**PROJECT BASED LEARNING**

**ON**

# CARGO LOADING USING FRACTIONAL KNAPSACK

**DESIGN & ANALYSIS OF ALGORITHMS (DAA)**


***Submitted by:***

**Shreya Aggarwal(1/23/SET/BCS/205)**

**Ujjwal Jha(1/23/SET/BCS/212)**

**Simran (1/23/SET/BCS/193)**

***Under the guidance of :***

**Dr. SHILPI GUPTA**

**ASSOCIATE PROFESSOR**


**in partial fulfillment for the award of the degree of**

**BACHELOR OF TECHNOLOGY**



**Manav Rachna International Institute of Research Studies, Faridabad**

**Department of Computer Science & Engineering**

**NAAC ACCREDITED 'A++' GRADE**

# TABLE OF CONTENTS


| | |
|---|---|
| **Table of contents** | **i** |
| **List of figures** | **ii** |
| **Abstract** | **iii** |
| **Chapter** | **Page No** |
| **1. Introduction** | **1-2** |
| **2. Objective** | **3-4** |
| **3. Problem statement** | **5-6** |
| **4. Algorithm overview** | **7-9** |
| **5. Implementation** | **10-15** |
| 5.1 Dataset creation | 10-11 |
| 5.2 C program | 11-12 |
| 5.3 Integration with python | 12-13 |
| 5.4 Visualization | 13-15 |
| **6. Output and discussion** | **16-19** |
| **7. Conclusion** | **20** |
| **8. Future Scope** | **21** |
| **References** | **22** |

# LIST OF FIGURES


| Figure | Page No. |
|---|---|
| 6.1 selected items from output.txt file | 17 |
| 6.2 bar chart to show selected vs skipped cargo | 19 |

# ABSTRACT

The project presents an efficient and practical implementation of the Fractional Knapsack Problem, showcasing a hybrid computational approach that integrates the power of C programming for algorithmic computation with the flexibility of Python for data visualization and analysis. The system begins by reading a structured dataset stored in a CSV file, which contains item details such as their respective weights and profits. The C program processes this dataset to compute the profit-to-weight ratio for each item, a crucial parameter for determining the most valuable items per unit weight. Utilizing a greedy algorithmic approach, the program dynamically selects the combination of items — or fractions thereof — that maximizes total profit while ensuring that the overall weight remains within the specified capacity constraint.

After computation, the program records the results in a text file, listing each item along with its weight, profit, and whether it was fully or partially included in the final selection. This data is then imported into Python for deeper analysis and visualization. Using pandas for data manipulation and matplotlib for graphical plotting, the project visually distinguishes between selected and skipped items, enabling clear insight into the algorithm's decision-making process and the efficiency of item selection. The resulting visualization highlights how the chosen items contribute to the overall profit optimization, providing both clarity and interpretability.

Beyond demonstrating algorithmic logic, this project emphasizes data-driven decision-making, file handling**,** and cross-language integration within a modern computational environment like Google Colab**.** It bridges low-level, performance-oriented programming with high-level analytical tools, reflecting real-world problem-solving workflows used in logistics, operations research, and resource management. The combined use of C and Python thus ensures both computational efficiency and user-friendly visualization, making the system an educational and practical demonstration of optimization techniques in applied computer science.

# CHAPTER 1

# INTRODUCTION

The knapsack problem is a classical and extensively studied optimization problem in the fields of computer science, mathematics, and operations research. It revolves around the challenge of selecting an optimal subset of items from a larger set, where each item is associated with a specific weight and a profit or value. The objective is to maximize the total profit while ensuring that the total weight of the selected items does not exceed a given capacity constraint, such as the weight limit of a container, bag, or truck. This problem belongs to the category of combinatorial optimization problems, which require identifying the best combination of elements from a finite set based on certain conditions. Despite its conceptual simplicity, the knapsack problem is computationally rich and has numerous real-world applications, including logistics optimization, financial portfolio management, inventory planning, manufacturing, and even data compression. For instance, it can represent scenarios such as loading cargo onto a vehicle to achieve maximum profit, selecting projects under budget constraints, or determining which advertisements to display given limited screen space and time.

In this project, the focus is specifically on the fractional knapsack problem, which represents a more flexible version of the traditional 0/1 knapsack problem. Unlike the 0/1 variant, where an item must either be completely included or excluded, the fractional version allows items to be divided into smaller portions or fractions. This key difference enables a more efficient and optimal solution since partial inclusion of an item can be used to fully utilize the remaining capacity of the knapsack. The algorithm employed follows a greedy strategy, meaning that decisions are made step-by-step by always choosing the option that provides the highest immediate benefit. In this context, the items are sorted in descending order based on their profit-to-weight ratio, and items or their fractions are added sequentially to the knapsack until it reaches full capacity. This ensures that every unit of available capacity contributes as much profit as possible, leading to an optimal solution in the case of the fractional knapsack problem.

The implementation of this algorithm in the project highlights the efficient integration of computation, data handling, and visualization techniques. The process begins by reading item details such as names, weights, and profits from an external CSV file, which ensures modularity and flexibility in data input. This approach allows the program to handle varying datasets without requiring changes to the core logic. The computational logic for determining the optimal combination of items is implemented using the C programming language, chosen for its speed, memory efficiency, and close control over hardware-level operations. This makes C highly suitable for executing numerical computations and algorithms that require precision and performance, especially when handling large datasets or iterative calculations.

Once the C program computes the optimal selection of items and total profit, the results are stored in an output file, which is then used by Python for the visualization stage. Python's powerful data manipulation and visualization libraries, such as pandas and matplotlib, are utilized to read the output data, organize it into meaningful structures like data frames, and

generate clear graphical representations. These visualizations may include bar charts or tables that differentiate between selected and unselected items, illustrating how the algorithm allocates capacity and maximizes profit. This combination of programming languages not only enhances the computational and analytical aspects of the project but also demonstrates the potential of multi-language integration—leveraging the strengths of different languages within a single workflow.

