PROJECT BASED LEARNING REPORT

CARGO LOADING

DESIGN AND ANALYSIS OF ALGORITHMS - BCS-DS-501

Submitted by:

SHIVAM SINGH	**1/23/SET/BCS/457**
SHIV SHANKAR	**1/23/SET/BCS/452**
ANURAG SINGH	**1/23/SET/BCS/446**
SANJAY THAKRAN	**1/23/SET/BCS/452**

Under the Guidance of

Ms. Deepika Kataria

(Assistant Professor)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

Computer Science & Engineering

Faculty of Engineering & Technology

MANAV RACHNA INTERNATIONAL INSTITUTE OF RESEARCH AND STUDIES, Faridabad

NAAC ACCREDITED 'A' GRADE

Aug – Dec, 2025

TABLE OF CONTENTS

ABSTRACT

The Cargo Loading project aims to optimize the selection of cargo items to achieve maximum profit while staying within a given weight capacity. This problem is modeled using the classical 0/1 Knapsack Problem, a widely studied optimization challenge in computer science. The project implements and compares multiple algorithms, including Dynamic Programming, Greedy Approach, Memoization (Top-Down Dynamic Programming), Pure Recursion, and Branch and Bound. Each algorithm provides a different perspective on problem-solving, offering variations in accuracy, execution time, and complexity.

A web-based system is developed using Flask for backend processing and HTML, CSS, JavaScript, and Chart.js for interactive visualization. The system enables users to add cargos, choose an algorithm, specify capacity, and instantly view results. Selected cargos, total weight, and total profit are displayed clearly, accompanied by visual comparison charts such as bar graphs and pie diagrams for better understanding. Additionally, a "Compare All Algorithms" option provides a consolidated performance overview of all implemented techniques.

This project demonstrates how algorithmic decision-making, combined with intuitive visualization, can improve the efficiency of cargo management, logistics planning, and resource utilization. It also highlights the strengths and limitations of each algorithm, making the system both educational and practical for real-world applications.

INTRODUCTION

Cargo loading is a fundamental task in logistics, transportation, and supply chain management, where the goal is to utilize available storage capacity efficiently while maximizing overall profit or value. Whether loading goods into trucks, containers, aircraft, or ships, selecting the right combination of items is essential to reduce transportation costs and improve operational efficiency. This selection process becomes increasingly complex when the number of items grows and constraints such as weight limits, volume, and profitability must be considered simultaneously.

The Cargo Loading project is based on the well-known **0/1 Knapsack Problem**, an optimization problem widely studied in the field of algorithms and data structures. The core objective is to determine which combination of cargos yields the highest profit without exceeding the maximum weight capacity. To address this, the project incorporates multiple algorithmic strategies—Dynamic Programming, Greedy Method, Memoization, Pure Recursion, and Branch and Bound. Each algorithm offers distinct advantages and trade-offs in terms of accuracy, computational efficiency, and complexity.

By integrating these algorithms into an interactive web-based system built using Flask, HTML, CSS, JavaScript, and Chart.js, the project provides users with practical insights into algorithm behavior. Users can input cargo details, compare algorithm results, and visualize performance through dynamic charts. This makes the system both a functional optimization tool and an educational platform for understanding algorithmic problem-solving.

PROBLEM STATEMENT

Cargo loading in transportation and logistics requires selecting the best combination of items that can fit within a limited capacity while maximizing profit. Manual selection often leads to inefficiencies such as underutilized space, overweight loads, or reduced profit margins. As the number of cargo items increases, determining the optimal combination becomes computationally difficult due to multiple constraints like weight, value, and quantity.

This project aims to address this challenge by applying algorithmic techniques to automate and optimize the cargo loading process. Using the 0/1 Knapsack model, the system identifies the most profitable set of items that can be loaded without exceeding the given capacity. The problem further includes comparing different algorithms to determine which approach provides the most effective and efficient solution.

OBJECTIVE

Primary Objectives

1. To develop a system that automates cargo loading using optimization algorithms.
2. To select cargo items that maximize total profit within a given capacity limit.
3. To compare various algorithms such as Dynamic Programming, Greedy, Memoization, Recursion, and Branch & Bound.

