit, and can either be included in its entirety or excluded.

The problem can be formulated as the following integer linear program:

$$\text{maximize} \quad z = \sum_{i=1}^{n} p_i x_i \tag{1.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq C \tag{1.2}$$

$$x_i \in \{0, 1\}, \quad \text{for } i = 1, \ldots, n \tag{1.3}$$

$$w_i, p_i, C, \in \mathbb{Z}_{>0} \tag{1.4}$$

$$\tag{1.5}$$

In this formulation, the decision variable $x_i = 1$ indicates that the item i is included in the knapsack, while 0 means it is excluded. The parameters p_i and w_i denote, respectively, the profit and weight associated with item i.

The fractional knapsack problem is a relaxation of the 0-1 KP, in which it's permissible to select fractional amounts of each item. Mathematically, this is modeled by allowing the decision variables to take values in the interval $[0, 1]$, i.e., $0 \leq x_i \leq 1$, for $i = 1, \ldots, n$.

2 EXPERIMENTS

All experiments were conducted on a desktop computer equipped with an AMD Ryzen 5 2600 Six-Core Processor, operating at a base clock of 3.40 GHz. The system featured 16 GB of DDR4 RAM running at 2400 MHz.

For storage, the system utilized a 477 GB SATA SSD (SanDisk SDSSDH3512G), ensuring high-speed data access and low-latency disk operations during the execution of the benchmarks. The motherboard used was a Gigabyte B450M DS3H, and the operating system was a 64-bit version of Microsoft Windows 10, based on the x64 architecture.

The instance **Instancia_n10000000_C1500.txt** required a significantly larger amount of memory than the available system could support. Consequently, it was executed on a different machine, operated by a colleague, which was equipped with 64 GB of RAM to accommodate the memory demands of the instance.

The Table 1 presents the execution details for each instance.

Table 1: Execution Times for Different Knapsack Implementations

Instance	**Execution Times (ms)**		
	knapsack_matrixs	**knapsack_array**	**fractional_knapsack**
Instancia_n4_C11.txt	0.00002s	0.00001s	7.1526e-06s
Instancia_n4_C20.txt	0.00004s	0.00002s	8.5831e-06s
Instancia_n5_C80.txt	0.00015s	0.00007s	8.1062e-06s
Instancia_n7_C50.txt	0.00012s	0.00006s	7.1526e-06s
Instancia_n10_C60.txt	0.00021s	0.00011s	8.1062e-06s
Instancia_n10_C269.txt	0.00090s	0.00047s	1.0014e-05s
Instancia_n10_C1500.txt	0.00565s	0.00294s	8.8215e-06s
Instancia_n20_C878.txt	0.00609s	0.00335s	1.0252e-05s
Instancia_n20_C879.txt	0.00611s	0.00335s	1.0729e-05s
Instancia_n23_C10000.txt	0.07655s	0.04389s	1.1921e-05s
Instancia_n50_C1500.txt	0.02490s	0.01400s	1.7643e-05s
Instancia_n60_C1500.txt	0.02973s	0.01709s	1.812e-05s
Instancia_n70_C1500.txt	0.03545s	0.01988s	2.0742e-05s
Instancia_n80_C1500.txt	0.04008s	0.02280s	2.5034e-05s
Instancia_n90_C1500.txt	0.04497s	0.02483s	2.4796e-05s
Instancia_n100_C200.txt	0.00582s	0.00249s	2.6464e-05s
Instancia_n100_C1500.txt	0.05060s	0.03138s	3.0041e-05s
Instancia_n110_C1500.txt	0.05704s	0.03345s	2.9325e-05s
Instancia_n120_C1500.txt	0.07923s	0.04646s	3.4809e-05s
Instancia_n130_C1500.txt	0.06823s	0.03674s	6.3181e-05s
Instancia_n140_C1500.txt	0.07418s	0.03966s	3.6955e-05s
Instancia_n150_C1500.txt	0.07799s	0.04290s	4.0293e-05s

