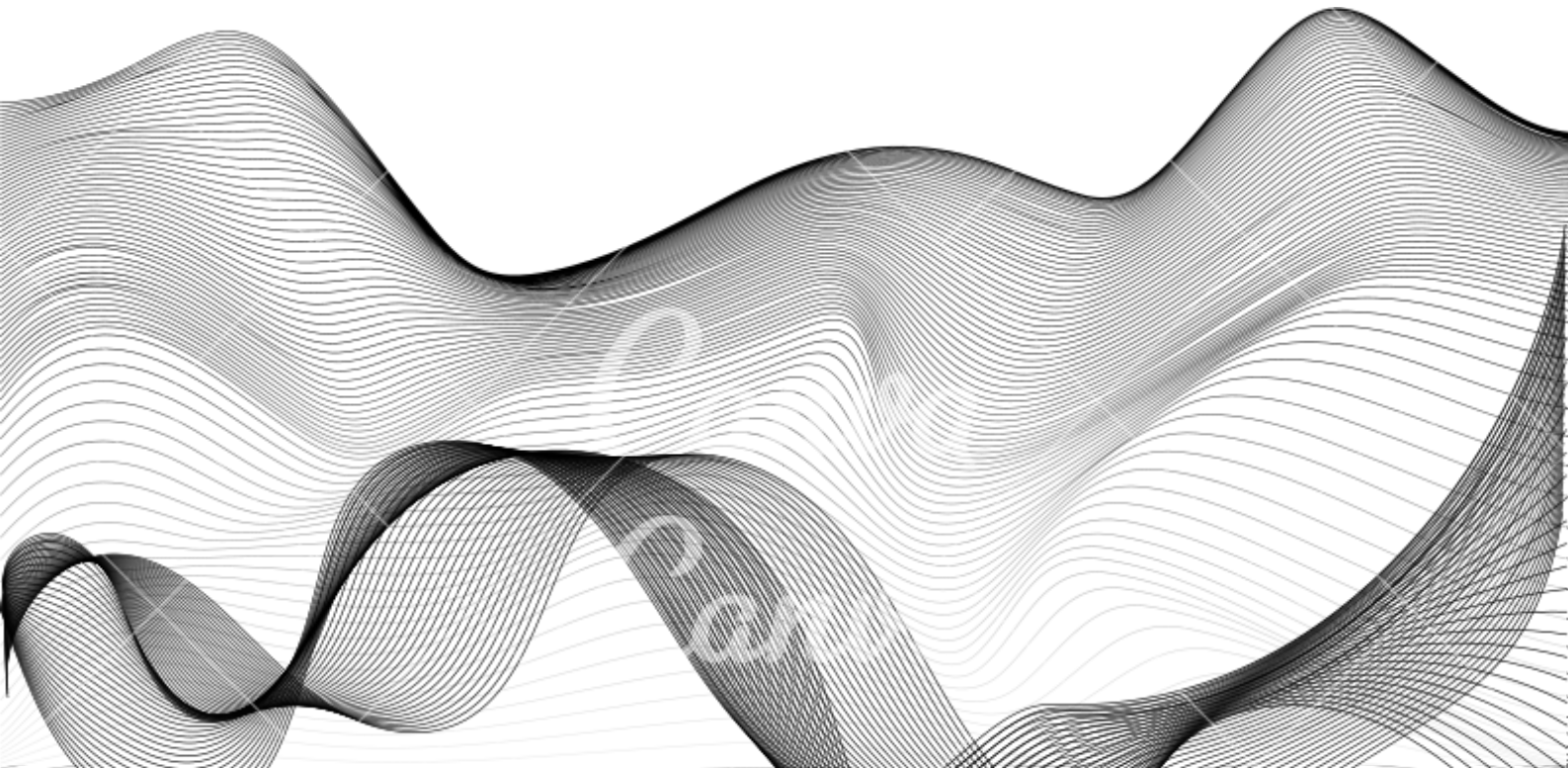
جامعة جدة
University of Jeddah

PROJECT OPTIMIZATION AND REGRESSION

EXACT AND APPROXIMATE METHODS TO SOLV 0-1 KNAPSACK PROBLEM

CCAI - I3L

RUBA ALSULAMI - 2110618
ARJWAN ALHARBI - 2110826



INTRODUCTION

The "knapsack problem" is a well-known optimization problem in computer science and mathematics. Given a set of items, each with a weight and a value, the goal of the problem is to select a subset of the items that maximizes the total value while keeping the total weight below a certain limit (the "knapsack" capacity).There are different approaches to solving the knapsack problem, including the greedy approach and the brute force approach (exhaustive search).