Technical Objectives

4. To implement multiple algorithms for solving the 0/1 Knapsack problem.
5. To build an interactive web interface using Flask, HTML, CSS, JavaScript.
6. To visualize results using bar charts and pie charts for better understanding.
7. To display selected cargos, total weight, and total profit clearly and accurately.

Performance Objectives

8. To analyze efficiency, accuracy, and complexity of each algorithm.
9. To enable a “Compare All Algorithms” feature for consolidated performance analysis.
10. To enhance understanding of algorithmic decision-making in real-world logistics.

THEORETICAL BACKGROUND

Overview of the Knapsack Problem

The **Knapsack Problem** is a classical optimization problem in computer science and operations research. It models the situation where we must choose a subset of items, each with a specific **weight** and **value**, such that the total value is **maximized** without exceeding a given capacity constraint**.**

In simple terms, imagine you are going on a trip with a backpack (knapsack) that can hold a **limited weight**. You have several items to choose from, each having a weight and a value. The challenge is to select the most valuable combination of items that fits within your backpack's capacity.

This problem is **NP-hard**, meaning that no polynomial-time algorithm is known to solve all instances optimally. However, various algorithms and optimization techniques provide efficient and practical solutions for specific cases.

Importance in real-world applications

The Knapsack Problem is not just theoretical — it models many real-world optimization problems such as:

- **Cargo loading** and **freight management** (like in this project)
- **Budget allocation** in investment or resource planning
- **Project selection** under limited manpower or capital
- **Data compression** and **memory allocation** in computing
- **Portfolio optimization** in finance

Because of these wide-ranging applications, the knapsack problem is a fundamental topic in **Design and Analysis of Algorithms (DAA)**.

Types of Knapsack Problems

There are several variations of the knapsack problem depending on constraints and problem formulation. The main types are:

(a) 0/1 Knapsack Problem

In this version, each item can either be **included or excluded** — you cannot take fractional parts of an item.

Example:
If a truck can carry up to 50 kg and you have three cargo items, you must decide whether to load each item completely or leave it out.

Algorithms commonly used:

- Dynamic Programming (Bottom-up / Top-down)
- Branch and Bound
- Recursive Backtracking

This is the variant primarily used in your **Cargo Loading Optimizer** project.

(b) Fractional Knapsack Problem

In this variation, you can take **fractions of an item**. The goal remains to maximize total value within capacity.

Example:
If a cargo item (e.g., grains or liquid) can be partially loaded, you can take 70% of its weight and value.

Algorithms used:

- **Greedy Algorithm**, which sorts items by their **value-to-weight ratio (v/w)** and selects items in that order until the knapsack is full.
- It can be solved optimally in **O(n log n)** time.

Solution Techniques

Branch and Bound - Branch and Bound is an **optimization technique** used to solve combinatorial problems more efficiently than brute-force methods. It systematically explores all subsets of items, but prunes (cuts off) parts of the search tree that cannot possibly lead to a better solution.

Steps:

1. Divide the problem into subproblems (Branching).
2. Calculate an upper bound on the maximum value possible for each subproblem.
3. If the upper bound is less than the current best solution, discard that subproblem (Bounding).
4. Continue until all promising branches are explored.

Advantages:

- Provides the **exact optimal solution**.
- Reduces the number of cases compared to brute force.

Disadvantages:

- Still computationally heavy for very large inputs.
- Complex to implement due to bounding logic.

UseCase:
Suitable for small to medium datasets where accuracy is essential.

Dynamic Programming (Tabulation Method) - Dynamic Programming (DP) solves the Knapsack problem by breaking it down into smaller subproblems and solving each one **iteratively**. The results of subproblems are stored in a **table (2D array)** to avoid re-computation.

Steps:

1. Create a DP table where rows represent items and columns represent capacity.
2. Fill the table iteratively based on whether including an item gives a better total value than excluding it.
3. The final cell gives the maximum possible profit.

Advantages:

- Guarantees **optimal solution**.
- More efficient than recursion because it avoids redundant calculations.

Disadvantages:

- Requires additional **O(n × W)** space, where n = number of items and W = capacity.
-

- **UseCase:**
  Best for medium-sized problems where both efficiency and accuracy are important.

