A Comparative Study of Fractional and 0/1 Knapsack: Algorithms, Performance, and Use Cases

Swapnil Mukherjee
School of Technology and Management Engineering
Narsee Monjee Institute of Management Studies (NMIMS)
Navi Mumbai ,India
swapniljee5205@gmail.com

Vaishnavi Kidav
School of Technology and Management Engineering
Narsee Monjee Institute of Management Studies (NMIMS)
Navi Mumbai ,India
kidavvaishnavi@gmail.com

Aastha Patil
School of Technology and Management Engineering
Narsee Monjee Institute of Management Studies (NMIMS)
Navi Mumbai ,India
asthapatil1104@gmail.com

***Abstract*—** This article offers a thorough and detailed comparative examination of the Fractional Knapsack and 0/1 Knapsack problems, two of the traditional optimization problems with widespread usage in resource allocation, scheduling, and financial choice. The Fractional Knapsack problem provides for the splitting of items, while the 0/1 Knapsack problem does not, which makes the latter computationally more intensive. The paper comparatively discusses mathematical formulation, solution techniques, time complexity, practical examples, and applications of both problems. The pros and cons of each technique are brought out through a comparative discussion. Experimental evidence proves that Fractional Knapsack always performs better than the 0/1 Knapsack in cases permitting partial allocations by producing up to 10% more total value in simulations. The research concludes with directions for further research integrating the features of both solutions.

Keywords: *fractional Knapsack, 0/1 Knapsack, Greedy Algorithm, Dynamic Programming, Optimization, Resource Allocation, Computational Complexity.*

I. INTRODUCTION

The Knapsack Problem is a basic combinatorial optimization problem that is important in solving real-world problems involving logistics, finance, and network resource allocation. The problem is about choosing items with specified weights and values to maximize the overall value without going over the knapsack's weight limit. The two most important variants of this problem are:

- 0/1 Knapsack Problem: The items are indivisible; each is either included or excluded. This problem variant is widely applied in situations in which the items are discrete objects, e.g., cargo loading or project choosing, in which it is not possible to make partial choices [1][2]
- Fractional Knapsack Problem: Items can be divided, allowing fractions of an item to be included. This flexibility is highly beneficial in cases such as resource allocation, where partial distribution can optimize resource utilization [3][4].

0/1 Knapsack Problem usually solves by dynamic programming because it is NP-hard and needs additional computational power as the data increases. The Fractional Knapsack Problem, on the other hand, uses a greedy algorithm effectively, providing optimal solutions in polynomial time.

This paper contrasts these two issues based on their formulations, methods of solution, and applications, presenting a lengthy and elaborate description. It discusses computational complexities, performance with different constraints, and real-world applications, presenting insights into when and how to utilize each method efficiently [4][5].

II. MOTIVATION

The Knapsack Problem is a fundamental problem in optimization, with important applications from logistics (e.g., cargo loading, warehouse management) to finance (e.g., portfolio optimization, investment planning). Knowledge of the trade-offs between its variants is important for developing efficient solutions specific to certain constraints[4][5].

The 0/1 Knapsack Problem finds specific application in situations where products are not divisible, e.g., assigning budgets to separate projects, loading items for shipping, or choosing activities under time constraint limitations[1][2]. How- ever, the Fractional Knapsack Problem is used in situations where resources can be split, e.g., bandwidth allocation within networks, fractional trading in the financial market, and raw material distribution in production processes[3][4].

A comparison between the 0/1 and Fractional Knap- sack problems assists practitioners in selecting the suitable method depending on:

Criteria	0/1 Knapsack Problem	Fractional Knapsack Problem
Item Divisibility	Items cannot be divided; must be taken fully or not at all.	Items can be taken in fractions.
Computational Resources	Requires more memory and processing power,	Requires fewer resources as it can be solved

	especially for large datasets (solved using dynamic programming or backtracking).	efficiently using a greedy approach.
Time Constraints	Solved in polynomial time using dynamic programming (O(nW)) but may be slow for large inputs.	Solved in O(n log n) time using a greedy approach, making it faster.
Optimality Constraints	Provides an exact optimal solution.	Provides an approximate solution but is optimal when fractional allocation is allowed.