Instance	**Execution Times (ms)**		
	knapsack_matrixs	**knapsack_array**	**fractional_knapsack**
Instancia_n200_C1500.txt	0.10770s	0.06122s	4.7922e-05s
Instancia_n300_C1500.txt	0.18421s	0.09323s	6.8426e-05s
Instancia_n400_C1500.txt	0.32060s	0.14576s	9.9421e-05s
Instancia_n500_C1500.txt	0.31781s	0.17091s	0.00014758s
Instancia_n500_C15000.txt	3.34830s	1.67130s	0.00012398s
Instancia_n600_C15000.txt	3.95334s	2.03631s	0.00013995s
Instancia_n700_C15000.txt	4.97988s	2.43616s	0.00016141s
Instancia_n800_C15000.txt	5.56325s	2.99706s	0.00018287s
Instancia_n900_C15000.txt	6.58090s	3.31355s	0.00021338s
Instancia_n1000_C1500.txt	0.64195s	0.34581s	0.00019526s
Instancia_n1000_C15000.txt	7.29286s	3.52476s	0.00021529s
Instancia_n1000_C150000.txt	76.50902s	39.90970s	0.00037789s
Instancia_n1100_C150000.txt	78.53516s	43.10122s	0.00037813s
Instancia_n1500_C150000.txt	112.33086s	60.04666s	0.000494s
Instancia_n2000_C1500.txt	1.27882s	0.64383s	0.00058746s
Instancia_n2000_C150000.txt	155.79723s	79.57818s	0.0009985s
Instancia_n2500_C150000.txt	222.95376s	99.14594s	0.0019848s
Instancia_n3000_C1000.txt	1.79190s	0.64277s	0.001224s
Instancia_n3000_C2000.txt	3.03347s	1.35515s	0.0016356s
Instancia_n3000_C3000.txt	4.07217s	2.28823s	0.00073266s
Instancia_n3000_C4000.txt	5.47980s	2.76673s	0.0007174s
Instancia_n3000_C5000.txt	6.88498s	3.72504s	0.00057387s
Instancia_n3000_C10000.txt	14.20639s	7.72616s	0.00065398s
Instancia_n3000_C20000.txt	29.45248s	14.47092s	0.0007031s
Instancia_n3000_C50000.txt	78.84365s	38.37094s	0.0012991s
Instancia_n3000_C150000.txt	295.85637s	118.18132s	0.0045125s
Instancia_n5000_C1500.txt	3.44422s	1.65534s	0.0010509s
Instancia_n6000_C1500.txt	4.17801s	2.12799s	0.0011613s
Instancia_n7000_C1500.txt	4.84521s	2.35997s	0.0014217s
Instancia_n8000_C1500.txt	5.72800s	2.67034s	0.0015898s
Instancia_n9000_C1500.txt	6.32539s	3.01313s	0.0017879s
Instancia_n10000_C1500.txt	7.16527s	3.29443s	0.0019531s
Instancia_n11000_C1500.txt	8.28845s	3.68849s	0.0022068s
Instancia_n12000_C1500.txt	8.72211s	4.08262s	0.0025287s
Instancia_n13000_C1500.txt	9.45511s	4.35835s	0.0027089s
Instancia_n14000_C1500.txt	10.54699s	5.00981s	0.00402s
Instancia_n15000_C1500.txt	11.08957s	5.07784s	0.003191s
Instancia_n50000_C1500.txt	37.78176s	16.80397s	0.014268s
Instancia_n100000_C1500.txt	73.77395s	33.63138s	0.031455s
Instancia_n10000000_C1500.txt*	337.67130s	33.26745s	7.23686s

* Ran in a different machine.

Figures 1, 2, 3, 7, 5, and 6, presented in the Appendix, illustrate the growth in execution time as a function of the number of items available for selection and the knapsack's capacity value.

