Yaseen Haffejee : 1827555

In [1]:

```
import numpy as np
```

Question 1

The Knapsack Problem (KP) is considered to be a combinatorial optimization problem. A Knapsack model serves as an abstract model with broad spectrum applications such as: Resource allocation problems, Portfolio optimization, Cargo-loading problems and Cutting stock problems. In linear KP the objective function and constraint(s) are linear. Formulate the linear KP mathematically using the following data.

Linear Knapsack Problem: Consider the following pairs:

$$(vi, wi) = (2,7),(6,3),(8,3),(7,5),(3,4),(4,7),(6,5),(5,4),(10,15),(9,10),(8,17),(11,3),(12,6),(15,11),(6, \; (26,24)$$

with profit v_i and weight w_i for the i-th item.
Total capacity W = 30.

Solution

The mathematical formulation of the above problem is as follows:

Denote the decision variable x_i for each item i , such that: $x_i = \begin{cases} 1 & \text{if item i is chosen} \\ 0 & \text{otherwise} \end{cases}$

The total profit is given by:

$\sum_{i=1}^{21} v_i x_i$

The total money spent in buying items is given by:

$\sum_{j=1}^{21} w_j x_j$

The optimisation problem is given by:

$\max \quad \sum_{i=1}^{21} v_i x_i$

$\text{subject to} \quad \sum_{j=1}^{21} w_j x_j \leq W, \quad W = 30$

Question 2

Use the following greedy algorithm to solve the above problem in Q1.

Algorithm 1: Greedy Algorithm

1. Identify the available items with their weights and values and take note of the maximum capacity of the bag.
2. Use a score or efficiency function, i.e. the profit to weight ratio: $\frac{v_i}{w_i}(\frac{v_i}{wi} \geq \frac{vj}{wj} \cdot \cdot\cdot)$
3. Sort the items non-increasingly according to the efficiency function.
4. Add into knapsack the items with the highest score, taking note of their accumulative weights until no item can be added.
5. Return the set of items that satisfies the weight limit and yields maximum profit.

Solution

In [2]:

```
profits_and_weights = [(2, 7), (6, 3), (8, 3), (7, 5), (3, 4), (4, 7), (6, 5),(5, 4), (10,
W = 30
```

In [3]:

```
def efficiency_function(data):
    ratios = []
    ## Getting the ratio of profit over weight
    for v,w in data:
        ratios.append(v/w)

    non_increasing_indices =list(np.argsort(ratios)[::-1])
    sorted_profits_and_weights = [data[i] for i in non_increasing_indices]
    return sorted_profits_and_weights
```

In [4]:

```
profits_and_weights_sorted_non_increasingly = efficiency_function(profits_and_weights)
accumulative_weight = 0
items_in_knapsack = []
total_profit = 0
for profit,weight in profits_and_weights_sorted_non_increasingly:

    if accumulative_weight+weight <= W:
        items_in_knapsack.append((profit,weight))
        accumulative_weight += weight
        total_profit += profit

print(f"------------------- Algorithm 1 ------------------- \n")
print(f"The total weight of the items in the knapsack is: {accumulative_weight}.\n")
print(f"The total profit is: {total_profit}. \n")
print(f"The items in the knapsack are: {items_in_knapsack}. \n")
print(f"----------------------- END ----------------------- \n")
```

```
------------------- Algorithm 1 -------------------

The total weight of the items in the knapsack is: 27.

The total profit is: 64.

The items in the knapsack are: [(11, 3), (13, 4), (8, 3), (12, 6), (6, 3),
(14, 8)].

----------------------- END -----------------------
```

Question 3

Construct a penalty function of the maximization problem in Q1 with penalty parameter R = 25. Maximize the linear KP problem in Q1 via maximizing the penalty function using the iterative improvement local search (IILS). IILS uses passes and epochs. Each Pass executes a number of Epochs and each Epoch lock a variable. Epoch 1 always begins with x_0 . IILS operates as follows:

