Performance Evaluation Report

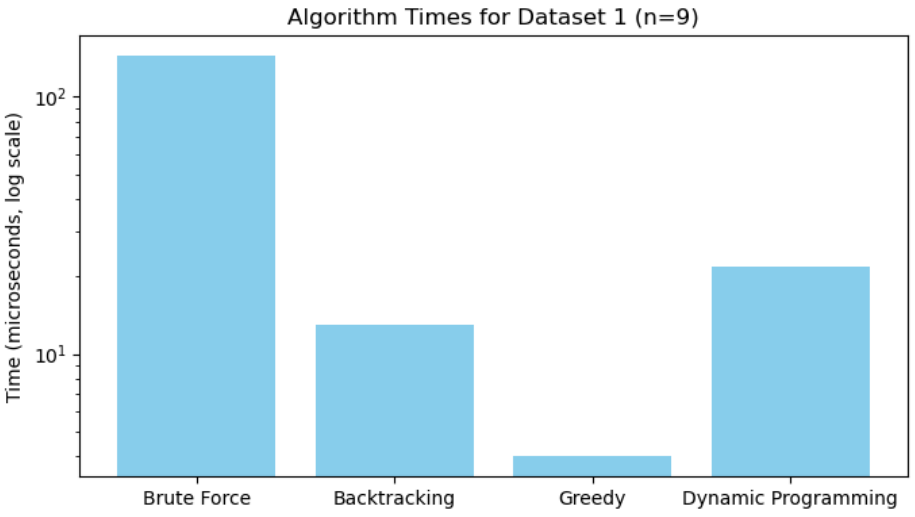
Important Notes

The performance evaluation of each algorithm was primarily based on their running time. While this metric offers a basic insight into efficiency, it is not the most accurate indicator, as it can be influenced by several external factors such as the underlying hardware, programming language, and system load. To reduce such variability, all tests were conducted on the same machine, and all algorithms were implemented in C++ for consistency.

The Integer Linear Programming (ILP) solution, however, utilizes a solver library available for Python. As a result, its execution time is not directly comparable to that of the other C++ implementations due to differences in runtime environments and language efficiency. Therefore, we evaluate the ILP algorithm solely based on its performance across varying input sizes, without drawing direct comparisons to the other algorithms.

A detailed analysis of the ILP solution's performance is provided at the end of this report.

Dataset 1 (n = 9)

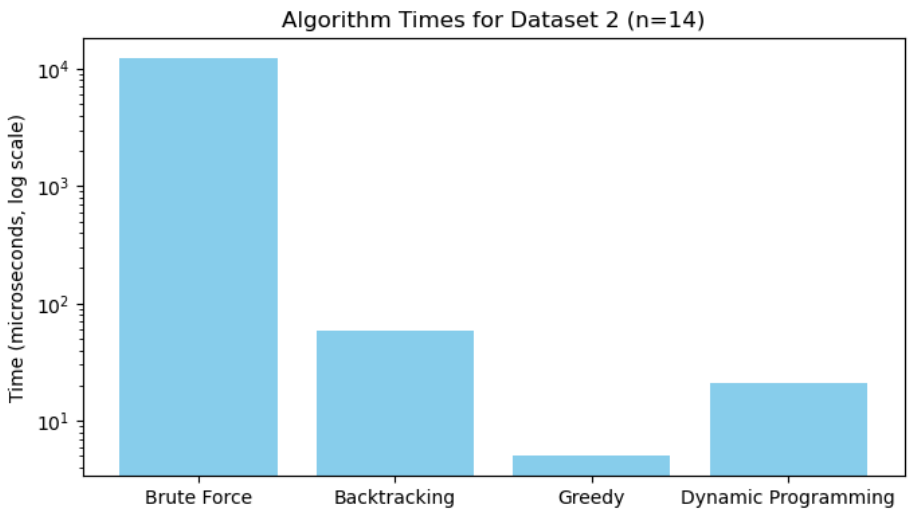


As expected, the results align with the theoretical asymptotic complexities studied for each of these approaches. The greedy algorithm demonstrated the most efficient runtime, although it sacrifices accuracy and does not always produce the optimal solution. In this case it was

able to reach the maximum profit, but as we conclude in other datasets, that is not always true. The brute-force and backtracking approaches exhibited significantly higher execution times due to their exponential time complexity. However, the backtracking method showed noticeable improvements over brute-force by pruning unnecessary branches, reducing the number of computations.