3 DISCUSSION

Based on the analysis of the charts, the fractional knapsack algorithm is the fastest among the three approaches, as it iterates only over the number of items in the instance. However, this method does not guarantee an optimal solution, since it is designed for a relaxed version of the problem where fractional quantities of items are allowed.

The matrix-based algorithm is consistently slower than the array-based algorithm. Although both implementations involve two nested loops, resulting in a computational complexity of approximately $O(n{\cdot}C)$, their performance diverges due to memory usage and access patterns.

Specifically, the matrix-based approach (MA) stores a full $n \times C$ matrix, while the array-based version (AA) maintains a single array of size C, reusing memory across iterations. This difference in memory usage has significant implications for performance: when the matrix does not fit entirely into the CPU cache, memory access becomes slower, leading to increased execution time. In contrast, the array-based algorithm benefits from better cache locality, resulting in faster execution.

Additionally, the matrix-based implementation performs more frequent and dispersed memory accesses compared to the array-based one, which primarily updates a single array. These two factors — higher memory consumption and less efficient memory access — explain why the execution time of MA grows more rapidly than AA as the input size increases, despite similar theoretical complexity.

4 APPENDIX

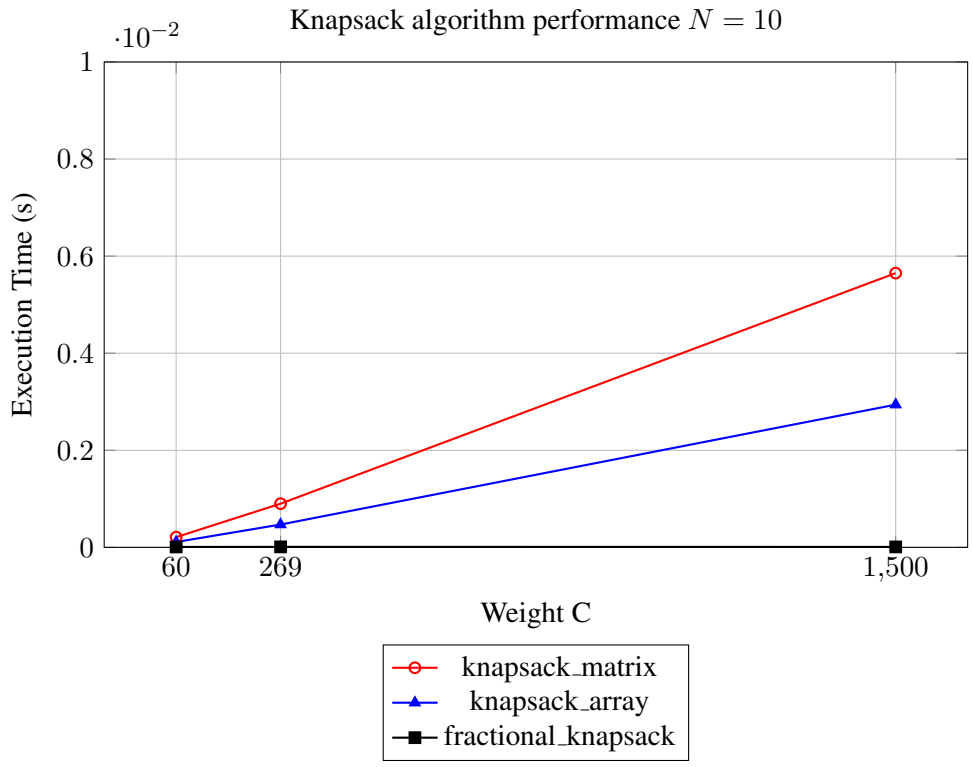

Figure 1: Execution time comparison of knapsack algorithm for different weights and $N = 10$