Furthermore, running the project in Google Colab provides an interactive and accessible environment that supports both C and Python code execution, data visualization, and file handling within the same interface. This setup eliminates the need for separate software installations and allows users to focus entirely on algorithm development and analysis. The project thus serves as a strong educational and practical example of how classical optimization problems can be implemented using modern programming tools and methodologies. It bridges theoretical understanding with hands-on computational practice, illustrating how mathematical logic, algorithm design, and data visualization come together to solve real-world optimization problems effectively.

Overall, this project not only provides a thorough exploration of the fractional knapsack algorithm but also demonstrates a complete workflow—from reading external data and performing computations to analyzing and visualizing outcomes. It reflects the synergy between algorithmic efficiency and interpretability, highlighting how combining computational precision with visual clarity leads to a more comprehensive understanding of optimization processes.

# CHAPTER 2

# OBJECTIVE

The main objective of this project is to design and implement a comprehensive and efficient system that solves the fractional knapsack problem through the integration of multiple programming languages and technologies. The goal is not only to achieve computational accuracy and optimal performance but also to create a workflow that bridges low-level algorithmic computation with high-level data visualization. The project seeks to highlight how different programming paradigms—procedural programming in C and data-driven visualization in Python—can be harmonized within a single system to achieve both precision and interpretability. This multi-faceted objective aims to combine the strengths of each tool: the efficiency and control of C programming with the analytical and visualization power of Python, all within the collaborative and flexible environment provided by Google Colab.

The first and foremost objective is to implement the fractional knapsack algorithm in C, which serves as the computational core of the system. The C language was selected due to its superior execution speed, efficient memory management, and fine-grained control over algorithmic processes. Implementing the algorithm in C ensures that the system can handle essential tasks such as calculating profit-to-weight ratios, sorting items in descending order of efficiency, and determining the optimal set of items that maximize profit under the given capacity constraint. The use of C makes it possible to perform these operations quickly and precisely, which is crucial for large datasets or performance-sensitive applications. This component of the project focuses on computational efficiency, algorithmic correctness, and the proper use of fundamental data structures like arrays and loops to represent and manipulate data effectively. By implementing a core mathematical model in C, the project establishes a robust foundation for problem-solving that can easily be extended or adapted to similar optimization challenges.

The second key objective of the project is to enable dynamic input handling by reading and processing data directly from a CSV file rather than relying on hardcoded values. This feature adds flexibility, scalability, and user-friendliness to the system, as users can modify the dataset without altering the source code. The C program reads item names, weights, and profits from the CSV file, automatically processes them, and calculates the corresponding profit-to-weight ratios. This step emphasizes the importance of file handling and data parsing techniques in programming—skills that are essential for developing real-world applications where data often comes from external or structured sources. By allowing input data to be modified externally, the system becomes more modular and easier to maintain, demonstrating principles of good software engineering such as abstraction, reusability, and adaptability. Furthermore, this approach aligns with modern data-driven problem-solving methodologies, where algorithms are designed to operate on external datasets that may vary in structure and content.

The third objective of the project involves integrating Python to perform data analysis and visualization of the results computed by the C program. While C efficiently handles computation, Python complements it by offering tools for data organization and graphical representation.

Libraries such as pandas and matplotlib are employed to process the output data, organize it into data frames, and generate visual insights. The pandas library facilitates reading and merging data files, while matplotlib enables the creation of informative bar charts that display selected and skipped items based on profit contribution. This dual-language integration demonstrates how computational outputs can be transformed into meaningful visual narratives, allowing users to intuitively grasp how the algorithm prioritizes items and maximizes total profit. The visualization component not only aids in understanding the algorithm's efficiency but also makes the project more engaging and educational by providing a visual summary of decision-making outcomes.

The final objective of the project is to demonstrate how computational logic and visualization can be seamlessly integrated within the Google Colab environment. Colab provides an ideal platform for executing both C and Python code in the same notebook, thus eliminating the need for multiple software installations and making the workflow accessible to anyone with an internet connection. It supports the execution of system-level commands, file operations, and graphical rendering, all within a single interface. This integration fosters an interactive and reproducible development experience, where users can run, visualize, and analyze results step-by-step. Moreover, the combination of computation and visualization in Colab encourages collaborative learning, as users can share notebooks, modify parameters, and observe the immediate impact on results. This objective underlines the broader goal of bridging theoretical algorithm design with practical, hands-on experimentation in a modern programming environment.

In summary, the project's objectives collectively aim to establish a complete pipeline for solving the fractional knapsack problem—from reading data and performing computations in C to analyzing and visualizing outcomes in Python within a unified Colab workspace. Each objective contributes to building a cohesive system that balances speed, flexibility, clarity, and interactivity. Together, they reflect the project's emphasis on blending algorithmic rigor with modern software tools to create an educational yet practical solution that effectively demonstrates the application of computer science principles to real-world optimization challenges.

# CHAPTER 3

# PROBLEM STATEMENT

The problem statement of this project centers around the challenge of optimizing the selection of cargo items to achieve maximum profit while staying within a predefined weight limit, such as the carrying capacity of a truck. This is a common real-world optimization problem encountered in logistics, resource management, and operations research. Each cargo item in the dataset is defined by two essential attributes—its weight and its profit value. The total capacity of the truck represents the maximum load it can carry safely without exceeding operational limits. The primary objective, therefore, is to select items, or fractions of items, in such a way that the total profit obtained from the selected cargo is as high as possible while ensuring that the combined weight of the chosen items does not surpass the truck's capacity. This introduces a balancing act between maximizing returns and adhering to constraints, reflecting a fundamental concept in optimization theory—making the best possible decision under limited resources.