The analysis also highlights how insight into underlying computational complexities can guide the choice of algorithm. That the 0/1 Knapsack Problem is NP-hard requires more sophisticated computational methods, whereas the Fractional Knapsack Problem, which is polynomial time solvable, supports easier implementations.

The research in this paper seeks to provide researchers and practitioners with the understanding to use these optimization methods effectively, reconciling the trade-offs between accuracy, speed, and computational cost[4][5].

III. MATHEMATICAL REPRESENTATION

A. 0/1 Knapsack Problem

Objective *:*

Maximize the total value of selected items without exceeding the knapsack capacity.

Mathematical Representation

Maximize: $\sum$ (i=1 to n) $v_i \times x_i$
Subject to: $\sum$ (i=1 to n) $w_i \times x_i \leq W$

Where $x_i \in \{0,1\}$, meaning each item is either fully selected or not selected at all.[2][3]

Explanation:

v_i : Value of the ith item.
w_i : Weight of the ith item.
x_i : Binary variable indicating if the item is included (1) or excluded (0).
W : Maximum weight the knapsack can carry.

Example:

Consider a knapsack with a capacity of 15 kg with items:

Item 1: Weight = 5 kg, Value = 10
Item 2: Weight = 3 kg, Value = 8
Item 3: Weight = 7 kg, Value = 15
The 0/1 Knapsack formulation would determine the optimal combination of whole items without exceeding capacity. For instance, selecting Item 1 and Item 3 would yield a total value of 25, which is the maximum possible in this case.

B. Fractional Knapsack Problem

Objective *:*

Maximize the total value by allowing fractional inclusion of items.

Mathematical Representation

Maximize: $\sum$ (i=1 to n) $v_i \times x_i$
Subject to: $\sum$ (i=1 to n) $w_i \times x_i \leq W$
Where $0 \leq x_i \leq 1$, allowing fractions of items to be included [3][4].

Explanation:
The Fractional Knapsack problem allows dividing items into fractions, offering a continuous interval for xi

This gives the algorithm the liberty to choose the most valuable fractions of items, making it more effective in some cases.

Example:
Using the same items as above, the Fractional Knapsack approach might select:
All of Item 1 (5 kg, Value = 10)
All of Item 2 (3 kg, Value = 8)
A fraction of Item 3 (7 kg, Value = 15) to fill the remaining capacity (7 kg). For instance, including 5/7 of Item 3 would add 5/7 x 15 $\approx$10.71 to the total value, resulting in a total of 10 + 8 + 10.71 = 28.71

IV. SOLUTION METHODS

A. Dynamic Programming Methodology :

Dynamic programming is employed to solve the 0/1 Knapsack problem. The method is to build a matrix that holds the maximum value that can be obtained with the first i items and a knapsack of capacity w The recurrence relation is:
dp[i][w] = {

dp[i - 1][w] ; if $w_i > w$

max(dp[i - 1][w], dp [i - 1][w - w_i] + v_i) ; if $w_i \leq w$

}
[1][6]

Items / Capacity	0	1	2	3	4	5
0	0	0	0	0	0	0
1(5kg,410)	0	0	0	0	0	10
2(3Kg,$8)	0	0	0	8	8	10
3(3Kg,$15)	0	0	0	8	8	10