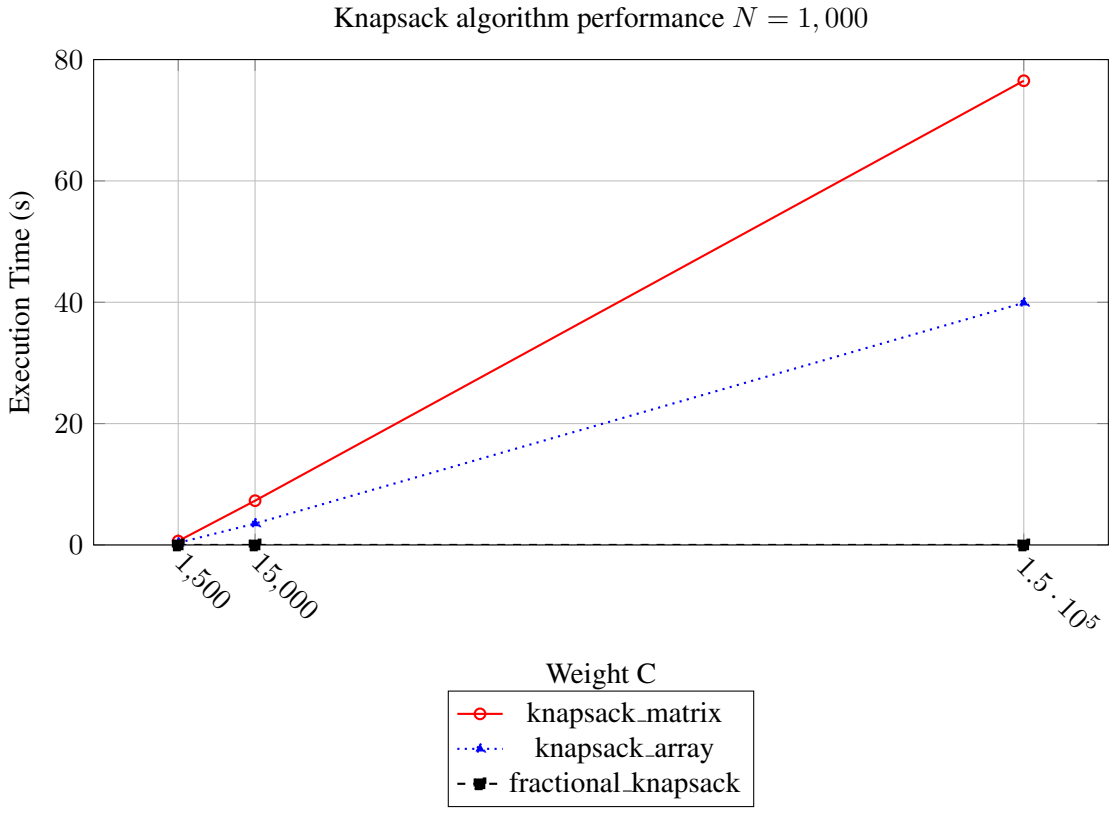

Figure 2: Execution time comparison of knapsack algorithm for different weights and $N = 1,000$