The dynamic programming solution had a higher runtime in this particular case, primarily due to its increased memory usage and overhead. Despite this, as we will observe in tests with larger input sizes, its runtime grows much more slowly compared to the other methods. This makes it a far more scalable and efficient approach in the long run—always yielding optimal results and representing the most balanced solution overall.

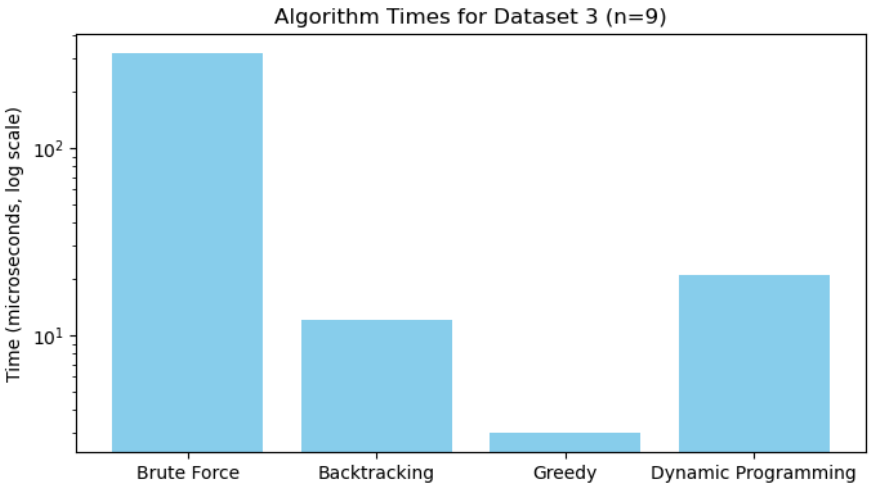
Dataset 2 (n = 14)



In this example, we can confirm the findings observed in the previous dataset. As the number of pallets increases slightly, the dynamic programming approach maintains nearly the same running time, demonstrating its stability and efficiency. Meanwhile, the backtracking method and the brute force already show a noticeable increase in execution time, reinforcing the conclusion that dynamic programming scales much better and is more suitable for larger input sizes.

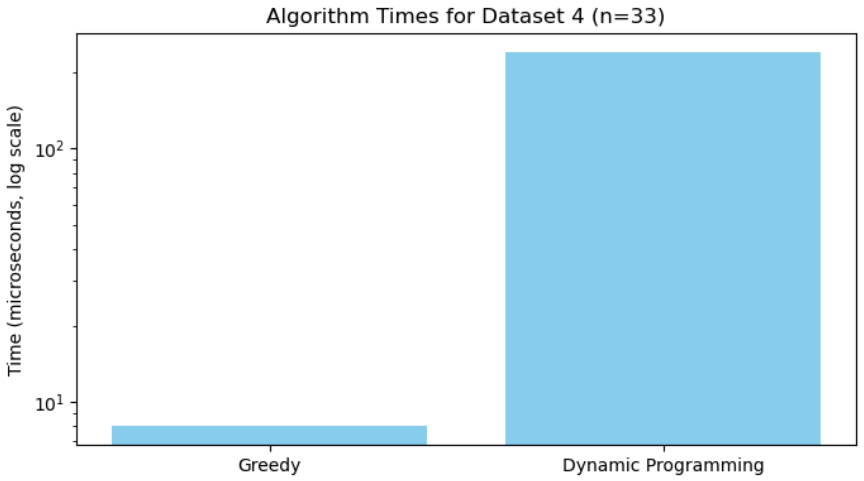
In this particular dataset, the greedy algorithm was also able to reach the optimal solution. Although the optimal solution is not guaranteed and this outcome does not always occur, the maximum profit was found in many instances across other datasets as well.

Dataset 3 (n = 9)



In this example, we can further validate the results obtained previously. The greedy algorithm once again achieved the maximum profit, demonstrating that although it does not guarantee an optimal solution in all cases, its performance in small instances and its impressive running times often compensate for this limitation.