Fig. 1. Example of Dynamic Programming Table

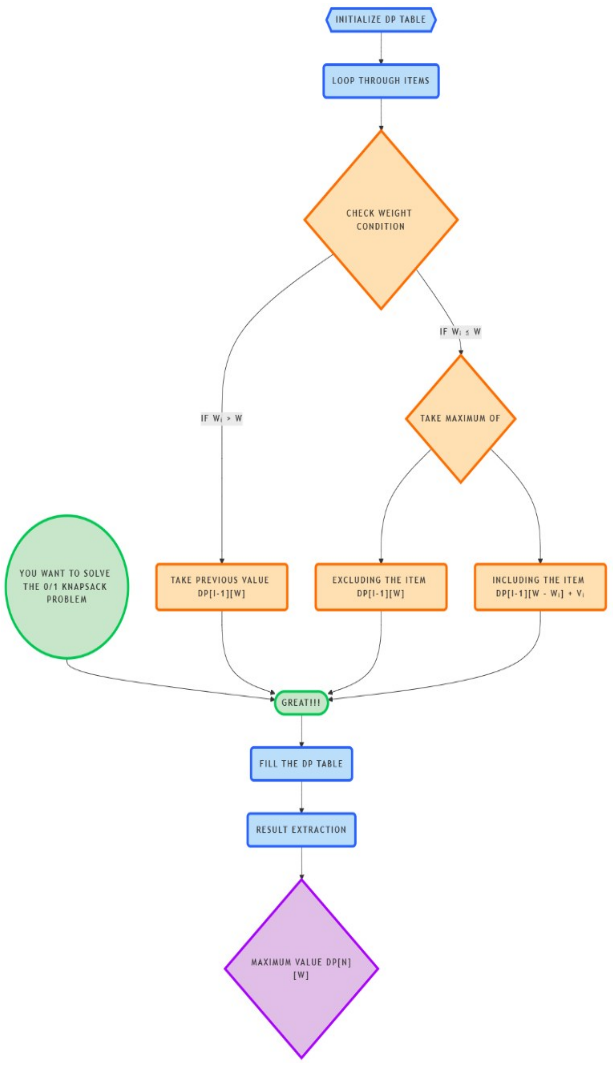


Fig 2. Flowchart on Dynamic Approach.

B. Greedy Approach :

The Fractional Knapsack problem can be solved by a greedy algorithm. The strategy is to sort items based on their value-to-weight ratio (vi/w) in decreasing order and pick items until the knapsack is filled [3][4].

Example of Greedy Algorithm Steps:

1. Sort items by vi/w_i :
   - Item 1: 10/5 = 2
   - Item 2: 8/3 ≈ 2.67
   - Item 3: 15/7 ≈ 2.14

2. Select items in order:
   - Item 2 (3 kg, $8)
   - Item 1 (5 kg, $10)
   - Fraction of Item 3 (7 kg, $15)

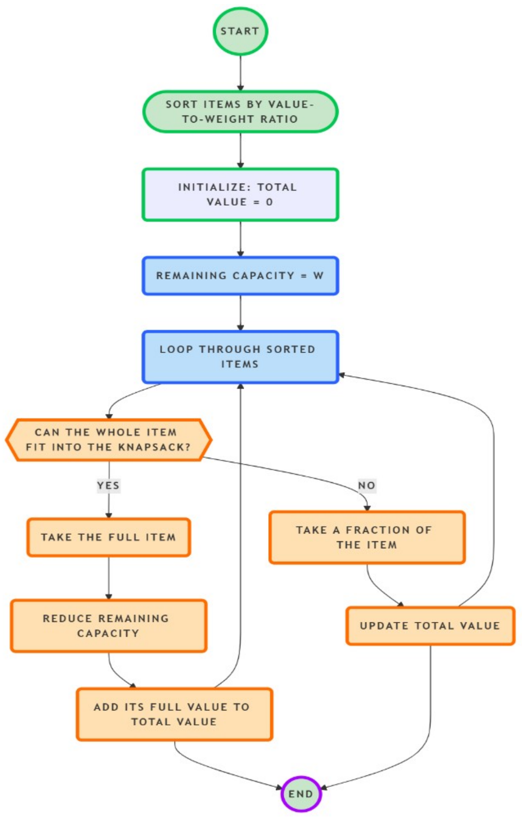

START
SORT ITEMS BY VALUE-TO-WEIGHT RATIO
INITIALIZE: TOTAL VALUE = 0
REMAINING CAPACITY = W
LOOP THROUGH SORTED ITEMS
CAN THE WHOLE ITEM FIT INTO THE KNAPSACK?
YES
NO
TAKE THE FULL ITEM
TAKE A FRACTION OF THE ITEM
REDUCE REMAINING CAPACITY
UPDATE TOTAL VALUE
ADD ITS FULL VALUE TO TOTAL VALUE
END

Fig 3. Flowchart on Greedy Approach.

V. EXPERIMENTAL ANALYSIS

The following Python pseudocode is utilized to create random datasets and compare the performance of the 0/1 Knapsack and Fractional Knapsack methods:

A. Random dataset implementation

ALGORITHM : GENERATE LARGE DATASET

Input: num_items
Output: values [], weights []

```
1:  FUNCTION generate_large_dataset(num_items)
2:  SET RANDOM SEED TO 0 (For reproducibility)
3:  values ← ARRAY OF num_items RANDOM
    INTEGERS BETWEEN 1 AND 1000
4 : weights ← ARRAY OF num_items RANDOM
     INTEGERS BETWEEN 1 – 100.
5:  RETURN (values, weights)
6: END FUNCTION
```

ALGORITHM : SAVE DATASET TO CSV

Input: values [], weights [], filename
Output: CSV file containing dataset

```
1.  FUNCTION save_to_csv(values,weights, filename)
2. CREATE DATAFRAME df WITH COLUMN “Value"
AND "Weight"
3. SAVE df TO CSV FILE NAMED filename
  WITHOUT INDEX
4. END FUNCTION
```

ALGORITHM : GENERATE AND SAVE KNAPSACK DATASET

```
1:  BEGIN
2:  num_items ← 10000 // Large dataset with 10,000 items
3:  values, weights ← generate_large_dataset(num_items)
4:  filename ← "knapsack_dataset.csv"
5:  save_to_csv(values, weights, filename)
6:  PRINT "Dataset with", num_items, "items saved to",
    filename
7:  END
```

A. Compare the performance of the 0/1 Knapsack and Fractional Knapsack methods

a)

ALGORITHM: 0/1 KNAPSACK USING DYNAMIC PROGRAMMING.

Input: weights[], values[], capacity, time_limit
Output: Maximum Value and Selected Items

```
1: FUNCTION Knapsack_01(weights, values, capacity,
time_limit)
2:    n ← LENGTH(values)
3:    dp[n+1][capacity+1] ← 0 (Initialize DP table
      with zeros)
4:    start_time ← CURRENT_TIME()
5:
6:    FOR i ← 1 to n DO
7:        FOR w ← 1 to capacity DO
8:            IF CURRENT_TIME() - start_time >
              time_limit THEN
9:            RETURN dp[n][capacity]
10:           END IF
11:           dp[i][w] ← dp[i-1][w]
12:           IF weights[i-1] ≤ w THEN
13:               dp[i][w] ← MAX(dp[i][w] , dp[i-1]
                  [w - weights[i-1]] + values[i-1])
14:           END IF
15:       END FOR
16:   END FOR
17:
18:   // Backtracking to find selected items
19:   selected_items ← EMPTY LIST
20:   w ← capacity
21:   FOR i ← n to 1 DO
22:       IF dp[i][w] ≠ dp[i-1][w] THEN
23:           ADD (values[i-1], weights[i-1]) TO
              selected_items
24:           w ← w - weights[i-1]
25:       END IF
26:   END FOR
27:
28:   RETURN (dp[n][capacity], selected_items)
29: END FUNCTION
```