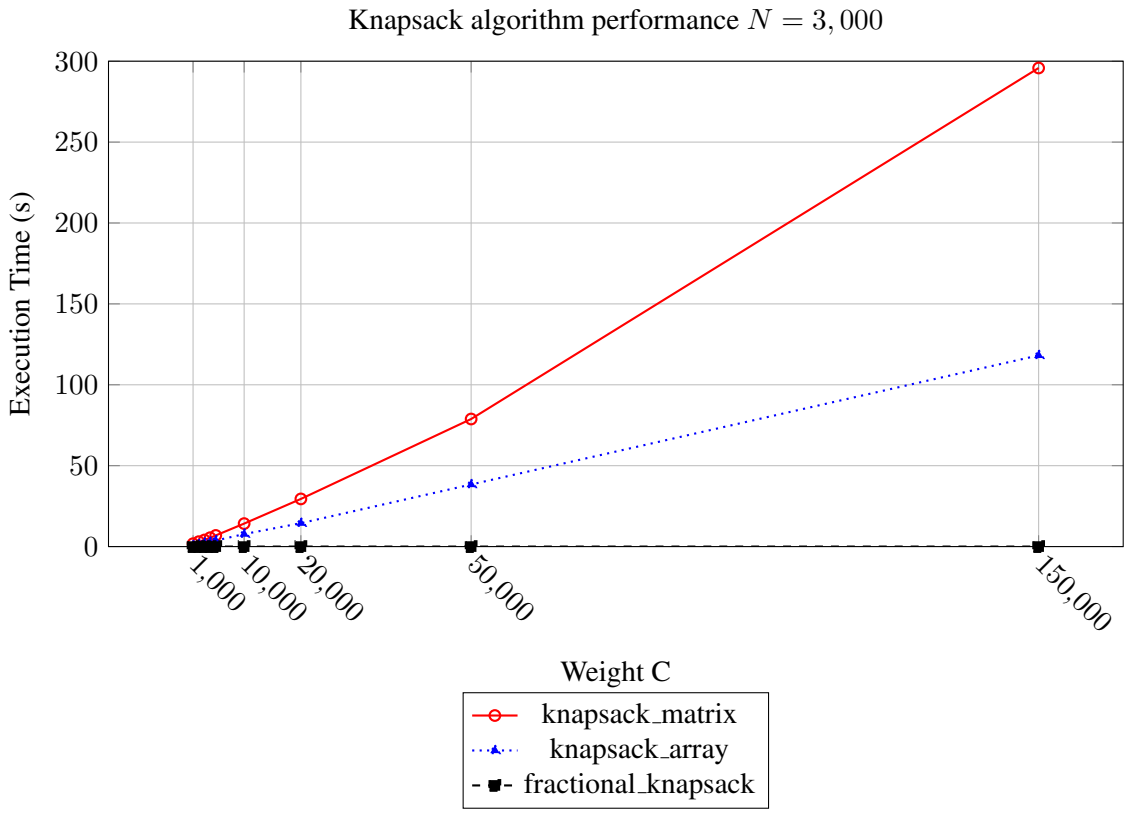

Figure 3: Execution time comparison of knapsack algorithm for different weights and $N = 3,000$