1- APPROXIMATE METHOD FOR THE KP

(GREEDY KNAPSACK ALGORITHM)

```python
# Greedy Knapsack Algorithm which returns the maximum value and which items are selected in terms of a binary list
def knapsack_greedy(weights, values, capacity):
    # Creating a list of tuples with items, their values, and weights
    items = list(zip(values, weights))
    # Sorting the items based on the value-to-weight ratio in descending order
    items.sort(key=lambda x: x[0] / x[1], reverse=True)
    total_value = 0  # The total value of items picked
    total_weight = 0  # The total weight of items picked
    picked_items = []  # The list to store the items picked
    for value, weight in items:
        if total_weight + weight <= capacity:
            # If the current item can be picked without exceeding the capacity,
            #pick it
            picked_items.append(1)  # Add 1 to indicate the item was picked
            total_value += value
            total_weight += weight
        else:
            picked_items.append(0)  #Add 0 to indicate the item was not picked
    return total_value, total_weight, picked_items
# Prompting the user to enter the capacity, weights, and values
capacity = float(input("Enter the knapsack capacity: "))
n = int(input("Enter the number of items:- "))
print("\n")
weights = [float(input("Enter the weight of item %d:- " %  (i+1))) for i in range(n)]
print("\n")
values = [float(input("Enter the value of item %d:- " % (i+1))) for i in range(n)]
# Calling the function and printing the results
total_value, total_weight, picked_items = knapsack_greedy(weights, values, capacity)
print("Total value:", total_value)
print("Total weight:", total_weight)
print("Picked items:", picked_items)
```

The greedy approach involves selecting items based on their value-to-weight ratio. In each step, the algorithm selects the item with the highest value-to-weight ratio and adds it to the knapsack. The algorithm continues this process until the knapsack is full or there are no more items to add. The greedy approach is fast and efficient, but it may not always produce the optimal solution.

2- EXACT METHOD FOR THE KP
(BRUTE FORCE: EXHAUSTIVE KNAPSACK ALGORITHM)

```python
def exhaustive_search_knapsack(C, W, V):
    N = len(W) # Get the number of items
    P = [0] * N # Initialize a list of 0s with length equal to the number of items
    maxValue = 0 # Initialize the maximum value and weight to 0
    maxWeight = 0
    # Iterate over all possible subsets of items
    for i in range(1 << N):
        weight = 0 # Initialize the weight and value of the current subset to 0
        value = 0
        # Iterate over all items in the subset and calculate their total weight and value
        for j in range(N):
            if (i & (1 << j)) != 0:
                weight += W[j]
                value += V[j]
        # If the total weight of the subset is less than or equal to the capacity of the knapsack
        # and its value is greater than the maximum value seen so far, update the maximum value,
        # maximum weight, and the list of picked items
        if weight <= C and value > maxValue:
            maxValue = value
            maxWeight = weight
            for j in range(N):
                P[j] = 1 if (i & (1 << j)) != 0 else 0
    # Return a list containing the maximum value, maximum weight, and the list of picked items
    return [maxValue , maxWeight] + P
```

```python
# Prompting the user to enter the capacity, weights,number of items, and values
capacity = float(input("\nEnter the capacity of the knapsack:- "))
print("\n")
n = int(input("Enter the total number of items:- "))
print("\n ENTER WEIGHT")
weights = [float(input("Enter the weight of item %d:- " % (i + 1))) for i in range(n)]
print("\n ENTER VALUE")
values = [float(input("Enter the value of item %d:- " % (i + 1))) for i in range(n)]

# Calling the function and printing the results
result = exhaustive_search_knapsack(capacity, weights, values)
print("\nItems picked in Exhaustive:", result[2:])
print("Total value:", result[0])
print("Total weight:", result[1])
print("\n")
```

The brute force approach involves generating all possible subsets of items and calculating their total value and weight. The algorithm then selects the subset with the highest value that does not exceed the knapsack capacity. The brute force approach is guaranteed to find the optimal solution, but it is computationally expensive and impractical for large instances of the problem.

