Instituto Tecnológico y de Estudios Superiores de Monterrey

Análisis y diseño de algoritmos avanzados
(Gpo 604)

Actividad:

“InClassAssignment: Logistics w/DP, Greedy, and Backtracking”

Fecha de entrega: 14 de agosto de 2025

Profesor:

Nezih Nieto Gutiérrez.

Equipo #6

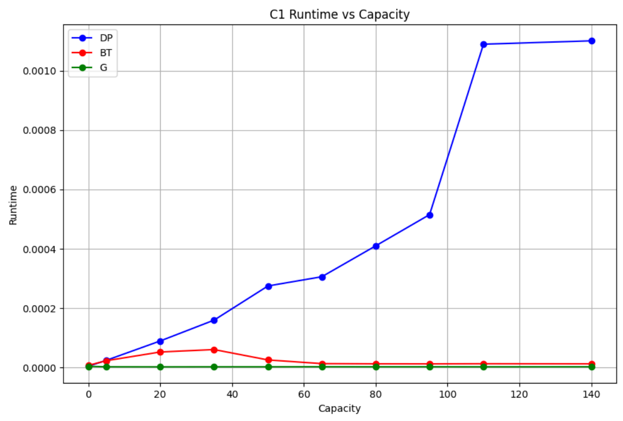
Angel Alexandro Angulo Prudencio | A00840313

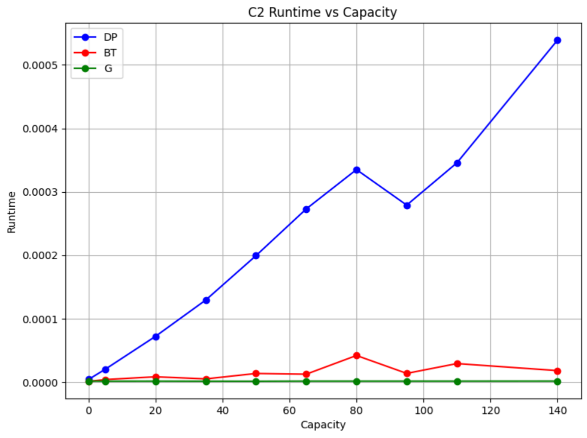
Rutilo Alberto de la Pena Rodriguez A01384647

Hector Aranda Garcia | A01178284

Sergio Gómez Guerrero | A01571538

Theoretical Analysis





Greedy Solution

The goal was to maximize the value of selected items without exceeding a weight limit. The solve method function applies a greedy heuristic then sorts items by value to weight ratio and selects them if they fit. Its complexity is O(n log n), making it more efficient than exact methods like dynamic programming. The main advantage is speed and good practical results, but the limitation is that it does not always guarantee the optimal solution.

DP Solution

In order to solve the problem with Dynamic Programming a solution using a Bottom-up approach was implemented using a two-dimensional table. The final result was calculated by using memoization to take into account the maximum value that can be carried in the trailer for each weight threshold up to weight *W* and building up the solution until the solution was reached. The maximum value between the alternatives of picking a certain value or not picking it is considered each iteration, this is what iteratively builds the final subset.

Experimental setup for the DP solution involved using the catalogues and weight capacities available to obtain the mean runtime for each one of them as well as the results of the solution. Graphs showing how this solution compares to others for each catalogue were created.

The theoretical complexities for the DP solution are as follows:

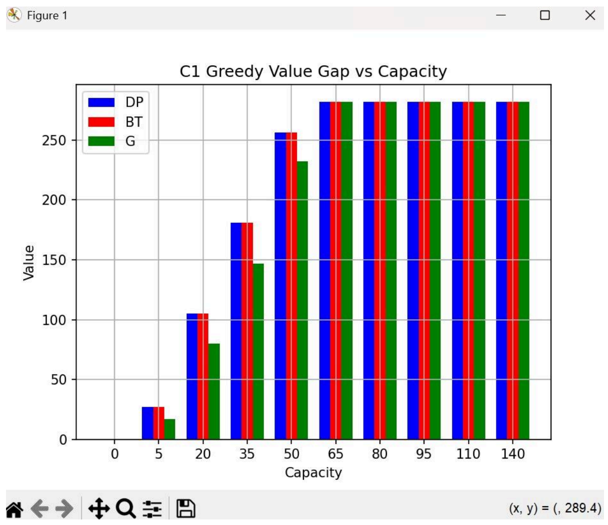
Time Complexity: $O(n \ * \ W)$
Space Complexity: $O(n \ * \ W)$

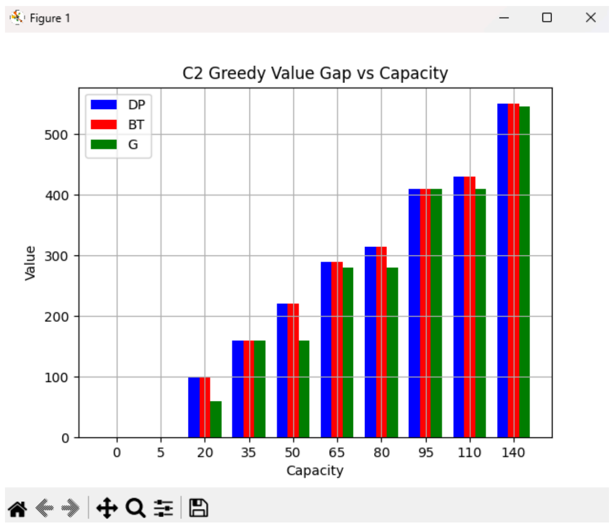
Where n is the size of the catalogue and W is the weight capacity. Complexity similar to a standard Knapsack Problem solution.

Branch-and-Bound Solution

Our implementation of the branch-and-bound algorithm uses a depth-first search, where the first branch is fully explored before looking at other possible paths. Products are included only if they fit within the remaining capacity. To eliminate unpromising branches, the maximum value that can be obtained from the current state is calculated. If there is a product that doesn't fit, a fractional portion of the maximum value is added.

Since the products are already organized according to their value-to-weight ratio, the bound calculation is linear: O(n). Since this calculation is performed for each node, it is relevant when calculating the time complexity. In the worst case, every possible node will be explored. This will take O(2^n), since each node has the decision to choose the next product in the list or ignore it. Combining this complexity with the bound calculation gives a complexity of O(n * 2^n). In turn, the space complexity is O(n) because it changes linearly with the number of products. As one can see, both time and space complexity are independent of W. Overall, Branch-and-Bound demonstrates to be a more expensive solution for smaller list sizes, but that expense mellows out as the list size increases.





Greedy Heuristic Underperformance

Given the Catalog List C2, a greedy approach underperforms with most capacities with the exception of 35 and 95. The Dynamic Programming and Branch-and-Bound consistently give either the same or better value for the allotted capacity.