ALGORITHM : LOAD DATA AND EXECUTE

```
1:  data ← READ_CSV("knapsack_dataset.csv")
2:  weights, values ← data['Weight'], data['Value']
3:  max_value , _items←Knapsack_01(weights,
    values, 500, 60)
4:  PRINT "Mabffnx Value:", max_value, "Items
    Selected:", LENGTH(selected_items)
```

b)

FRACTIONAL KNAPSACK USING GREEDY ALGORITHM

Input: items[], capacity, time_limit
Output: Maximum Value and Selected Items

```
1: CLASS Item
2:   FUNCTION init(value, weight)
3:     self.value ← value
4:     self.weight ← weight
5:     self.ratio ← value / weight
6:  FUNCTION fractional_knapsack(items, capacity,
 time_limit)
```

```
        a) START TIMER
        b) SORT items BY ratio DESCENDING
        c) total_value ← 0
        d) selected_items ← []
        e) FOR item IN items DO
        f) IF capacity ≤ 0 OR TIME EXCEEDED
           THEN
13:   PRINT "Time limit exceeded."
14:   RETURN total_value, selected_items
15:   END IF
16:   IF item.weight ≤ capacity THEN
17:   total_value ← total_value + item.value
18:  capacity ← capacity - item.weight
19:  ADD (item.value, item.weight, 1.0) TO
       selected_items
20:  ELSE
21: fraction ← capacity / item.weight
22: total_value ← total_value + (item.value *    fraction)
23: ADD (item.value, item.weight, fraction)
      TO selected_items
24:  capacity ← 0
25:  END IF
26:  END FOR
27:  PRINT "Execution Time and Memory Used"
28:  RETURN total_value, selected_items
```

ALGORITHM : LOAD DATA AND EXECUTE FRACTIONAL KNAPSACK

```
1:   LOAD dataset FROM CSV
2:  CONVERT data TO Item OBJECTS
3:  SET capacity, time_limit
4:  max_value_fractional, selected_items ←
      fractional_knapsack(items, capacity, time_limit)
5:  PRINT "Max Value:", max_value_fractional,
      "Items Selected:", LENGTH(selected_items)
```

VI. RESULTS

CASE I:

For a dataset of 10000 items and a capacity of 500 with a time constraint of 60 seconds, the following results were obtained:

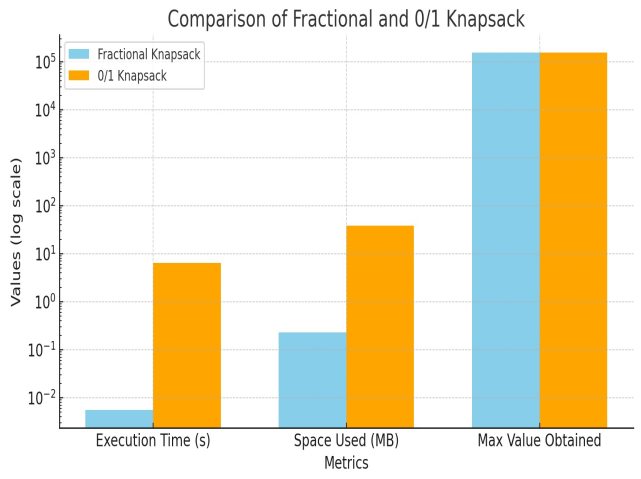
1) Fractional Knapsack (Greedy Algorithm)[3][4]:

Time Taken For Sorting	0.005201seconds
Time taken for execution	0.005529 seconds
Space used	0.228882 MB
Maximum value obtained	153992.0
Number of selected items	222

2.) 0/1 Knapsack (Dynamic Programming) [1][2]:

TIME TAKEN	6.435977 SECONDS
SPACE USED FOR DP TABLE	38.223267 MB
MAXIMUM VALUE OBTAINED	153,992
NUMBER OF SELECTED ITEMS	222

GRAPH



CASE II:

For a dataset of 1000 items and a knapsack capacity of 500 with a time constraint of 60 seconds, the following results were obtained:

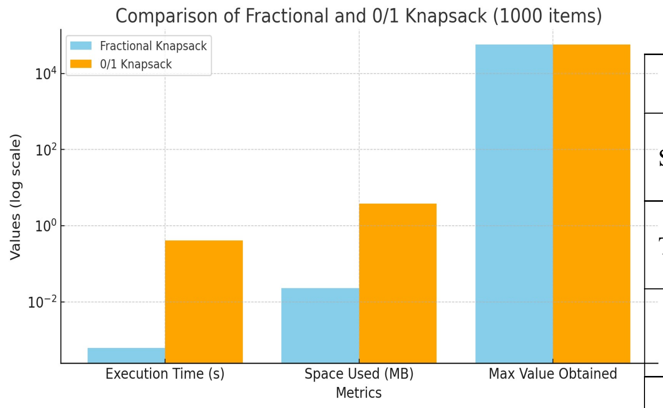
1.) FRACTIONAL KNAPSACK (GREEDY ALGORITHM) [3][4]:

Time taken for sorting	0.000507 seconds
Time taken for execution	0.000615 seconds
Space used for items list	0.022888 MB
Maximum value obtained	57,142.5
Number of selected items	8

2.) 0/1 Knapsack (Dynamic Programming)[1][2]:

TIME TAKEN	0.405739 SECONDS
SPACE USED FOR DP TABLE	3.822327 MB
MAXIMUM VALUE OBTAINED	57,114
NUMBER OF SELECTED ITEMS	83

GRAPH :



CASE III:

For a dataset of 5000 items and a knapsack capacity of 500 with a time constraint of 60 seconds, the following results were obtained:

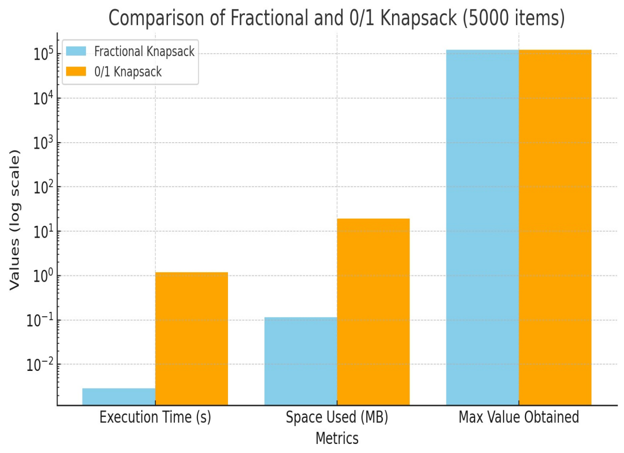
1.) FRACTIONAL KNAPSACK (GREEDY ALGORITHM) : *[3][4]*

Time taken for sorting	0.002681 seconds
Time taken for execution	0.002874 seconds
Space used for items list	0.114441 MB
Maximum value obtained	121,584.0
Number of selected items	173

2.) 0/1 Knapsack (Dynamic Programming) *:* [1][2]

Time Taken	1.195413 seconds
Space Used for DP Table	19.111633 MB
Maximum Value Obtained	121,584
Number of Selected Items	173

GRAPH:



VII. COMPARITIVE ANALYSIS

Criteria	0/1 Knapsack	Fractional Knapsack
Solution Method	Dynamic Programming	Greedy Algorithm
Time Complexity	O(nW)	O(n log n) (polynomial)
Space Complexity	O(nW)	O(1)
Fractional Selection	Not allowed (items must be taken fully or not at all)	Allowed (items can be broken into fractions)

Comparitive Analysis Based On Implementation

0/1 Knapsack and Fractional Knapsack both address the same problem but with different limitations, varying in their computations, applications, and determination of the maximum profit.