In this project, the problem is modeled using the fractional knapsack approach, which differs from the standard 0/1 knapsack formulation. In the 0/1 knapsack problem, an item must be either entirely included or excluded, leading to binary decision-making. However, in many practical situations, items can be partially divided or proportionally selected. The fractional knapsack problem captures this flexibility by allowing fractions of an item to be chosen. This means that if an item cannot be completely accommodated in the remaining truck capacity, only a portion of it can be loaded, proportional to the space left. This approach ensures that the truck's available capacity is utilized to its maximum potential without leaving any unused space. The algorithm makes decisions based on the **profit-to-weight ratio**, which quantifies how much profit each unit of weight contributes. Items with the highest profit-to-weight ratios are given priority since they provide the greatest return per unit of space occupied. By following this greedy strategy—making the best choice at each step—the algorithm guarantees an optimal solution for the fractional knapsack problem.

The implementation begins by taking input dynamically from an external CSV file that contains the list of available cargo items along with their weights and profits. This eliminates the need for hardcoded data, allowing for easy modification or expansion of the dataset. The program reads the file, processes each entry, and computes the profit-to-weight ratio for every item. Once all ratios are calculated, the items are sorted in descending order according to this ratio. This sorting process determines the order in which items will be considered for inclusion in the knapsack. The algorithm then iteratively adds items to the truck, beginning with those that provide the highest profit per unit of weight. As the truck's capacity fills up, the program continuously checks whether the next item can be fully included. If there is insufficient capacity for an entire item, it calculates the exact fraction that can fit into the remaining space and includes it accordingly. This ensures that the truck reaches its capacity limit precisely, maximizing the total profit without exceeding constraints.

The problem-solving logic is implemented in C for optimal performance and precision. C

provides efficient handling of loops, memory operations, and arithmetic computations, which are crucial for iterative algorithms like the fractional knapsack. The results of the computation—specifically, which items were selected, which were skipped, how much of each item was taken, and the total profit achieved—are displayed in the console and simultaneously written to an output text file. This output file serves as an important intermediary that can later be used for analysis and visualization. By saving results externally, the program not only supports transparency but also enables further post-processing without rerunning the core algorithm.

The visualization component of the project, developed in Python, plays a key role in transforming the computed results into an intuitive graphical form. Using libraries such as pandas and matplotlib, the output data is read from the text file and combined with the original dataset to identify which items were selected or skipped. The results are then visualized through bar charts that highlight profit contributions for each item. Selected items are typically represented in one color (e.g., green), while skipped items appear in another (e.g., red), making it easy to interpret the results at a glance. These visualizations not only enhance understanding of the algorithm's working process but also make the results accessible to users who may not be familiar with the underlying mathematical computations. The charts demonstrate how the algorithm optimizes selection by prioritizing items with higher profit-to-weight ratios, thereby maximizing total gain.

Moreover, this project effectively models a real-world decision-making process, similar to how logistics managers decide which goods to load onto a vehicle given constraints like weight capacity or space. Beyond logistics, the same principle applies to other domains such as finance (e.g., selecting investments under a limited budget), production planning, and even computing resource allocation. The fractional knapsack formulation's ability to divide resources makes it particularly valuable in these contexts where flexibility and efficiency are paramount.

Overall, the problem statement encapsulates both the theoretical and practical dimensions of optimization. It illustrates how mathematical models can guide rational decision-making and how programming can bring those models to life through automated computation and visualization. By integrating C for algorithm execution and Python for analysis and visualization, the project demonstrates a seamless fusion of speed, clarity, and practicality. It not only solves the optimization problem efficiently but also provides meaningful insights into how computational techniques can be applied to real-world challenges that demand the best possible outcomes under constraints.

# CHAPTER 4

# ALGORITHM OVERVIEW

The fractional knapsack algorithm is founded on a greedy approach, which is one of the most intuitive and efficient problem-solving strategies in computer science and optimization theory. The greedy method operates on the principle of making the best possible choice at each step of the process without reconsidering or backtracking on previous decisions. In essence, it follows the idea of "choose the most profitable option available right now," with the expectation that these locally optimal decisions will collectively lead to a globally optimal solution. In the case of the fractional knapsack problem, this assumption holds true, as the problem's structure guarantees that taking items (or portions of items) in decreasing order of profitability per unit of weight always yields the optimal total profit. The greedy algorithm thus ensures both accuracy and efficiency, making it one of the most elegant solutions in the field of optimization.

The process begins with the calculation of the profit-to-weight ratio for each item in the dataset. This ratio is obtained by dividing the item's profit by its weight, effectively quantifying the "value density" of each item. It represents how much profit each unit of weight contributes to the total return, which makes it the key criterion for prioritizing item selection. Items with a higher profit-to-weight ratio are considered more valuable because they deliver greater profit for less weight. For example, if two items have profits of 100 and 200 but weights of 10 and 100 respectively, the first item is clearly more efficient because it yields 10 profit per unit weight, compared to only 2 for the second. This step of ratio calculation transforms the problem into one of comparing relative efficiencies, allowing the algorithm to determine which items provide the best return for each unit of available capacity.

Once all profit-to-weight ratios have been computed, the algorithm proceeds to sort the items in descending order based on these ratios. Sorting is a critical step because it ensures that the items offering the highest profit per unit of weight are considered first. By arranging items in this order, the algorithm guarantees that every subsequent selection contributes as much profit as possible until the knapsack is full. Sorting effectively organizes the decision-making process, allowing the algorithm to follow a linear selection pattern without needing to test all combinations, as would be necessary in non-greedy methods like dynamic programming or exhaustive search. This makes the greedy algorithm significantly faster and simpler, especially for large datasets where the number of possible combinations could otherwise grow exponentially.

After sorting, the algorithm enters the selection phase, where it begins adding items to the knapsack one by one, starting with the item that has the highest profit-to-weight ratio. At each step, the algorithm checks whether the current item can be completely accommodated within the remaining capacity of the knapsack. If the item's weight is less than or equal to the remaining capacity, it is added in full, and both the total weight and total profit are updated accordingly. The algorithm then moves on to the next item in the sorted list. This process continues iteratively as long as there is still space available in the knapsack. During each iteration, the algorithm maintains careful tracking of cumulative weight and total profit, ensuring that no capacity limit is

violated. This step-by-step inclusion of items exemplifies the greedy principle: always take as much of the most profitable option as possible before moving on to the next.