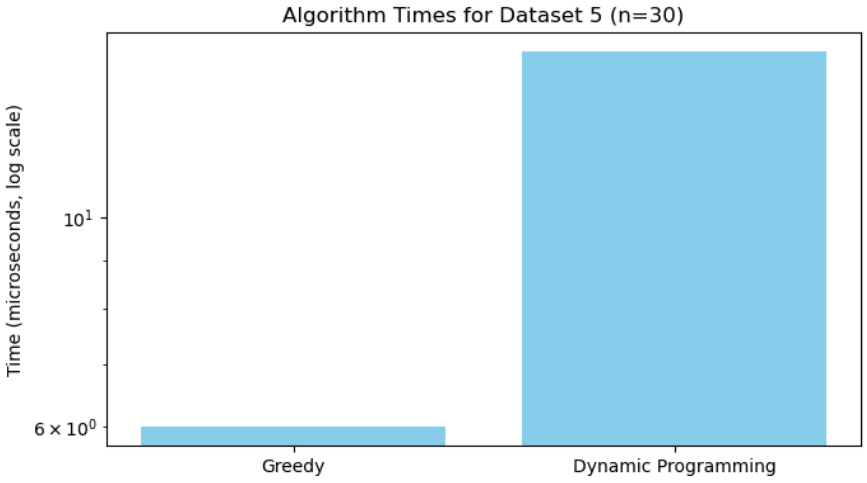
Dataset 4 (n = 33)



In this dataset, a significant difference in performance was observed. Due to their exponential time complexity, the brute-force and backtracking approaches require impractically long execution times, rendering them unsuitable for real-world applications. Consequently, these methods were restricted to instances involving up to 25 pallets.

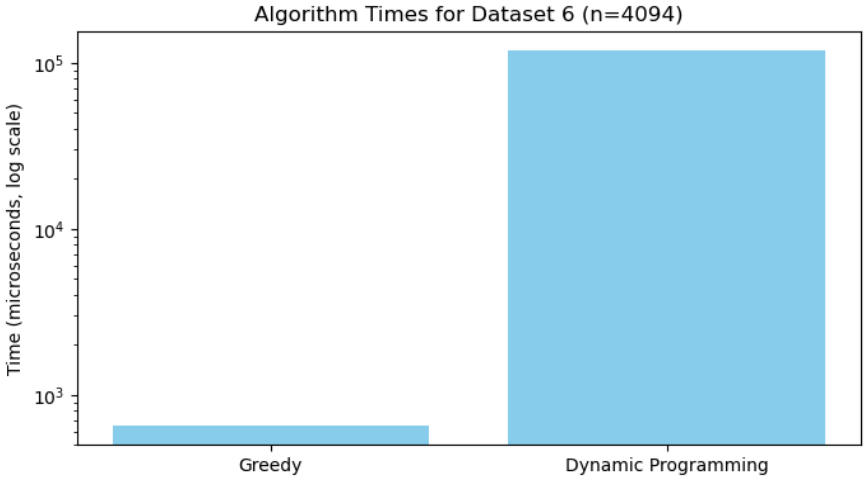
The dynamic programming approach, on the other hand, consistently delivers excellent performance, providing the optimal solution with reasonable execution times. In contrast, while the greedy algorithm is extremely fast, it does not guarantee an optimal solution. In this case, the greedy algorithm failed to reach the optimal solution, achieving a maximum profit of 158 compared to the optimal profit of 161, although indicating only a small difference.

Dataset 5 (n = 30)



In this case, we can reaffirm the results observed in the previous dataset. Although the input size is already considerable, the greedy approach was still able to achieve the maximum profit.

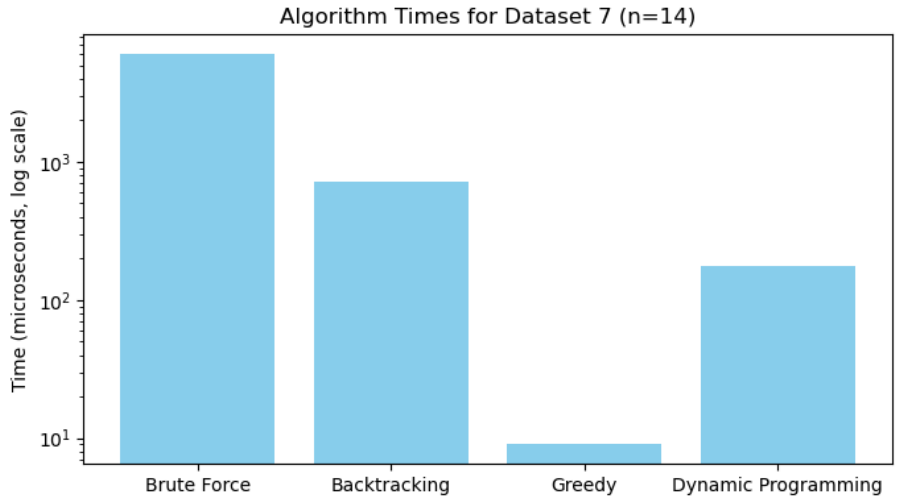
Dataset 6 (n = 4094)



In this dataset, the dynamic programming solution demonstrates a truly remarkable running time, especially given the large input size. However, for even larger inputs, memory management becomes a significant concern, as the construction of the DP table can consume substantial memory and impact performance.

