**College of Computer Science and Engineering**
**Department of Computer Science and Artificial Intelligence**

جامعة جدة
University of Jeddah

# Project Optimization and regression: Exact and approximate methods to solve 0-1 Knapsack problem

**CCAI311**
**I3L**
**Dr. Afraa Khalefah**



| Student | ID |
|---|---|
| Yara Alshehri | |
| Taif Alharbi | |

**College of Computer Science and Engineering**
**Department of Computer Science and Artificial Intelligence**

جامعة جدة
University of Jeddah

# 0-1 Knapsack problem

## Knapsack Problem

Exhaustive search, also known as brute force search, is an algorithmic technique used to solve the Knapsack problem. It involves evaluating all possible combinations of items and selecting the one with the highest value within the weight capacity of the knapsack. While it guarantees the optimal solution, it becomes computationally expensive for larger problem instances due to its exponential time complexity of O(2^n), where n is the number of items. As a result, more efficient algorithms like dynamic programming or heuristic approaches are often preferred for solving the Knapsack problem in practice.

## Greedy algorithm for the Knapsack problem

The Greedy algorithm for the Knapsack problem is a heuristic approach that works by selecting items in order of their value-to-weight ratio, starting with the items that have the highest ratio. The two types of Greedy algorithms for the Knapsack problem are the Fractional Knapsack algorithm and the 0/1 Knapsack algorithm. The Fractional Knapsack algorithm works by selecting a fraction of the next item to fill the remaining capacity of the knapsack if it cannot be filled entirely. This algorithm is simpler to implement than the 0/1 Knapsack algorithm and can be solved in O(n log n) time complexity using a sorting step. However, the solution obtained using the Fractional Knapsack algorithm may not always be optimal. The 0/1 Knapsack algorithm, on the other hand, works by selecting either all or none of the next item to fill the knapsack, and requires a dynamic programming approach to solve it optimally. The time complexity of the dynamic programming approach is O(nW), where n is the number of items and W is the capacity of the knapsack. Overall, the choice of algorithm for the Knapsack problem depends on the specific problem instance and the desired trade-off between solution quality and time complexity.

## Dynamic Algorithm

```python
def knapsack(values, weights, capacity):
    n = len(values)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]

    solution = [0] * n  # Initialize all items as not selected
    w = capacity
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            solution[i - 1] = 1
            w -= weights[i - 1]

    best_value = dp[n][capacity]
    total_weight = sum([weights[i] for i in range(n) if solution[i] == 1])
    return best_value, total_weight, solution
```

## Greedy Algorithm

```python
def knapsack_greedy(capacity, weights, values):
    ratios = [v / w for v, w in zip(values, weights)]
    items = sorted(range(len(values)), key=lambda k: ratios[k], reverse=True)

    solution = [0] * len(values)
    total_value = 0
    total_weight = 0
    remaining_capacity = capacity

    for i in items:
        if weights[i] <= remaining_capacity:
            solution[i] = 1
            total_value += values[i]
            total_weight += weights[i]
            remaining_capacity -= weights[i]
        else:
            solution[i] = 0

    return total_value, total_weight, solution
```

## Solving The Problems

- **Problem 1**

Capacity: 165

Weights: [23,31,29,44,53,38,63,85,89,82]

Profits: [92,57,49,68,60,43,67,84,87,72]

Optimal Solution: [1,1,1,1,0,1,0,0,0,0]

```
Capacity = 165
Weights = [23,31,29,44,53,38,63,85,89,82 ]
Values =  [ 92,57,49,68,60,43,67,84,87,72]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 309
Optimal total weight: 165
Optimal solution: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

Greedy best value: 309
Greedy total weight: 165
Greedy solution: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

Percentage of error in the match: 0.0 %

- **Problem 2**

Capacity: 26

Weights: [12,7,11,8,9]

Profits: [24,13,23,15,16]

Optimal Solution: [0,1,1,1,0]

```
Capacity = 26
Weights = [12,7,11,8,9]
Values =  [24,13,23,15,16]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 51
Optimal total weight: 26
Optimal solution: [0, 1, 1, 1, 0]

Greedy best value: 47
Greedy total weight: 23
Greedy solution: [1, 0, 1, 0, 0]

Percentage of error in the match: 7.8431372549019605 %

- **Problem 3**

Capacity: 190

Weights: [56,59,80,64,75,17]

Profits: [50,50,64,46,50,5]

Optimal Solution: [1,1,0,0,1,0]

```
Capacity = 190
Weights = [56,59,80,64,75,17]
Values =  [50,50,64,46,50,5]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 150
Optimal total weight: 190
Optimal solution: [1, 1, 0, 0, 1, 0]

Greedy best value: 146
Greedy total weight: 179
Greedy solution: [1, 1, 0, 1, 0, 0]

Percentage of error in the match: 2.666666666666667 %

- **Problem 4**

Capacity: 50

Weights: [31,10,20,19,4,3,6]

Profits: [70,20,39,37,7,5,10]

Optimal Solution: [1,0,0,1,0,0,0]