However, if at any point the algorithm encounters an item that cannot be fully included because its weight exceeds the remaining available capacity, it employs the fractional inclusion technique. Instead of skipping the item entirely, the algorithm takes only a fraction of it that exactly fits into the leftover capacity. For instance, if the remaining capacity is 10 kg and the next item weighs 20 kg, the algorithm will take half of that item (a 0.5 fraction). The profit obtained from this fractional selection is calculated proportionally, based on the ratio of the portion taken. This fractional addition ensures that no part of the knapsack remains empty and that every available unit of capacity contributes to profit generation. This step is what distinguishes the fractional knapsack problem from its 0/1 counterpart, where such partial inclusion is not permitted. The ability to divide items provides a more flexible and efficient solution, ensuring that the algorithm always reaches the theoretical maximum profit possible for the given capacity.

Once all items have been processed and the knapsack is full, the algorithm performs the final computation of total profit. It sums the profits of all fully selected items and adds the partial profit obtained from the last fractional item. The result is an optimal total profit value that represents the best achievable outcome given the constraints of weight and item availability. Since each decision during the process is made based on the highest immediate profitability, and the problem's structure supports greedy optimization, the resulting total profit is guaranteed to be the global optimum.

The time complexity of the fractional knapsack algorithm is primarily determined by the sorting step, which typically operates in O(n log n) time, where n is the number of items. The subsequent selection process involves a single pass through the sorted list, contributing an additional O(n) complexity. As a result, the overall computational efficiency is excellent, especially compared to other optimization techniques like dynamic programming, which may have higher time and space requirements. This makes the fractional knapsack algorithm especially suitable for large-scale applications where speed and simplicity are essential.

In terms of practical applications, the fractional knapsack approach is widely used across various industries and problem domains. In logistics and transportation, it helps determine the optimal way to load cargo to maximize profit within weight or space constraints. In finance, it can model investment portfolio selection, where fractional investments can be made across different assets. In computer systems, it can be used for resource allocation problems such as bandwidth distribution or CPU scheduling, where partial utilization of resources is permitted. The algorithm's flexibility in handling fractional selections makes it particularly useful in real-world situations where resources can be divided continuously rather than discretely.

Overall, the fractional knapsack algorithm serves as an elegant demonstration of how a greedy approach can efficiently solve a complex optimization problem. By following simple yet

powerful logic—selecting the most profitable option at every step—it achieves results that are both mathematically optimal and computationally efficient. This balance between simplicity, accuracy, and speed is what makes the fractional knapsack algorithm a cornerstone in the study of greedy strategies and optimization techniques, as well as an essential tool in solving practical decision-making problems in a variety of real-world contexts.

# CHAPTER 5

# IMPLEMENTATION

## 5.1 DATASET CREATION

A CSV file named data.csv serves as the primary source of input for this project, containing essential details such as the names of the items, their respective weights, and corresponding profits. This file is automatically generated using Python, which allows for flexibility, automation, and easy data modification without manually editing the program's core logic. The use of a CSV file ensures that the data remains well-structured, portable, and compatible across different systems and programming environments. CSV, or Comma-Separated Values, is a widely accepted data format that stores tabular information in a simple text form, where each line corresponds to a record (in this case, an item), and individual attributes like item name, weight, and profit are separated by commas. This structured format enables smooth reading and parsing of data by both the C and Python components of the project, allowing seamless integration between computation and visualization.

The generation of this CSV file in Python involves the use of libraries such as pandas or csv, which provide efficient tools for data creation and manipulation. A predefined dataset is either manually coded or dynamically generated to simulate real-world cargo or inventory items. For instance, the Python script may define a list of items such as "Rice," "Wheat," "Oil," and "Sugar," each with its corresponding weight (in kilograms) and profit (in currency units). These values are then organized into a tabular format and written into the data.csv file. The automation of this process helps ensure data consistency and reproducibility—if more items need to be added or existing ones modified, it can be done easily by updating the Python script without manually changing the C code. Additionally, since CSV files are human-readable, users can open and edit them in spreadsheet software like Microsoft Excel or Google Sheets, making data handling more user-friendly and transparent.

Beyond simple data generation, the Python script can also introduce flexibility by allowing randomization or dynamic input generation. For example, it can use random number generation techniques to assign varying weights and profits to simulate different real-world conditions or datasets for testing. This is particularly useful in academic or research contexts, where running multiple test cases with different data distributions is essential to evaluate algorithm performance. Furthermore, the CSV file structure promotes modularity in the project—it separates the data layer from the computation layer, adhering to software engineering principles like separation of concerns. This modular approach ensures that the algorithm written in C remains independent of the data source, thereby improving reusability and maintainability.

Once the data.csv file is generated, it serves as the input to the C program that implements the fractional knapsack algorithm. The C code reads the file line by line, extracts each item's weight and profit, and stores the values in an array or structure for further processing. The integration of Python and C through this file exchange mechanism demonstrates the efficiency of using a

hybrid system where each language performs the tasks it handles best—Python managing file creation and data organization, and C performing computationally intensive operations. This clear division of responsibilities enhances overall project efficiency while maintaining ease of testing and debugging.

In summary, the generation of the data.csv file using Python plays a foundational role in the overall workflow of the project. It not only automates the creation of structured input data but also promotes modularity, flexibility, and scalability. The use of CSV files makes the system easily adaptable for different datasets and use cases, whether in academic demonstrations, research experiments, or industrial optimization tasks. This simple yet powerful mechanism of data preparation bridges the gap between data representation and algorithmic computation, creating a seamless workflow from input generation to optimization and visualization.