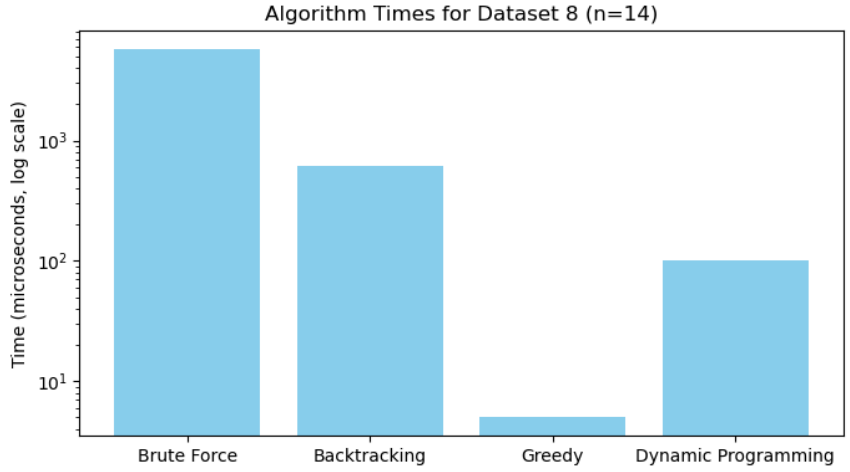
In this case, the greedy approach was able to reach the maximum profit, however, since all elements have the same weight-to-value ratio, this is a very specific scenario where the greedy method is guaranteed to find the maximum profit.

Dataset 7 (n = 14)



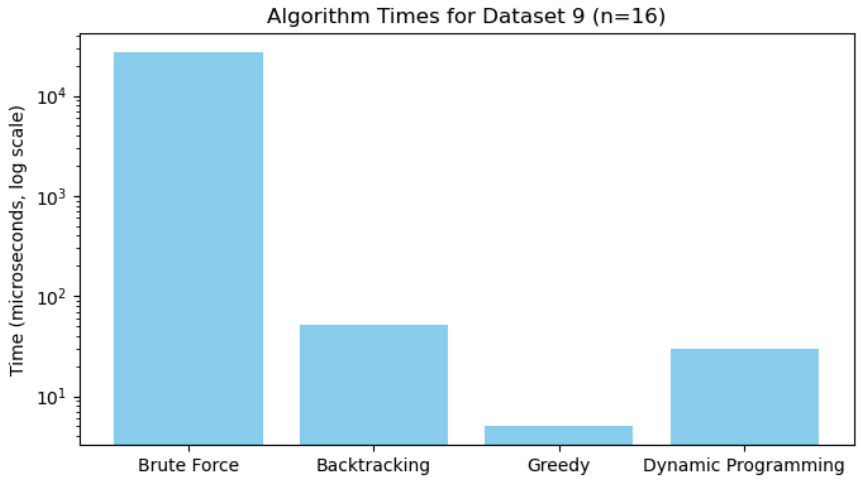
This example serves only to validate the findings observed in previous datasets. In this example the greedy algorithm was able to reach the maximum profit again.

Dataset 8 (n = 14)



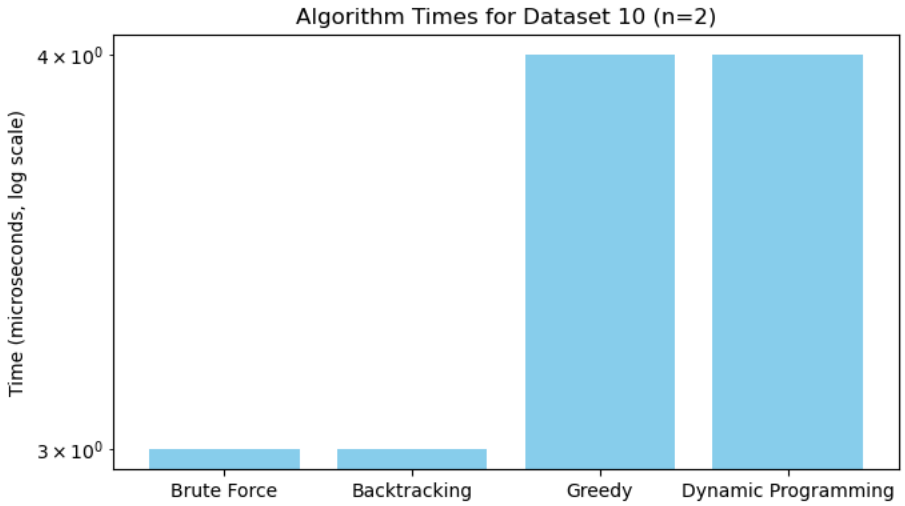
This example serves solely to validate the findings observed in previous datasets. The greedy algorithm was once again able to reach the maximum profit, which is a particularly surprising result.