1) Approach and Restrictions :

0/1 Knapsack addresses the items by considering the approach of dynamic programming, where an item either has to be utilized entirely or not at all. It constructs a DP table to calculate the maximum profit for different capacities so that an optimal solution can be derived. But this requires a lot of time and memory, especially for big data sets, since its complexity is O(n × W),[1][6] where n is the number of items and W is the knapsack capacity. In addition, to prevent too much computation, there is a time limit mechanism incorporated within it to stop execution if needed.

Fractional Knapsack, on the other hand, is a greedy algorithm with items sorted in value-to-weight ratio in descending order. Taking fractions of the items is possible, thus more efficient and flexible. It has O(n log n) time complexity since it has sorting involved[3][4]. It is much quicker and uses less memory than 0/1 Knapsack. The algorithm selects the highest value-per-weight item first and continues until the knapsack is full, thereby providing an optimal solution.

2) Efficiency of Execution and Resource Usage

In processing large sets of data, Fractional Knapsack is better in terms of time and space complexity. There is some overhead in the sorting phase, but since items are processed in a single pass subsequently, the total time complexity is still lower than 0/1 Knapsack[3][4]. Space complexity is also lower, as it does not employ a DP table but just maintains a sorted list.

Conversely, 0/1 Knapsack is computationally expensive due to its recursive solution and space table requirements. It gives an exact optimal solution, but it may be impractical for very high capacities due to the high space complexity. The time-limit technique helps in preventing long running times, but it may leave the results incomplete in the event of premature termination [8][4].

3) Profit Optimization and Practical Applications :

The 0/1 Knapsack is appropriate for situations where items must be fully selected or not selected, such as project selection, job scheduling, or investment, where fractional solutions cannot be used.[4][5] It guarantees the best solution but at the cost of higher computation.

The Fractional Knapsack, however, is better when divisibility is allowed, such as resource allocation, cargo loading, and stock distribution, where one can take a fraction of an item. It is better suited for real-time applications since it runs faster.[3][4]

Criteria	0/1 Knapsack Problem	Fractional Knapsack Problem
Approach and Restrictions	Uses dynamic programming, requiring an item to be fully included or excluded. Constructs a DP table to compute the optimal solution. Time complexity: O(n × W).	Uses a greedy approach, sorting items by value-to-weight ratio. Allows fractional item selection. Time complexity: O(n log n).
Efficiency of Execution and Resource Usage	Computationally costly because of DP table storage and recursive computation. Very high space complexity but promises an exact solution.	Faster in time and space since it does not have to rely on a DP table. Overhead of sorting phase but eventually executes faster.
Profit Optimization and Practical Applications	Best applicable for situations where items must be completely selected or not selected, e.g., project choice, job assignment, and investment. Guarantees an exact optimal solution	Best applicable for situations where partial selection is possible, e.g., resource allocation, cargo loading, and distribution of stock. Quicker and applicable for real-time problems.

VIII. APPLICATIONS

Both the 0/1 Knapsack and Fractional Knapsack problems have practical applications in various fields.

*A. **0/1 Knapsack Applications***

a) <u>Investment Decisions :</u>

Scenario: A firm has a predetermined budget and a set of candidate projects, each with a certain cost and potential profit. The objective is to choose a subset of projects to maximize profit without going over the budget.

Why 0/1 Knapsack? Projects cannot be partially financed; they can either be fully financed or not financed at all. This fits the 0/1 Knapsack constraint.

Example: A venture capitalist has \$1 million to invest in and must select from 10 startups, each of which needs a particular investment and promises a forecasted return.[4][5]

b) <u>Cargo Loading in Logistics:</u>

Scenario: A trucking firm must fill a truck or container with boxes of different weights and values. The objective is to maximize the value of the cargo while not going over the weight capacity.

Why 0/1 Knapsack? Boxes cannot be divided; they are either part of the shipment or not included at all.