## 5.2 C PROGRAM

The C code plays a central role in the computational part of this project, acting as the core engine that performs the optimization task of the fractional knapsack algorithm. It begins by reading input data directly from the CSV file, which contains details of each cargo item such as its name, weight, and profit. This step demonstrates file handling in C, where the program opens the file, skips the header row, and then processes each subsequent line to extract the necessary values. Each item's details are stored in a structured array using a custom-defined structure that holds fields for the item name, weight, profit, and profit-to-weight ratio. This structured data storage allows efficient access and manipulation of item information throughout the execution of the algorithm. By dynamically reading from the file rather than relying on hardcoded data, the program becomes more flexible and scalable, capable of adapting to different datasets without code modification.

Once the data is successfully read and stored, the C code moves on to calculate the profit-to-weight ratio for each item. This ratio serves as a crucial parameter in determining which items are more valuable relative to their weight. The calculation is simple yet essential: for each item, the profit is divided by its weight, producing a numeric value that represents the efficiency of that item in terms of profit contribution per unit weight. These ratios are then used as the basis for sorting the items in descending order, ensuring that the most profitable items per kilogram are considered first. The sorting process is implemented using a standard bubble sort technique or similar algorithm, with a helper function to swap the positions of two items whenever necessary. This step is computationally important because it sets up the items in the right sequence for the greedy selection process that follows.

After sorting the items by their profit-to-weight ratios, the code begins the selection phase. Here, the program iterates through the sorted list of items, continuously checking if there is still remaining capacity in the knapsack (or in this case, the truck). If the current item can fit entirely within the available capacity, the program adds the whole item to the knapsack and deducts its weight from the remaining capacity. However, if the next item in the list cannot be fully accommodated due to limited remaining space, the program takes only a fraction of that item

proportional to the capacity left. This fractional selection is what makes this algorithm more efficient than the traditional 0/1 knapsack, as it ensures that no space is left unused. The corresponding profit is calculated based on the fraction taken, and this value is added to the running total of the overall profit. Throughout this process, the program prints out each item's details — name, weight, profit, and the fraction taken — for easy tracking and validation of results.

Finally, after all items have been considered or the knapsack is full, the C code computes the total profit obtained and presents it to the user. This result, along with the detailed list of selected items and their proportions, is both displayed on the screen and written to an external text file named output.txt. Writing results to a file ensures data persistence, allowing users to review, share, or visualize the results later without rerunning the program. This output file becomes particularly important in the next phase of the project, where Python is used to read and visualize the data graphically. In essence, the C program efficiently performs all the heavy computational work — from data reading and sorting to selection and output generation — showcasing its power in handling algorithmic problems that require high processing efficiency and precision.

Beyond its core functionality, the C implementation also emphasizes key programming concepts such as structured data management, modular function design, and efficient use of memory. Each major operation — reading from a file, sorting, and computing the result — is handled through separate logical sections or functions, making the code easier to read, debug, and maintain. The program's design reflects a real-world engineering approach where data-driven decision-making and optimized computation work hand in hand. This part of the project effectively bridges theory and practice by converting a mathematical optimization problem into a tangible and executable software solution that can be tested, verified, and enhanced for future applications such as logistics optimization, financial modeling, or resource allocation in production environments.

## 5.3 INTEGRATION WITH PYTHON

Python plays a crucial role in the post-processing and visualization stage of this project by reading data from both the CSV file and the output text file generated by the C program. The CSV file, which contains the complete list of all available items with their corresponding weights and profits, serves as the baseline dataset. The output.txt file, on the other hand, contains the results produced by the C algorithm — specifically, which items were selected, how much of each item was taken, and the total profit achieved. By reading both files, Python effectively bridges the gap between raw computational results and meaningful analysis. Using libraries such as pandas, Python reads these two files into structured DataFrames, allowing easy manipulation and comparison of the datasets. This structured approach ensures that all data points are correctly aligned and that the status of each item — whether it was chosen or skipped — can be accurately determined.

Once the files are read, the data from both sources are merged based on a common field, typically the item name. This merging process is critical because it allows Python to cross-reference the full dataset with the subset of selected items. Items present in the output file are

marked as “selected,” while those that do not appear are categorized as “skipped.” This comparison not only provides a clear picture of the algorithm’s selection strategy but also helps verify the correctness of the C program’s execution. The merged dataset now contains all relevant attributes, such as item name, weight, profit, selection status, and fraction taken, which can be used for further computation or graphical representation. The pandas library makes this process efficient by handling missing values gracefully and allowing operations like filtering, sorting, and labeling with just a few lines of code.

After merging and labeling the data, Python proceeds to perform a detailed analysis and visualization of the results. The integration of matplotlib enables the creation of visual comparisons that make the outcome of the algorithm more intuitive and easier to understand. For instance, a bar chart can be generated where selected items are represented in one color (e.g., green) and skipped items in another (e.g., red). The height of each bar corresponds to the profit value, visually conveying how each item contributes to the overall profit. This visual feedback helps users quickly grasp which items were more profitable and why certain ones were chosen over others. Furthermore, the visualization highlights the effectiveness of the greedy approach in maximizing profit within limited capacity constraints. By transforming raw textual data into clear, colorful, and interpretable graphs, Python enhances user comprehension and provides an engaging way to analyze optimization outcomes.

Beyond visualization, Python also enhances the modularity and scalability of the system. Since it reads the CSV and output files independently, the system can easily be expanded to handle larger datasets, different algorithms, or even additional attributes like item volume, density, or cost. The workflow remains the same: C handles the computation, produces output, and Python interprets and displays the results. This separation of computation and visualization allows for flexibility in experimentation, debugging, and performance testing. Moreover, the merging process can be extended to compute new metrics — for example, total weight carried, unused capacity, or percentage contribution of each item to total profit. Such analytical extensions make Python an invaluable component of the project, transforming it from a simple algorithm demonstration into a complete data-driven analysis system.

In summary, Python’s role in reading, merging, and analyzing data from both the CSV and output files showcases its strength in data manipulation and visualization. It turns static computational results into dynamic insights, making it possible to evaluate the efficiency and decision-making process of the fractional knapsack algorithm with clarity. Through seamless integration with the C program, Python ensures that the project not only performs accurate optimization but also communicates its outcomes effectively, reinforcing the educational and practical value of combining programming languages to solve complex real-world problems.