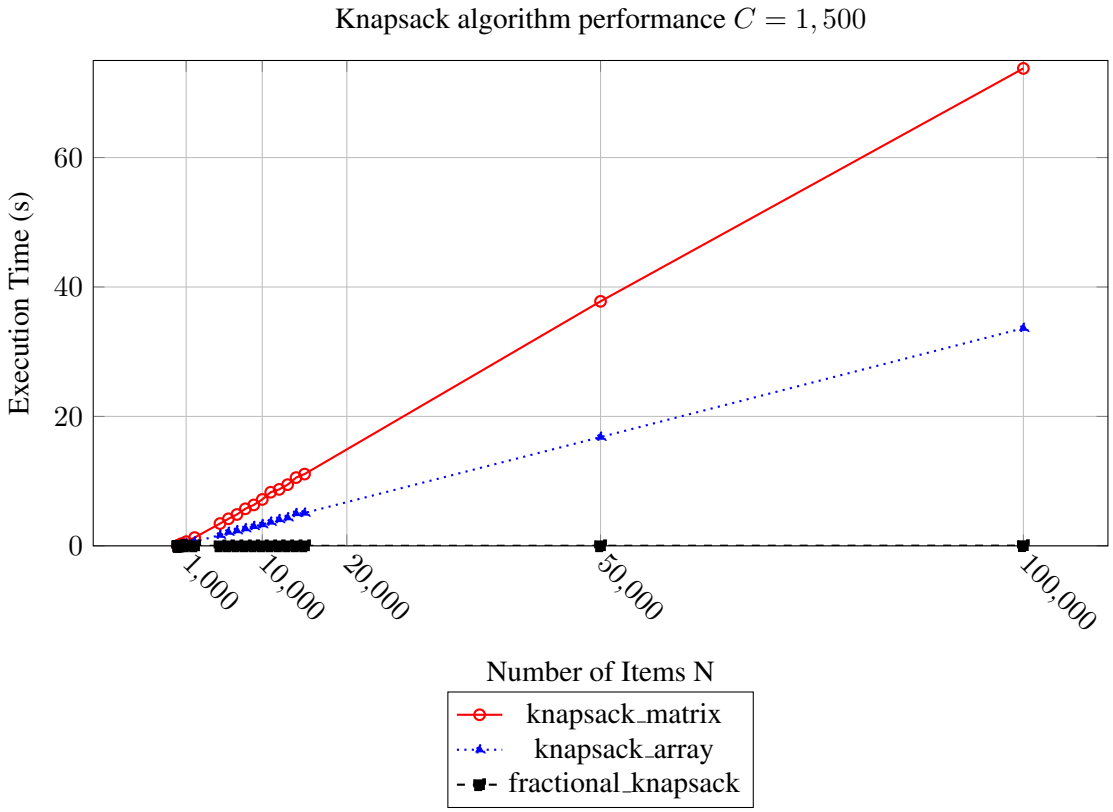

Figure 4: Execution time comparison of knapsack algorithm for different number of items to be added in the knapsack and $N = 1,500$