- Epochs within a Pass continue locking variables until an overall best solution (better than x0) is found when a new pass begins (with Epoch 1).
- When all the Epochs in a Pass is unable to find an overall best solution (better than x0) then IILS stops with x0 as the minimum value. Note that execution of all Epochs in a Pass means all variables are locked.
- You must start your initial solution $x_\theta = (x_1, x_2, \cdots, x_{21})^T$ such that $x_1 = x_2 = x_3 = x_4 = x_5 = 1$, and $x_i = 0$ for all $i = 6, 7, \cdots, 21.$

Solution

The constrained maximisation problem is denoted by:

$F(x; R) = \sum_{i=1}^{21} v_i x_i - R\phi(x)$

where $\phi(x) = max(0, g(x))$ and $g(x) = \sum_{j=1}^{21} w_i x_i$

$\Rightarrow F(x; R = 25) = \sum_{i=1}^{21} v_i x_i - 25max(0, \sum_{j=1}^{21} w_i x_i)$

In [5]:

```
def penalty_function(weights,x,W = 30):

    g_x = np.sum(weights*x) - W

    return max(0,g_x)

profits = np.array(profits_and_weights)[:,0]
weights = np.array(profits_and_weights)[:,1]

def F(x,R,weights = weights,profits = profits):
    vi_xi = np.sum(profits*x)
    return vi_xi - (R * penalty_function(weights,x))

def generate_subset_solutions(x0,current_max,locked_indices, n = 21, R =25):
    x_vals = []
    func_vals = []
    for i in range(n):
        new_x = np.copy(x0)
        new_x[i] = 1 - new_x[i]
        if(locked_indices[i] == 1):
            value = -7000000
        else:
            value = F(new_x,R)
        x_vals.append(new_x)
        func_vals.append(value)
        if value > current_max:
            return value, new_x,1

    # since all the new solutions are smaller, we return the one which yielded the max valu
    idx_max = np.argmax(func_vals)
    return idx_max, x_vals[idx_max]

def get_items_in_knapsack(x0,profits_and_weights):
    idx = np.where(x0 ==1)
    l = []
    for i in idx[0]:
        l.append(profits_and_weights[i])
    return l
```

In [6]:

```
x0 = np.zeros(21)
x0[:5] = 1

print(f"The initial solution x0 is: {x0}")
```

```
The initial solution x0 is: [1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0.]
```

In [7]:

```
pass_iterator = 1
R = 25
n = len(profits_and_weights)
while pass_iterator:
    locked_indices = [0]*n
    pass_iterator = 0 # remember to update it if the solution gives bigger value

    maximum_value = F(x0,R)
    # print(maximum_value)
    epoch_iterator = 1
    local_x = x0
    while epoch_iterator:
        # Calculate all new x vectors
        temporary = generate_subset_solutions(local_x,maximum_value,locked_indices,n,R)
        if(len(temporary)==2):  # None of the new x arrays are greater than current functio
            index = temporary[0]
            local_x = temporary[1]
            locked_indices[index] = 1
            if(np.sum(locked_indices) != n):
                pass_iterator = 1
            else: # All variables have been locked
                epoch_iterator = 0
                pass_iterator = 0
        else:# Found a solution that is greater than the current_max
            x0 = temporary[1]
            epoch_iterator = 0
            pass_iterator = 1


print(f"------------------- Algorithm 2 ------------------- \n")
print(f"The total weight of the items in the knapsack is: {np.dot(x0,weights)}.\n")
print(f"The total profit is: {np.dot(x0,profits)}. \n")
print(f"The items in the knapsack are: {get_items_in_knapsack(x0,profits_and_weights)}. \n"
print(f'The solution x is: {x0}')
print(f"----------------------- END ----------------------- \n")
```

```
------------------- Algorithm 2 -------------------

The total weight of the items in the knapsack is: 29.0.

The total profit is: 63.0.

The items in the knapsack are: [(6, 3), (8, 3), (7, 5), (6, 5), (11, 3), (1
2, 6), (13, 4)].

The solution x is: [0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0.
0. 0.]
----------------------- END -----------------------
```