ALL IN ONE CODE

```python
# Greedy Knapsack Algorithm which returns the maximum value and which items are selected in terms of a binary list
def knapsack_greedy(weights, values, capacity):
    # Creating a list of tuples with items, their values, and weights
    items = list(zip(values, weights))
    # Sorting the items based on the value-to-weight ratio in descending order
    items.sort(key=lambda x: x[0] / x[1], reverse=True)
    total_value = 0  # The Total value of items picked
    total_weight = 0  # The Total weight of items picked
    picked_items = []  # The List to store the items picked
    for value, weight in items:
        if total_weight + weight <= capacity:
            # If the current item can be picked without exceeding the capacity,
            #pick it
            picked_items.append(1)  # Add 1 to indicate the item was picked
            total_value += value
            total_weight += weight
        else:
            picked_items.append(0)  #Add 0 to indicate the item was not picked
    return total_value, total_weight, picked_items
```

```python
def exhaustive_search_knapsack(C, W, V):
    N = len(W) # Get the number of items
    P = [0] * N # Initialize a list of 0s with length equal to the number of items
    maxValue = 0 # Initialize the maximum value and weight to 0
    maxWeight = 0
    # Iterate over all possible subsets of items
    for i in range(1 << N):
        weight = 0 # Initialize the weight and value of the current subset to 0
        value = 0
        # Iterate over all items in the subset and calculate their total weight and value
        for j in range(N):
            if (i & (1 << j)) != 0:
                weight += W[j]
                value += V[j]
        # If the total weight of the subset is less than or equal to the capacity of the knapsack
        # and its value is greater than the maximum value seen so far, update the maximum value,
        # maximum weight, and the list of picked items
        if weight <= C and value > maxValue:
            maxValue = value
            maxWeight = weight
            for j in range(N):
                P[j] = 1 if (i & (1 << j)) != 0 else 0
    # Return a list containing the maximum value, maximum weight, and the list of picked items
    return [maxValue , maxWeight] + P
```

```python
# Prompting the user to enter the capacity, weights,number of items, and values
capacity = float(input("\nEnter the capacity of the knapsack:- "))
print("\n")
n = int(input("Enter the total number of items:- "))
print("\n ENTER WEIGHT")
weights = [float(input("Enter the weight of item %d:- " % (i + 1))) for i in range(n)]
print("\n ENTER VALUE")
values = [float(input("Enter the value of item %d:- " % (i + 1))) for i in range(n)]

# Calling the function and printing the results
result = exhaustive_search_knapsack(capacity, weights, values)
print("\nItems picked in Exhaustive:", result[2:])
print("Total value:", result[0])
print("Total weight:", result[1])
print("\n")

# Calling the function and printing the results
result2 = knapsack_greedy(weights, values, capacity)
print("\nItems picked in Greedy:", result2[2:])
print("Total value:", result2[0])
print("Total weight:", result2[1])
print("\n")

# Calculate the percentage error
percentageError = abs(result2[0] - result[0]) / result[0] * 100
print("Percentage Error:", percentageError, "%")

# Calculate the percentage correct
percentageCorrect = (result2[0] * 100) / result[0]
print("Percentage Correct:", percentageCorrect, "%")
```

3- COMPARISON OF THE OBTAINED RESULTS

	GREEDY KNAPSACK	EXHAUSTIVE KNAPSACK
P01	OUTPUT: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%	OUTPUT: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P02	OUTPUT: [1, 1, 0, 0, 0] CORRECT PERCENTAGE: 92.157% MISTAKE PERCENTAGE: 7.843%	OUTPUT: [0, 1, 1, 1, 0] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P03	OUTPUT: [1, 1, 0, 1, 0, 0] CORRECT PERCENTAGE: 97.333% MISTAKE PERCENTAGE: 2.667%	OUTPUT: [1, 1, 0, 0, 1, 0] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P04	OUTPUT: [1, 1, 0, 0, 1, 1, 0] CORRECT PERCENTAGE: 95.328% MISTAKE PERCENTAGE: 4.672%	OUTPUT: [1, 0, 0, 1, 0, 0, 0] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P05	OUTPUT: [1, 1, 0, 1, 1, 1, 1, 1] CORRECT PERCENTAGE: 95.333% MISTAKE PERCENTAGE: 4.667%	OUTPUT: [1, 0, 1, 1, 1, 0, 1, 1] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P06	OUTPUT: [1, 1, 1, 0, 0, 0, 0] CORRECT PERCENTAGE:85.1874 % MISTAKE PERCENTAGE: 14.8126%	OUTPUT: [0, 1, 0, 1, 0, 0, 1] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P07	OUTPUT: [1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0] CORRECT PERCENTAGE: 98.834% MISTAKE PERCENTAGE: 1.1659%	OUTPUT: [1,0,1,0,1,0,1,1,1,0,0,0,0,1,1] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%
P08	OUTPUT: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] CORRECT PERCENTAGE: 99.0168% MISTAKE PERCENTAGE: 0.98315%	OUTPUT: [1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1] CORRECT PERCENTAGE: 100% MISTAKE PERCENTAGE: 0%

4- CONCLUSION

The Knapsack algorithm is a comprehensive method that is guaranteed to find an optimal solution but is computationally expensive for large datasets. A greedy algorithm, is a faster approach that makes a locally optimal decision at each step but they may not always produce the optimal solution.The goal is to select a subset of items to maximize the total value while keeping the total weight within the capacity of the knapsack. On the other hand,exhaustive Knapsack Algorithm is a simple algorithm that tries all possible combinations of items to find the optimal solution to the knapsack problem. It works by generating all possible subsets of items and calculating the total value and weight of each subset.

5- REFERENCES

-Hebei, H. (2011). 0-1 Knapsack Problems by Greedy Method and Dynamic. Switzerland. Retrieved from https://citeseerx.ist.psu.edu/document repid=rep1&type=pdf&doi=b924b40734cd3d52a523385d4bcbe5217ad0f41c

- Etawi, N. A. (2020). 0/1 KNAPSACK PROBLEM: GREEDY VS. DYNAMIC-PROGRAMMING. Jordan. Retrieved from https://www.ijaemr.com/uploads/pdf/archivepdf/2020/IJAEMR_393.pdf

- http://www.iiitdm.ac.in/old/Faculty_Teaching/Sadagopan/pdf/DAA/greedy.pdf

-Mikhail, K. (2019). On Using Gray Codes to Improve the Efficiency of the Parallel Exhaustive Search Algorithm for the Knapsack Problem.

-google developers. (2023). The Knapsack Problem. Retrieved from https://developers.google.com/optimization/pack/knapsack?hl=en