Dataset 9 (n = 16)



This example serves only to validate the findings observed in previous datasets. The greedy approach was able to reach the maximum profit again.

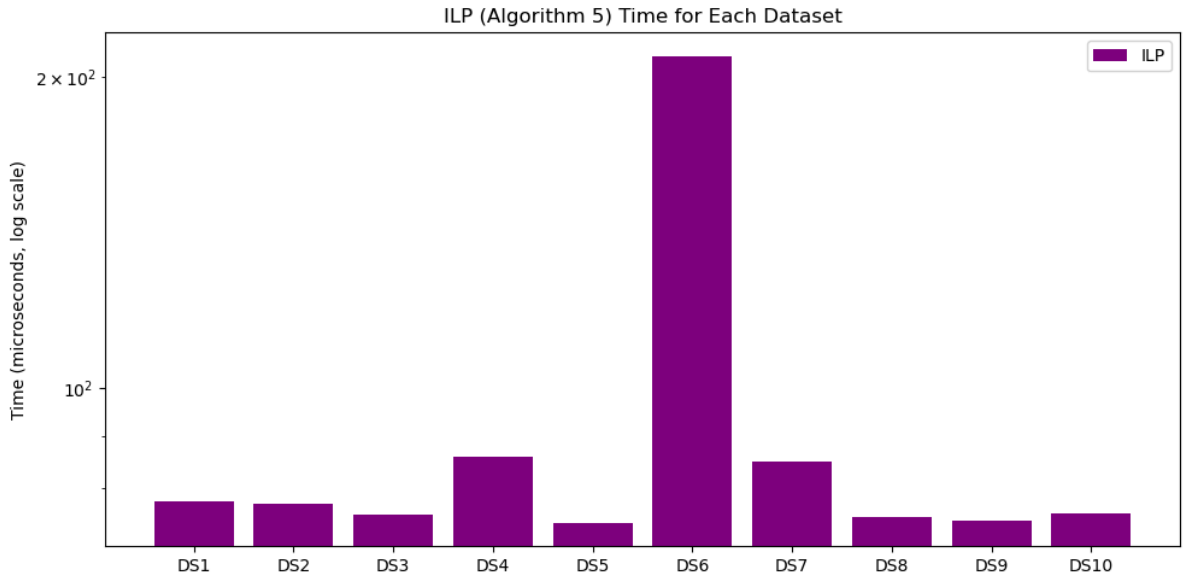
Dataset 10 (n = 2)



Due to the small input size in this example, all algorithms exhibited identical execution times. Although the graphs might suggest a notable difference, the actual variation was only 1 microsecond.

This case also represents the scenario where the greedy approach performs the worst, achieving a profit of only 1 compared to the optimal profit of 15, marking the largest observed discrepancy.

ILP approach



This approach demonstrates remarkable consistency, successfully solving all datasets with minimal difficulty. It delivers efficient running times even for input sizes as large as 4094, underscoring its computational efficiency. Furthermore, the algorithm consistently attained the optimal solution in every case, thereby rigorously validating its optimality.

Conclusion

Across all tested datasets, the results consistently aligned with the theoretical expectations of each algorithm. The dynamic programming approach proved to be the most reliable, consistently delivering optimal solutions with reasonable execution times, especially as input sizes increased. The brute-force and backtracking methods, while accurate, were limited by their exponential time complexity and practical applicability.

Surprisingly, the greedy algorithm demonstrated strong performance, achieving the optimal solution in 8 out of 10 cases. Despite its lack of guaranteed optimality, it delivered

exceptional execution times and often produced near-optimal results—failing significantly in only 2 instances.

Additionally, the Integer Linear Programming (ILP) approach consistently produced optimal solutions with really good running times, an incredible approach too.