```
Capacity = 50
Weights = [31,10,20,19,4,3,6]
Values =  [70,20,39,37,7,5,10]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 107
Optimal total weight: 50
Optimal solution: [1, 0, 0, 1, 0, 0, 0]

Greedy best value: 102
Greedy total weight: 48
Greedy solution: [1, 1, 0, 0, 1, 1, 0]

Percentage of error in the match: 4.672897196261682 %

- **Problem 5**

Capacity: 104

Weights: [25,35,45,5,25,3,2,2]

Profits: [350,400,450,20,70,8,5,5]

Optimal solution: [1,0,1,1,1,0,1,1]

```
Capacity = 104
Weights = [25,35,45,5,25,3,2,2]
Values =  [350,400,450,20,70,8,5,5]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 900
Optimal total weight: 104
Optimal solution: [1, 0, 1, 1, 1, 0, 1, 1]

Greedy best value: 858
Greedy total weight: 97
Greedy solution: [1, 1, 0, 1, 1, 1, 1, 1]

Percentage of error in the match: 4.666666666666667 %

- **Problem 6**

Capacity: 170

Weights: [41,50,49,59,55,57,60]

Profits: [442,525,511,593,546,564,617]

Optimal Solution: [0,1,0,1,0,0,1]

```
Capacity = 170
Weights = [41,50,49,59,55,57,60]
Values =  [442,525,511,593,546,564,617]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 1735
Optimal total weight: 169
Optimal solution: [0, 1, 0, 1, 0, 0, 1]

Greedy best value: 1478
Greedy total weight: 140
Greedy solution: [1, 1, 1, 0, 0, 0, 0]

Percentage of error in the match: 14.812680115273775 %

- **Problem 7**

Capacity: 750

Weights: [70,73,77,80,82,87,90,94,98,106,110,113,115,118,120]

Profits: [135,139,149,150,156,163,173,184,192,201,210,214,221,229,240]

Optimal Solution: [1,0,1,0,1,0,1,1,1,0,0,0,0,1,1]

```
Capacity = 750
Weights = [70,73,77,80,82,87,90,94,98,106,110,113,115,118,120]
Values =  [135,139,149,150,156,163,173,184,192,201,210,214,221,229,240]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 1458
Optimal total weight: 749
Optimal solution: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]

Greedy best value: 1441
Greedy total weight: 740
Greedy solution: [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]

Percentage of error in the match: 1.1659807956104253 %

- **Problem 8**

Capacity: 6404180

Weights:
[382745,799601,909247,729069,467902,44328,34610,698150,823460,903959,853665,551830,610856,670702,488960,951111,323046,446298,931161,31385,496951,264724,224916,169684]

Profits:
[825594,1677009,1676628,1523970,943972,97426,69666,1296457,1679693,1902996,1844992,1049289,1252836,1319836,953277,2067538,675367,853655,182602,65731,901489,577243,466257,369261]

Optimal Solution: [1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1]

```
Capacity = 6404180
Weights = [382745,799601,909247,729069,467902,44328,34610,698150,823460,903959,853665,
           551830,610856,670702,488960,951111,323046,446298,931161,31385,496951,264724,
           224916,169684]
Values =  [825594,1677009,1676628,1523970,943972,97426,69666,1296457,1679693,1902996,
           1844992,1049289,1252836,1319836,953277,2067538,675367,853655,182602,65731,
           901489,577243,466257,369261]

best_value, optimal_weight, optimal_solution = knapsack(Values, Weights, Capacity)
greedy_value, greedy_weight, greedy_solution = knapsack_greedy(Capacity, Weights, Values)

error_percentage = (best_value - greedy_value) / best_value * 100

print("Optimal best value:", best_value)
print("Optimal total weight:", optimal_weight)
print("Optimal solution:", optimal_solution)
print("\nGreedy best value:", greedy_value)
print("Greedy total weight:", greedy_weight)
print("Greedy solution:", greedy_solution)
print("\nPercentage of error in the match:", error_percentage, "%")
```

Optimal best value: 13549094
Optimal total weight: 6402560
Optimal solution: [1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]

Greedy best value: 13415886
Greedy total weight: 6323699
Greedy solution: [1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1]

Percentage of error in the match: 0.9831506077085302 %

## Comparison Between Dynamic Algorithm and Greedy Algorithm

Dynamic Algorithm involves checking all possible combinations of items to find the one that maximizes the value within the weight limit. This method guarantees finding the optimal solution, but it can be very time consuming for large problem sizes, as the number of possible combinations grows exponentially.

Greedy Algorithm  is to sort the items by their value to weight ratio and then add items to the knapsack in that order until the weight limit is reached. This method is much faster than exhaustive search, but it may not always find the optimal solution. In fact, there are cases where a greedy algorithm can produce a solution that is far from optimal.

In summary, while exhaustive search guarantees finding the optimal solution, it can be very slow for large problem size. Greedy search, on the other hand, is much faster but may not always give the optimal solution. The choice between these two methods ultimately depends on the specific problem size, time, constraints, and desired level of accuracy.