Dynamic Programming (Memoization Method) - This approach uses **top-down recursion** combined with **memoization** (storing computed results). Whenever a subproblem is solved, its result is stored in memory, so the next time it's needed, it can be directly reused instead of recalculated.

Steps:

1. Recursively compute the result for including or excluding each item.
2. Store the result of each subproblem in a dictionary or table.
3. Return stored results for repeated subproblems.

Advantages:

- Avoids redundant calculations like tabulation.
- Easier to understand and implement recursively.

Disadvantages:

- May cause stack overflow for large inputs due to recursion depth.
- Requires similar space as tabulation.

UseCase:
Preferred when the recursive structure of the problem is easier to understand or implement.

Greedy Algorithm -The Greedy approach selects items based on a **specific criterion**, usually the **value-to-weight ratio**, and includes them in the knapsack until the capacity is reached. It makes locally optimal choices at each step.

Steps:

1. Calculate the ratio (value/weight) for all items.
2. Sort items in descending order of this ratio.
3. Select items until the knapsack is full.

Advantages:

- Very **fast** and simple to implement.
- Works **optimally for fractional knapsack** problems.

Disadvantages:

- Does **not guarantee optimal solution** for 0/1 knapsack problem.
- May produce suboptimal results when items can't be divided.

UseCase:
Best suited for **fractional knapsack** or when approximate solutions are acceptable.

Pure Recursion (Brute Force) - This is the most straightforward but computationally expensive approach. It explores **all possible combinations** of items (include or exclude) to find the one with the maximum value that fits in the knapsack.

Steps:

1. For each item, choose between including or excluding it.
2. Recurse for the next item and track total value and weight.
3. Return the maximum value obtained

Aantages:

- Conceptually simple.
- Always finds the optimal solution.

Disadvantages:

- Time complexity is **$O(2^n)$** — grows exponentially.
- Not practical for large datasets.
-

UseCase:
Good for understanding the problem conceptually or testing with small input sets.

SCENARIO DESCRIPTIONS

The project includes multiple predefined scenarios that simulate real-world cargo loading and resource allocation problems. Each scenario represents a different difficulty level and helps test the performance of various algorithms under changing constraints and item distributions.

SMALL DELIVERY

A simple logistics scenario with a very limited carrying capacity. It involves 3–4 light packages, each with different weights and profits. This scenario is designed to test basic algorithm decision-making in selecting the best combination without exceeding a small weight limit. It serves as an introductory example for understanding how knapsack optimization works.

CROSS – COUNTRY FREIGHT

This medium-level scenario represents long-distance trucking operations. It includes multiple types of cargo—such as electronics, furniture, and tools—each differing in weight and value. The larger capacity makes the selection more complex compared to smaller cases. It tests how algorithms balance high-value but heavy items against lighter, moderately profitable ones.

CONTAINER SHIP LOADING

A high-difficulty scenario involving heavy, large containers with significant profit values. The overall capacity is much higher, increasing the number of possible combinations. This scenario tests algorithm scalability and efficiency in handling industrial-level cargo operations. It is ideal for evaluating advanced methods such as Branch and Bound.

RESOURCE ALLOCATION

This scenario focuses on distributing limited resources—such as budget, manpower, or materials—across multiple tasks. Each task has a requirement (weight) and benefit (value). The objective is to maximize total productivity without exceeding resource limits. It reflects common decision-making problems in operations and project management.

HIKING BACKPACK

The **hiking backpack problem** is a direct and intuitive example of the knapsack problem. A hiker has a backpack with limited weight capacity and must choose from a set of items such as food, clothes, tools, and gear, each with its own weight and usefulness. The aim is to **maximize the total utility or comfort of selected items** without exceeding the backpack's carrying limit. This real-life situation illustrates the essence of the knapsack problem — balancing constraints (capacity) with benefits (value) to achieve an optimal combination.

These scenarios provide a broad and practical foundation for analysis, supporting algorithm study in logistics, finance, resource management, and daily planning contexts.