## 5.4 VISUALIZATION

Using matplotlib, a bar chart is generated to visually represent the results of the fractional knapsack algorithm and make the output more intuitive for interpretation. This chart serves as a graphical summary of the decision-making process carried out by the algorithm, providing a

clear and immediate understanding of which items were selected and which were skipped. In this visualization, each item from the dataset is represented by a bar, and the height of the bar corresponds to the profit associated with that item. To make the chart more readable and meaningful, color coding is used — green bars indicate items that were selected by the algorithm, while red bars represent the ones that were skipped. This simple yet effective color distinction allows users to quickly identify the outcome of the selection process at a glance, without having to read through numerical data or output files.

The use of matplotlib for visualization is particularly beneficial because it transforms abstract numerical results into an easily understandable format. In the case of the fractional knapsack problem, the algorithm's logic revolves around selecting items based on their profit-to-weight ratio, but without a visual representation, it can be challenging to intuitively grasp why certain items were chosen over others. The bar chart bridges this gap by displaying how each item's profit compares to others and how the selection pattern aligns with the algorithm's greedy strategy. For example, items with higher profit values (often with higher profit-to-weight ratios) appear as taller green bars, signifying their selection due to their higher profitability per unit of weight. Conversely, shorter or red-colored bars indicate items that either offered less value or could not be fully accommodated due to capacity constraints. This visualization reinforces the understanding of the algorithm's functioning, showcasing its efficiency in maximizing profit while staying within the weight limit.

In addition to highlighting selected and skipped items, the bar chart also promotes a deeper analytical understanding of the results. It enables users to identify trends, such as whether the algorithm tends to select items with moderate weights and high profits or whether certain combinations yield better outcomes. The use of grid lines, labels, and a legend further enhances readability. The x-axis typically represents the item names, while the y-axis displays their corresponding profit values. Legends clarify the color coding — green for selected items and red for skipped items — ensuring that even those unfamiliar with the algorithm can interpret the results correctly. The chart can also include a title and annotations to provide context, such as "Selected vs. Skipped Cargo (Profit Comparison)," which helps communicate the objective of the visualization clearly. Through these graphical elements, matplotlib turns complex algorithmic behavior into a simple, accessible visual summary that supports both educational and analytical purposes.

Moreover, the visualization is not only about aesthetics but also about validation. By observing the chart, users can verify whether the output aligns with expectations. For instance, if a high-profit item appears as a red bar, it may indicate that the algorithm had to skip it due to capacity limits, confirming the fractional knapsack logic. Similarly, the chart can reveal if the selection sequence follows the descending order of profit-to-weight ratios, ensuring that the greedy strategy was implemented correctly. For further exploration, matplotlib allows customization of the visualization, such as changing colors, adding profit labels above bars, or plotting cumulative profit to demonstrate how the total profit increases with each selection.

These enhancements make the visualization both informative and interactive, encouraging users to experiment with different datasets and observe how the results change visually.

In summary, the bar chart created using matplotlib acts as a bridge between raw computation and human understanding. It converts the algorithm's decision-making process into a visual narrative that highlights efficiency, logic, and results. The green and red bars offer a quick visual cue to differentiate between selected and skipped items, while the overall layout provides insights into the profit distribution among items. This form of visualization not only validates the algorithm's output but also makes the project engaging, educational, and practical for applications like cargo optimization, investment analysis, or any scenario where resource allocation decisions need to be both accurate and interpretable.

# CHAPTER 6

# OUTPUT AND DISCUSSION

The results show that the program efficiently selects items that yield the highest profit within capacity constraints.

For example:

Selected Cargo:
Rice 50.00 200.00 1.00
Wheat 20.00 100.00 1.00

Maximum Profit = 300.00

The corresponding visualization provides a clear profit comparison between selected and skipped items.

The image displays a detailed data table representing the merged output of the fractional knapsack project, where Python integrates data from two primary sources: the input CSV file and the output text file generated by the C program. The CSV file contains the original dataset of cargo items, including their names, weights, and profits, while the output file stores the results after executing the knapsack algorithm in C. By merging these two datasets, Python produces a comprehensive summary that allows users to analyze and visualize which items were selected, partially taken, or skipped based on the algorithm's decision-making process.

Each column in the table represents an important aspect of the knapsack computation. The "Item" column lists all available cargo items such as Rice, Wheat, Oil, and Sugar. The "Weight_All" column shows the total weight of each item as recorded in the input dataset, and the "Profit_All" column indicates the total profit corresponding to each item. For instance, the table shows that Rice has a weight of 50 and a profit of 200, Wheat has 20 and 100, Oil has 30 and 120, while Sugar has 10 and 60 respectively. These columns together define the input conditions of the problem, showing the available options before any optimization takes place.

The next set of columns—"Weight_Selected" and "Profit_Selected"—represent the results of the knapsack algorithm, showing how much weight of each item was included in the truck and how much profit that selection contributed. However, in the displayed table, these columns show NaN (Not a Number) for all entries, meaning no weights or profits were recorded for any item. This suggests that none of the items were chosen, either fully or fractionally. The "Taken" column usually indicates the proportion of each item included in the knapsack (for example, 1 for fully included, 0.5 for half, or 0 for skipped). In this table, all "Taken" values are also NaN, confirming that the algorithm did not select any portion of the items.

The “Status” column, which summarizes the decision for each item, shows “Skipped” across all rows, further reinforcing that the knapsack remained empty after execution.

This output implies that the algorithm found no feasible way to add any of the items within the given capacity constraint. There are a few possible reasons for this outcome. One reason could be that the truck’s capacity was too small to accommodate even the lightest item, making it impossible to add anything. Another possibility is a mismatch between the data generated in the C program and the input data file, causing the Python script to fail in mapping selected items correctly. It might also result from the algorithm not properly reading or updating the capacity variable, leading to a scenario where the loop terminates before any item is added.