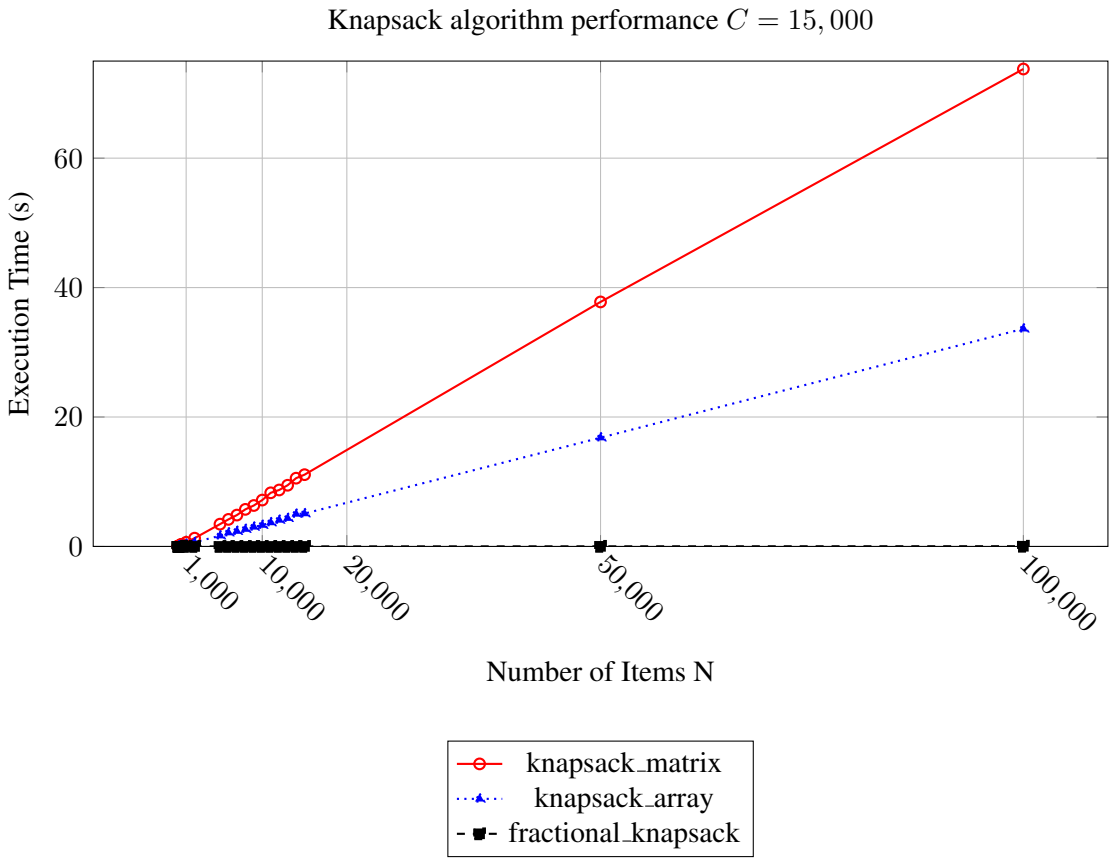

Figure 5: Execution time comparison of knapsack algorithm for different number of items to be added in the knapsack and $N = 15,000$

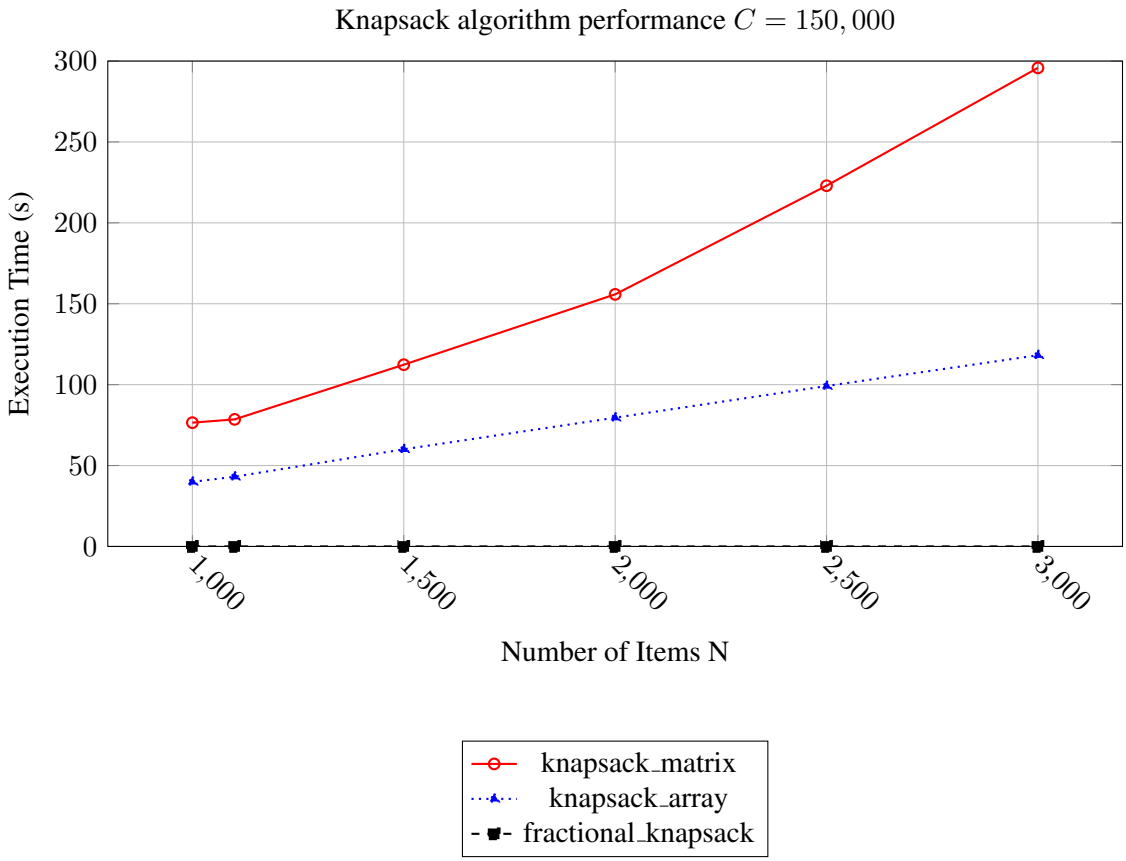

Figure 6: Execution time comparison of knapsack algorithm for different number of items to be added in the knapsack and $N = 150,000$