IMPLEMENTATION

This section explains the step-by-step working of the **Cargo Loading Optimizer**, a GUI-based project designed to demonstrate and compare different solution techniques for the **Knapsack Problem**. The system provides an interactive interface where users can input cargo details, choose suitable algorithms, and visualize results in both tabular and graphical form.

System Overview

The **Cargo Loading Optimizer** is implemented using a graphical user interface (GUI) that allows users to simulate the process of loading cargo items into a truck or container. Each cargo item has an associated **weight** and **profit/value**, and the system determines the most optimal combination of items to maximize profit without exceeding the truck's weight capacity.

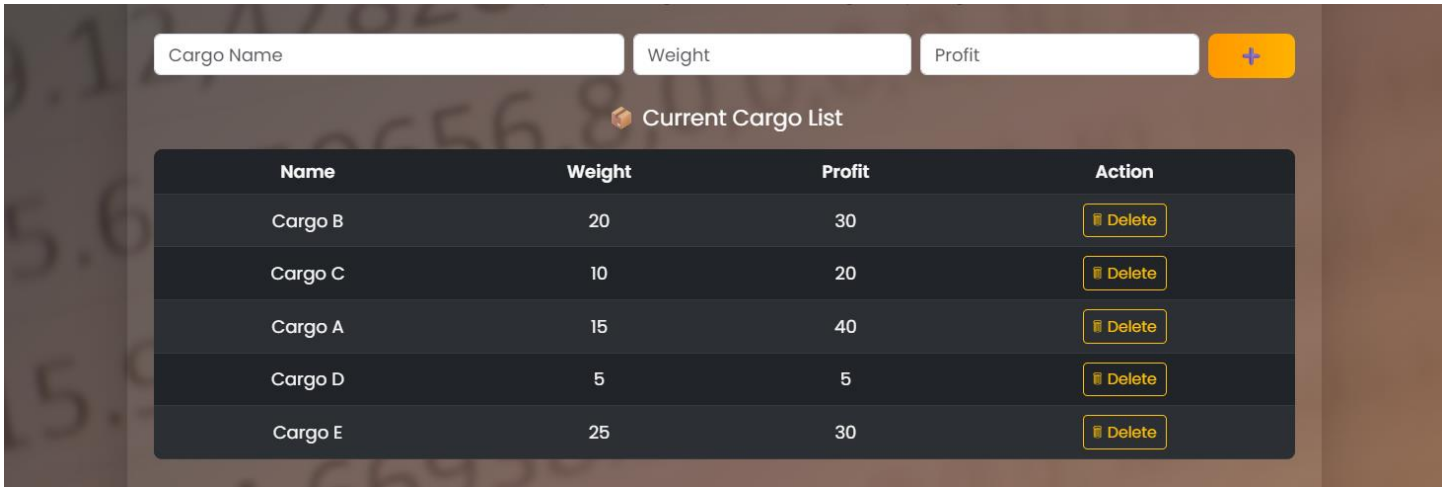
The project integrates multiple algorithms — **Dynamic Programming (Tabulation and Memoization)**, **Branch and Bound**, **Greedy**, and **Pure Recursion** — allowing users to compare their performance and results.

Input Section:

In the **input panel**, users specify:

- The **number of cargo items**.
- Each item's **weight** and **profit/value**.
- The **maximum loading capacity** of the truck or container.

The GUI accepts the data manually (via entry fields). Once the data is entered, it is validated to ensure that all weights and values are positive and that capacity is not zero.