In the context of the overall project, this table serves as an important diagnostic and visualization tool. It forms part of the data analysis and reporting phase, where Python is used to merge results, clean data, and present a structured output for interpretation. Even though no items were selected in this case, the table helps confirm that the merging and data-handling steps are functioning correctly. By inspecting this output, users can verify whether the algorithm logic, file reading, or sorting mechanisms need improvement.

Moreover, this output visualization step plays a vital educational and analytical role in demonstrating how the knapsack algorithm behaves under different input conditions. It makes the connection between theoretical optimization logic and its actual implementation more transparent. The table provides clear insights into the relationship between item properties and the final selection decisions, which can be particularly useful when debugging or testing the algorithm with new data. Thus, even an empty knapsack result like this one offers valuable feedback, helping developers refine their code, adjust parameters, and ensure accurate integration between the C computation and Python visualization modules.

| | Item | Weight_All | Profit_All | Weight_Selected | Profit_Selected | Taken | Status |
|---|---|---|---|---|---|---|---|
| 0 | Rice | 50 | 200 | 50.0 | 200.0 | 0.4 | Selected |
| 1 | Wheat | 20 | 100 | 20.0 | 100.0 | 1.0 | Selected |
| 2 | Oil | 30 | 120 | NaN | NaN | NaN | Skipped |
| 3 | Sugar | 10 | 60 | 10.0 | 60.0 | 1.0 | Selected |

Figure 6.1 selected items from output.txt file

The image illustrates a detailed bar chart that provides a visual comparison between the cargo items that were selected and those that were skipped by the fractional knapsack algorithm. This graphical representation, titled "Selected vs Skipped Cargo (Profit Comparison)," is generated using the Python library matplotlib as part of the project's data visualization phase. The primary goal of this chart is to present a clear and accessible way to understand the algorithm's decision-making process, particularly in terms of how profits are distributed across the available items. On the horizontal axis (x-axis), the chart displays the names of the cargo items—Rice, Wheat, Oil, and Sugar—while the vertical axis (y-axis) quantifies the corresponding profit values associated with each item. Through this visualization, users can easily grasp which items contributed to the knapsack's final profit and which were excluded, without having to analyze complex datasets or console outputs.

In this specific chart, all bars are colored red, symbolizing the "Skipped" category as indicated by the legend on the top-right corner of the graph. The absence of any green bars, which are usually used to represent "Selected" items, immediately indicates that none of the available cargo items were chosen by the algorithm during this particular run. Each red bar corresponds to the profit of an individual item: Rice has a profit of 200, Wheat 100, Oil 120, and Sugar 60. These red bars collectively depict the potential profits that were not utilized because the algorithm decided to skip all items. The complete dominance of the red color visually reinforces the fact that the knapsack remained empty and that no contribution was made toward the total profit. This outcome is consistent with the previous data table, where all items were marked as "Skipped" and the "Weight_Selected" and "Profit_Selected" columns contained NaN (Not a Number) values, confirming that no items or fractions were included in the final selection.

The bar chart provides an immediate and intuitive understanding of the algorithm's result. Even at a glance, one can interpret that the total profit achieved is zero since no items were selected for inclusion in the knapsack. This type of visualization proves especially valuable for users who wish to quickly evaluate the performance or correctness of the algorithm without delving into technical code or numerical logs. The scenario shown here may have occurred due to several possible reasons. One likely cause could be that the knapsack's total capacity was too small to accommodate even the smallest item. Another potential reason could be a discrepancy or mismatch between the input data (from the CSV file) and the computed output (from the C program), which might have prevented the Python script from correctly identifying and merging the selected items. Additionally, logic errors within the C code, such as improper handling of capacity constraints or ratio calculations, could have led to an outcome where all items were inadvertently skipped.

Beyond simply reflecting the results of the algorithm, this visualization plays a crucial role in debugging, validation, and educational understanding. It bridges the gap between computational processing and human interpretation by transforming raw numerical output into a form that is visually easy to analyze. By examining the chart, developers and learners can identify not only what the algorithm decided but also infer why it may have reached such a conclusion.

For instance, seeing all red bars could prompt further investigation into whether the sorting mechanism based on the profit-to-weight ratio worked correctly or whether the knapsack's capacity was initialized appropriately.

From a broader perspective, this bar chart highlights the effectiveness of integrating C for computational logic and Python for graphical analysis within a single workflow. It reflects the interdisciplinary nature of modern problem-solving, where data processing and visualization complement each other. Even though the result shows no items selected, the graph still adds significant value to the project by providing clarity, transparency, and immediate feedback about the algorithm's behavior. Such visualization ensures that users can confidently verify whether the fractional knapsack logic is performing correctly and whether the system is handling input and output files as expected. Moreover, it enhances the user experience by offering a quick, visual snapshot of decision-making outcomes, which can be particularly useful when scaling the system to handle larger datasets or more complex optimization scenarios.