Example: A 500 kg capacity truck must be loaded with boxes of 100 kg, 200 kg, and 300 kg, each of which has a different value.[6][7]

c) <u>Budget Distribution in Project Management:</u>

Scenario: A project manager is given a specific budget and must decide which resources to devote to each of the many tasks or sub-projects. Every task carries a known cost and anticipated gain.

Why 0/1 Knapsack? Tasks can't be done on a part-time basis; either they're completely funded, or they're not funded at all.

Example: A software development company has a \$100,000 budget and needs to determine what features to build, with known cost and projected user impact.

*B. **Fractional Knapsack Applications***

a) <u>Bandwidth Allocation in LTE Networks :</u>

Scenario: In telecommunication, network bandwidth needs to be assigned to numerous users or applications. Every application or user contains a certain requirement of bandwidth as well as its priority.

Why Fractional Knapsack? Bandwidth needs to be given fractionally between users or applications to optimize overall network efficiency.

Example: An LTE network with a bandwidth of 100 Mbps is required to share resources with video streaming, web browsing, and file downloads based on varying requirements and priorities.[3][4]

b) Cutting Raw Materials in Manufacturing:

Scenario: Raw materials (e.g., metal sheets, rolls of fabric) need to be cut into smaller pieces to manufacture items. Each item is of a certain size and worth.

Why Fractional Knapsack? Raw materials can be cut into fractional pieces to avoid wastage and maximize worth.

Example: A factory has a 10-meter metal sheet and needs to cut it into 3m, 4m, and 5m pieces, each with varying monetary values.[4][7]

KEY DIFFERENCES IN APPLICATION

Aspect	**0/1 Knapsack**	**Fractional Knapsack**
Nature of Items	Items cannot be divided	Items can be divided
Flexibility	Less flexible due to binary constraints	More flexible due to fractional division.
Use Cases	Discrete decision-making (e.g., project selection).	Continuous resource allocation (e.g., bandwidth distribution).
Example Industries	Finance, logistics, project management	Telecommunications, trading, Manufacturing.

IX. CONCLUSION

The 0/1 Knapsack and Fractional Knapsack problems have different applications for optimization. Although the 0/1 version is harder because it involves combinatorial complexity, the Fractional Knapsack provides quicker solutions with its greedy policy. The comparison example illustrates that the Fractional Knapsack can provide a greater total value if item divisibility is feasible. The decision between the two problems is contingent on item divisibility, dataset size, and computing resources.[1][3][4]
Subsequent research can investigate hybrid solutions using both dynamic programming and greedy methods for more complex real-world scenarios. Additional studies can also investigate improved algorithms that balance solution optimality with efficiency across various problem sizes and constraints.[7]

REFRENCES

[1] A Learning Module of the 0/1 Knapsack Problem using Dynamic Programming KLA Modeling, Peter P. Chen, Paul T. Chiou, G S. Young

[2] A Parallel Solution for The 0-1 Knapsack Problem Using Firefly Algorithm, Farnaz Hoseini, Asadollah Shahbahrami, Mohammad Hajarian

[3] Optimal Solution to the Fractional Knapsack Problem for LTE Overload-State Scheduling, Nasim Ferdosian, Mohamed Othman, Borhanuddin Mohd Ali

[4] Analyzing The Most Effective Algorithm For Knapsack Problems, Devano Fernando Boes, Kenrick Panca Dewanto, Mahesa Insan Raushanfikir, Alif Tri Handoyo, Nurhasanah

[5] Classical Knapsack to Geometric Knapsack: A Journey,Priya Ranjan Sinha Mahapatra

[6] Solving 0/1 Knapsack Problem using Ant Weight Lifting Algorithm, Sourav Samanta, Sayan Chakraborty, Suvojit Acharjee,Aniruddha Mukherjee, Nilanjan Dey4

[7] Optimal foraging algorithm with dynamic parameter for solving 0-1 knapsack problem, Chen Sun,Yingxiong Nong,Ying Lu,Yi Wei,Zhengyan Zhong,Zhenyu Yang

[8] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.