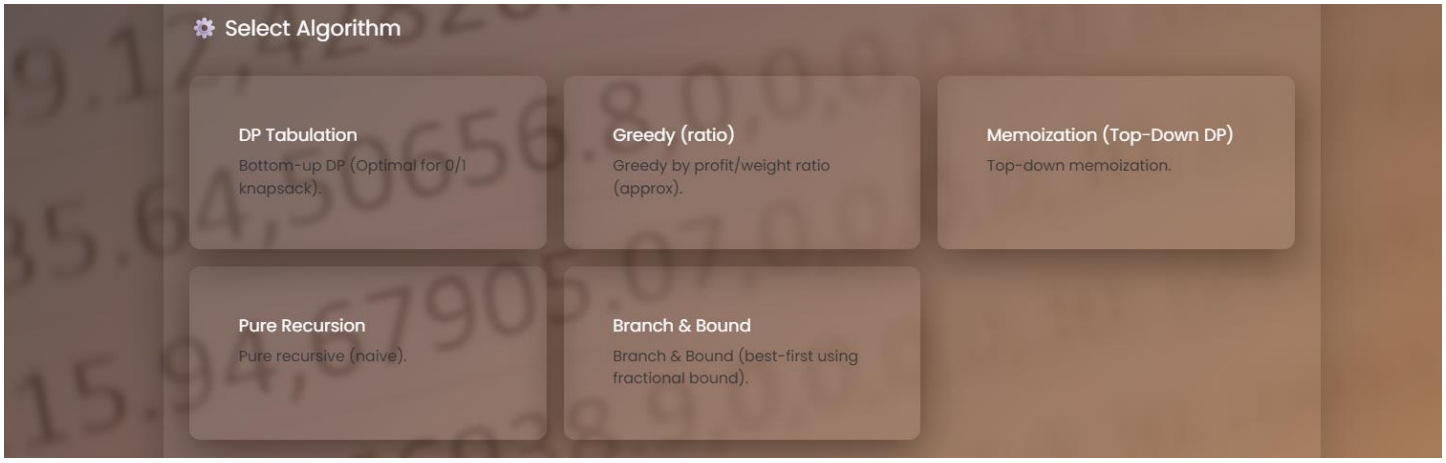


Algorithm Section:

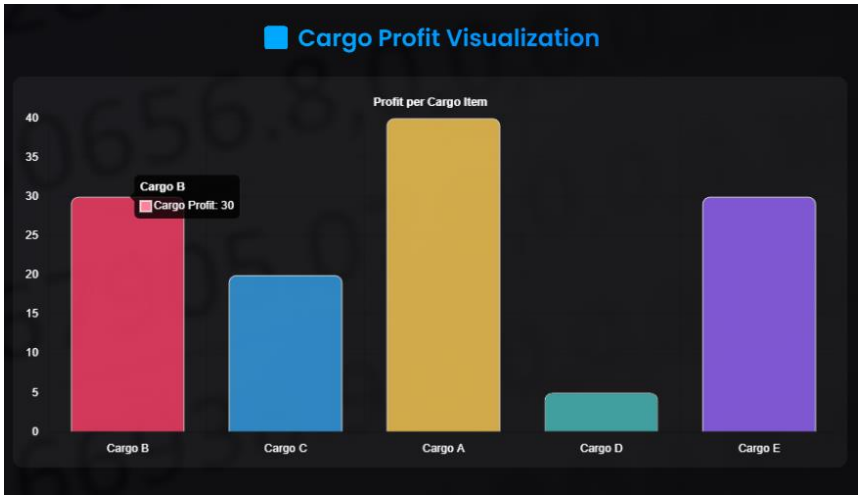
The system allows users to **choose the algorithm** by selecting one of the available options from a dropdown or button panel.
The implemented algorithms include:

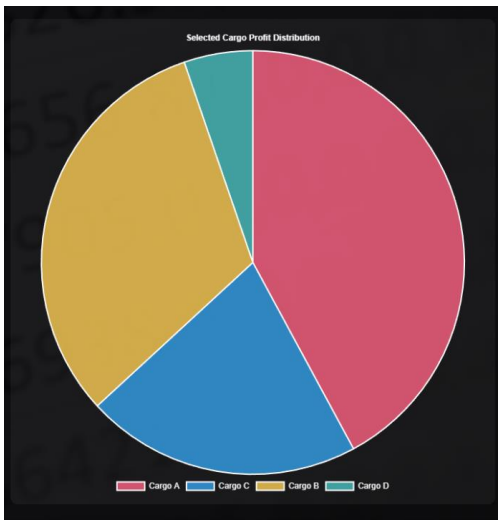
1. **Pure Recursion** – Basic brute-force approach exploring all combinations.
2. **Dynamic Programming (Memoization)** – Top-down optimization with caching.
3. **Dynamic Programming (Tabulation)** – Bottom-up approach using iterative DP table filling.
4. **Branch and Bound** – Pruning-based optimization for faster exact solutions.
5. **Greedy Algorithm** – Fast heuristic approach suitable for fractional knapsack cases

This feature lets users understand how different algorithms behave for the same input dataset, both in terms of **accuracy** and **execution time**.