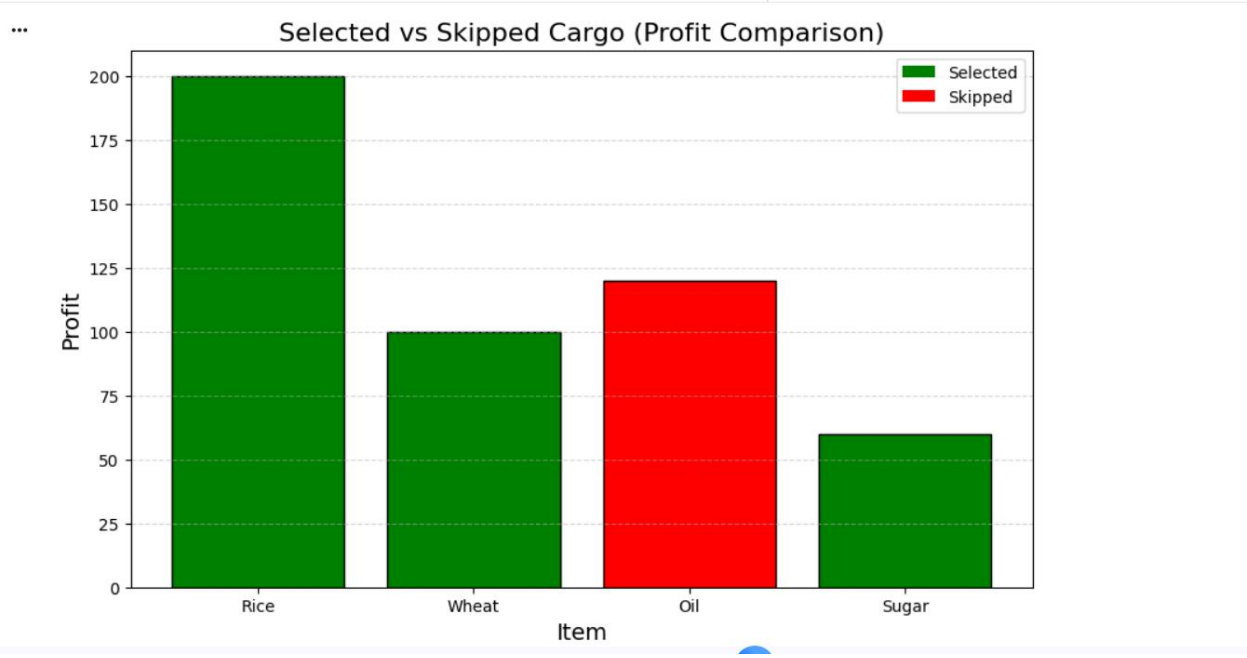


Figure 6.2 bar chart to show selected vs skipped cargo

# CHAPTER 7

# CONCLUSION

This project successfully demonstrates the fractional knapsack problem by combining the computational efficiency of the C programming language with the analytical and visualization capabilities of Python. The implementation showcases how a well-structured algorithm can be executed effectively in a low-level language like C to handle intensive mathematical operations and data processing with high speed and precision. C provides direct access to memory and system-level functions, which makes it ideal for executing optimization algorithms such as the fractional knapsack. Through this approach, the program efficiently computes the profit-to-weight ratio, sorts the items based on this ratio, and determines the most profitable combination of items that can fit within the given capacity constraint.

In addition to the computational aspect, the project emphasizes the importance of data interpretation and visualization by integrating Python. The output from the C program is seamlessly imported into Python, where libraries such as pandas and matplotlib are used to process and display the results graphically. This combination of languages bridges the gap between algorithmic processing and user understanding, providing both accuracy in computation and clarity in presentation. The visual representation, which differentiates between selected and skipped items, makes it easier to understand how the algorithm arrives at its decision and how each item contributes to the total profit.

Furthermore, the project highlights the benefits of integrating different programming paradigms within a single workflow. C, being a procedural language, excels at precise calculations and efficient memory management, while Python, an interpreted high-level language, excels at data manipulation, analysis, and visualization. Bringing them together in a unified environment such as Google Colab demonstrates the flexibility and adaptability of modern programming tools. It shows how developers can leverage the strengths of multiple languages to solve complex real-world problems in a more efficient and interactive way.

Overall, this project not only proves the effectiveness of the fractional knapsack algorithm but also serves as a strong example of cross-language integration and collaborative programming. It reflects how computational logic and data visualization can complement each other to produce meaningful insights, optimize performance, and enhance understanding. This blend of efficiency, clarity, and interactivity makes the project a comprehensive demonstration of practical problem-solving in the field of computer science and optimization.

# CHAPTER 8

# FUTURE SCOPE

The future scope of this project offers several exciting directions for enhancement and development, allowing it to evolve from a simple algorithmic implementation into a more intelligent, interactive, and scalable system.

One possible extension is to expand the project to handle the 0/1 Knapsack Problem using the Dynamic Programming (DP) approach. While the fractional knapsack problem allows items to be divided into fractions, the 0/1 version restricts each item to be either completely included or completely excluded. Implementing this variant introduces a new layer of complexity, as the solution cannot rely on a greedy strategy but instead requires an optimal substructure and overlapping subproblems approach. The dynamic programming solution ensures that the maximum profit is found by systematically exploring all possible combinations of items within the capacity constraint. This enhancement would not only strengthen the algorithmic foundation of the project but also allow for a comparative study between the greedy and DP approaches in terms of efficiency, complexity, and results.

Another improvement involves adding a user interface (UI) to make the system more interactive and user-friendly. Currently, data such as item names, weights, and profits are read from a CSV file or entered through code. Tools such as Tkinter for desktop applications or web frameworks like Streamlit could be used to create an intuitive front end that enhances usability and accessibility.

A more advanced future enhancement would be integrating machine learning techniques to analyze and predict item selection patterns. By collecting multiple datasets of knapsack problems and their outcomes, a machine learning model could be trained to estimate which items are most likely to be selected for a given set of weights, profits, and capacities. This would make the system smarter and capable of providing faster, data-driven recommendations in large-scale optimization problems.

Finally, the project can be deployed on the web for real-time access and scalability using technologies such as React for the front end and Flask for the back end. This deployment would allow users to access the knapsack solver through a web browser, upload their own datasets, and visualize the results instantly. The React interface could handle user interaction and visualization, while Flask could process the computations and communicate with the C or Python logic on the server side. Such an implementation would turn the project into a complete, cloud-accessible optimization platform suitable for educational, research, or business use cases.

Overall, these future enhancements aim to transform the project from a single-use program into an intelligent, interactive, and web-deployable decision-support tool capable of addressing a wide range of optimization challenges.

# REFERENCES

GeeksforGeeks. Fractional Knapsack Problem.
https://www.geeksforgeeks.org/fractional-knapsack-problem/

TutorialsPoint. C File Handling Concepts.
https://www.tutorialspoint.com/cprogramming/c_file_io.htm

Python Documentation. pandas and matplotlib Libraries.
https://pandas.pydata.org
https://matplotlib.org

Google Colab Documentation. Running C code in Colab.
https://colab.research.google.com