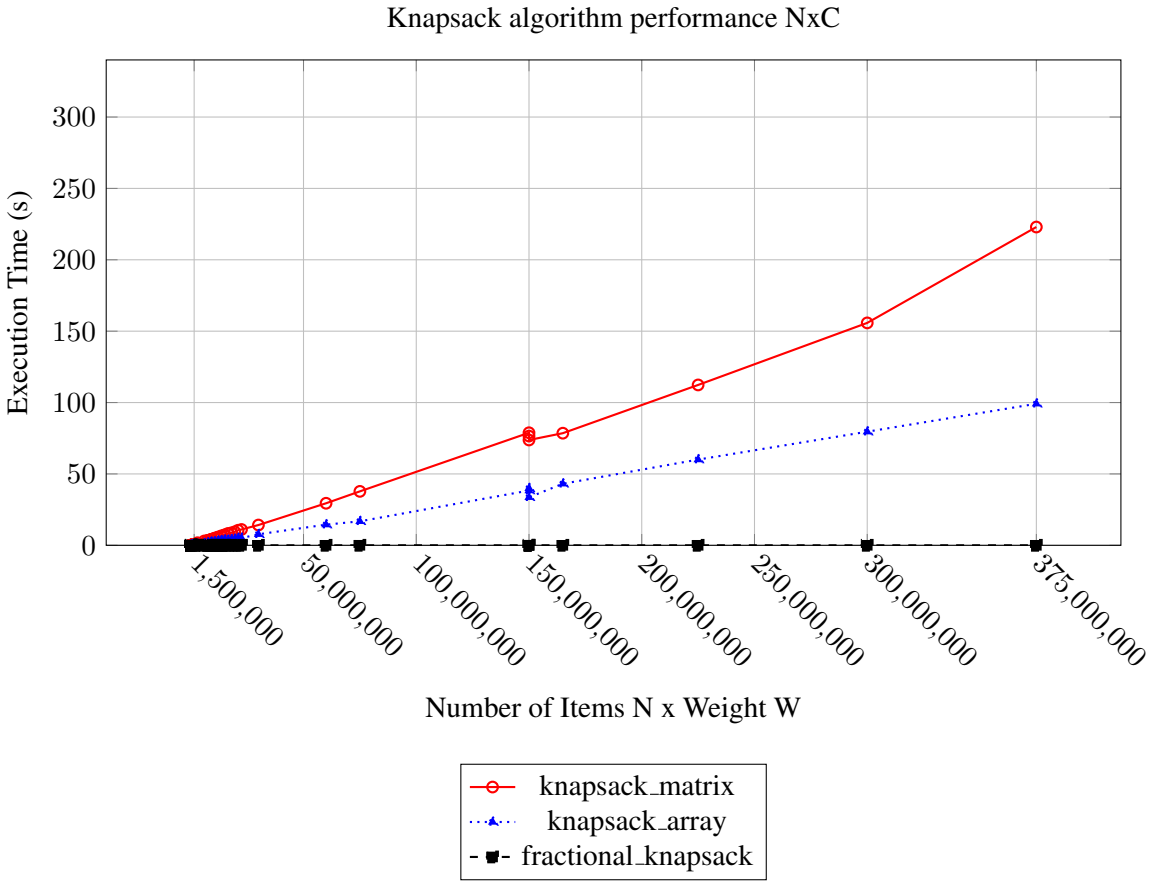

Figure 7: Execution time comparison of knapsack algorithm for different number of items to be added in the knapsack times Weight.