Execution Process:





After selecting the desired algorithm, the user clicks the **"Run" or "Compute"** button to start execution. The backend algorithm processes the data and computes the **maximum achievable profit** while identifying which items should be included in the optimal cargo load.

The GUI internally calls the selected algorithm's function, processes the data, and displays the following:

- The **optimal total profit/value**.
- The **list of items included** in the optimal cargo load.
- The **execution time** taken by the algorithm.

Display of Results:

Once computation is complete, the results are displayed in an **output panel** with clear labels and data tables. The output typically includes:

- **Maximum Profit Obtained**
- **Total Weight Loaded**
- **Items Selected (with IDs, weights, and values)**
- **Execution Time** (for performance comparison)

Additionally, the GUI provides **visual representation using graphs**:

- **Bar graphs or pie charts** compare algorithm performance (e.g., runtime or profit).
- **Comparison charts** show how each algorithm performs on the same dataset.

These visualizations make it easier to analyze efficiency and trade-offs between different algorithmic approaches.

Analysis and Comparison:

The system also supports **multi-run analysis**, allowing users to run the same dataset with different algorithms consecutively.
This feature helps in:

- Comparing **execution times** between algorithms.
- Observing how **solution quality** varies.
- Understanding the **time–space trade-off** in practical scenarios.

For example:

- **Dynamic Programming** gives the optimal result efficiently.
- **Branch and Bound** achieves similar accuracy with fewer state explorations.
- **Greedy Algorithm** runs fastest but may give a slightly suboptimal solution.
- **Pure Recursion** confirms the correct result but with the longest runtime.

Tools and Technologies Used:

- **Programming Language:** Python
- **GUI Framework:** Tkinter (or PyQt, based on your implementation)
- **Plotting Library:** Matplotlib (for result graphs)
- **Data Handling:** Lists and matrices for algorithmic computation
- **Development Environment:** Visual Studio Code / Google Colaboratory

Flowchart of the Methodology:

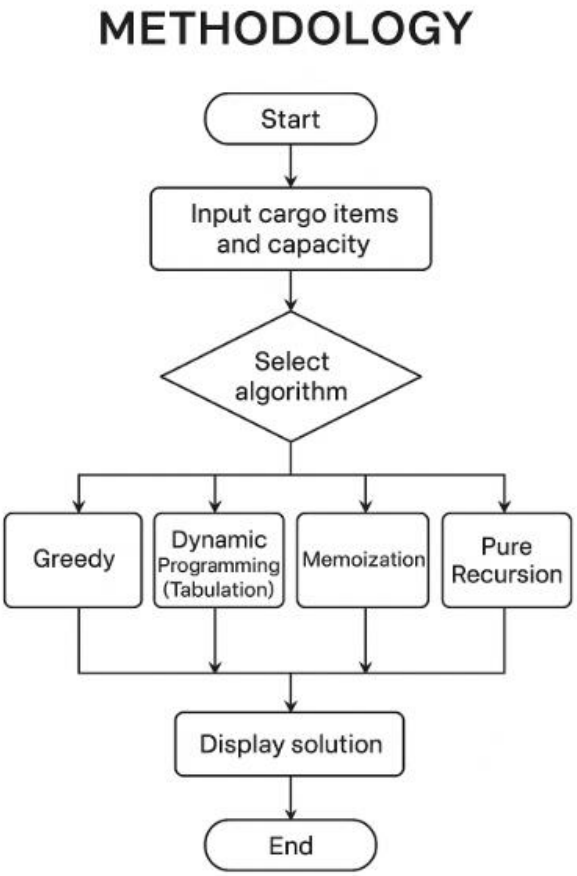


Fig. Flowchart of the Methodology

Flowchart of Architecture:

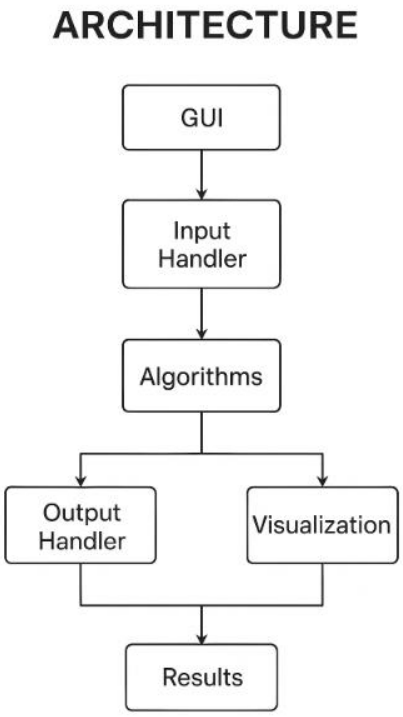


Fig. Flowchart of Architecture of System

CONCLUSION

The Cargo Loading project successfully demonstrates the practical application of optimization algorithms in solving real-world resource allocation problems. By implementing multiple variants of the 0/1 Knapsack Problem—such as Dynamic Programming, Greedy Approach, Memoization, Pure Recursion, and Branch & Bound—the project highlights how different algorithmic strategies perform under varying constraints, complexities, and datasets. Each algorithm contributes unique strengths, whether in accuracy, computational efficiency, or scalability, offering a comprehensive understanding of algorithmic problem-solving in optimization scenarios.

The web-based interface, developed using Python, Flask, HTML, CSS, and JavaScript, further enhances the usability of this system. Users can easily input cargo data, select algorithms, and visualize results through dynamic bar and pie charts powered by Chart.js. This makes the learning process interactive and enables users to clearly interpret how different algorithms evaluate the same cargo-loading scenario. The visual comparison feature strengthens the project's academic and practical relevance by providing performance insights at a glance.

Moreover, the project emphasizes the importance of decision-making under constraints—common in logistics, supply chain management, finance, and resource planning. By simulating realistic scenarios and demonstrating algorithmic decision processes, the system bridges theoretical computer science concepts with real-life applications.

In conclusion, this project not only fulfills its goal of optimizing cargo loading but also showcases the importance of algorithm selection in achieving the best results. Through its strong integration of backend logic, modern UI design, and visualization tools, the system stands as a comprehensive educational model for understanding optimization algorithms and their practical impact.

REFERENCES

1. “Knapsack problem” — Wikipedia. https://en.wikipedia.org/wiki/Knapsack_problem Wikipedia

2. “Research on Optimization of Knapsack Problem in Logistics Distribution” — ResearchGate. https://www.researchgate.net/publication/347656195_Research_on_Optimization_of_Knapsack_Problem_in_Logistics_Distribution ResearchGate

3. “Knapsack Problem – an overview” — ScienceDirect Topics. https://www.sciencedirect.com/topics/computer-science/knapsack-problem ScienceDirect

4. “Modeling the 0-1 Knapsack Problem in Cargo Flow Adjustment” — MDPI. https://www.mdpi.com/2073-8994/9/7/118 MDPI

5. “Knapsack Problems” — Springer (D. Pisinger). https://link.springer.com/chapter/10.1007/978-1-4613-0303-9_5 SpringerLink

6. “A three-dimensional container loading algorithm for solving the strongly heterogeneous Knapsack Problem” — ScienceDirect. https://www.sciencedirect.com/science/article/pii/S2192437625000160 ScienceDirect

7. “Solving single and multi-objective 0/1 Knapsack Problem” — Banaras Hindu University (PDF). https://www.bhu.ac.in/Images/files/36%283%29.pdf Banaras Hindu University

8. “A Study on Greedy Technique in Container Loading & Knapsack Problem” — IJSRST. https://ijsrst.com/IJSRST218389 Ijsrst

9. “Practical constraints in the container loading problem” — ScienceDirect. https://www.sciencedirect.com/science/article/pii/S0305054820303038 ScienceDirect

10. “An In-Depth Analysis of the Knapsack Problem” — Medium article. https://medium.com/%400x_Rorschach/an-in-depth-analysis-of-the-knapsack-problem-1d13714e3ead