ADVANCED ANALYTICS
ASSIGNMENT- 4

SAI TEJA KANNEGANTI
Id.No: 113384040

S.No	Heuristic Technique	Solutions Checked	Optimum Value
1	Local search with Best Improvement	3300	3454.9
2	Local search with First Improvement	953	2791.5
3	Local search with Random Restarts	319,100	3517.9
4	Local search with Random walk	3206	3514.5
5	Simulated Annealing (Cauchy cooling)	299	3681.9
6	Tabu Search		3687.2
7	Variable Neighborhood search	800	3208.5

Question 1: Strategies for the problem (14 points)

(a) Define and explain a strategy for determining an initial solution to this knapsack problem for a neighborhood-based heuristic.

My initial solution is considering first 10 elements and not considering the other elements, because maximum weight of each element can be 20 and thus total maximum weight can be 200, which is less than maximum weight limit of 500. So that the initial solution falls in feasible region.

(b) Recommend 3 neighborhood structure definitions that you think would work well with the example knapsack problem in this assignment.

Neighborhood structures that would work well with knapsack problem are:
1-Flip neighborhood of solution, 2-Flip neighborhood of solution, 3-Flip neighborhood of solution

(c) What is the size of each of the neighborhoods you recommended?

1. 1-flip neighborhood has 100 Choose 1 (100)
2. 2-flip neighborhood has 100 Choose 2
3. 3-flip neighborhood has 100 Choose 3

(d) Identify 2 neighborhood structure definitions that you think would NOT work well with the example knapsack problem in this assignment. Explain why.

Two neighborhood structures that will not work with this knapsack problem are:

1. Swapping – swapping of 1 & 0 between the elements
2. Inserting – Inserting 1 or 0 for the elements.

(e) In the evaluation of a given solution, an infeasible may be discovered. In this case, provide 2 strategies for handling infeasibility.

Two strategies for handling infeasibility are:

1. `Keeping return [-1,-1] in evaluate function`

```
def evaluate(x):
    a = np.array(x)
b = np.array(value)
c = np.array(weights)

totalValue = np.dot(a, b)
totalWeight = np.dot(a, c)

if totalWeight > maxWeight:
    return [-1,-1]
return [totalValue, totalWeight]
```

2. Limiting the Total weight to be less than maximum weight in the while loop

```
while done == 0:

    Neighborhood = neighborhood(x_curr)
    for s in Neighborhood:
        solutionsChecked = solutionsChecked + 1
        if evaluate(s)[0] > f_best[0] and evaluate(s)[1] <= maxWeight:
            x_best = s[:]
            f_best = evaluate(s)[:]
```

Question 2: Local Search with Best Improvement (10 points)

Complete the original Python Local Search code provided to implement Hill Climbing with Best Improvement. Note you will need to implement your strategy for determining an initial solution, handling infeasibility, and possible your neighborhood structures.
Apply the technique to the random problem instance and determine the best solution and objective value using your revised algorithm.

Solution:

Strategy for determining an initial solution:

- My initial solution is considering first 10 elements and not considering the other elements, because maximum weight of each element can be 20 and thus total maximum weight can be 200, which is less than maximum weight limit of 500. So that the initial solution falls in feasible region.

Handling infeasibility:

- We can handle infeasibility by checking the solution weight falls within the maximum weight limit. If the weight obtained is greater than maximum weight (infeasible) return[-1,-1] statement gets executed and it comes out of loop.

Neighborhood structure used:

- Neighborhood structure used is 1-flip neighborhood.

Output:

Final number of solutions checked: 3300

Best value found: 3454.93137899

Weight is: 499.381384307

Total number of items selected: 42

Local search with Best Improvement solution: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Question 3: Local Search with First Improvement (5 points)

Modify the completed Python Local Search code to implement Hill Climbing with First Improvement. Apply the technique to the random problem instance and determine the best solution and objective value using your revised algorithm.

Solution:

Strategy for determining an initial solution:

- Initial solution is considering first 10 elements and not considering the other elements, because maximum weight of each element can be 20 and thus total maximum weight can be 200, which is less than maximum weight limit of 500. So that the initial solution falls in feasible region.

Handling infeasibility:

- We can handle infeasibility by checking the solution weight falls within the maximum weight limit. If the weight obtained is greater than maximum weight (infeasible) return[-1,-1] statement gets executed and it comes out of loop.

Neighborhood structure used:

- Neighborhood structure used is 1-flip neighborhood.

Output:

Final number of solutions checked: 953

Best value found: 2791.52170883

Weight is: 498.667885204

Total number of items selected: 42

Local search with first improvement solution is : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Question 4: Local Search with Random Restarts (8 points)

Modify the completed Python Local Search code to implement Hill Climbing with Random Restarts. You may use Best Improvement or First Improvement (just clearly state your choice). Make sure to include the following:
. Make the number of random restarts an easily modifiable variable.
. Keep track of the best solution found across all of the restarts.
Apply the technique to the random problem instance and determine the best solution and objective value using your revised algorithm.

Solution:

- Choice: Used Best Improvement to implement Random Restart

Make the number of random restarts an easily modifiable variable:

- Number of restarts provided in code is 10 (using variable 'no_restarts', which can be easily modifiable). Tracks the best solution found across all the restarts and obtains the solution.

Strategy for determining an initial solution:

- Initial solution is a list having all zeros. That is no element is selected.

Handling infeasibility:

- We can handle infeasibility by checking the solution weight falls within the maximum weight limit. If the weight obtained is greater than maximum weight (infeasible) return[-1,-1] statement gets executed and it comes out of loop.

Neighborhood structure used:

- Neighborhood structure used is 1-flip neighborhood.

Output:

Total number of solutions checked: 319100

Best value found: 3517.857116053257

Weight is: 499.37158592778519

Total number of items selected: 40

Random Restart with Best improvement solution is : [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0]

Question 5: Local Search with Random Walk (8 points)

Modify the completed Python Local Search code to implement Hill Climbing with Random Walk. You may use Best Improvement or First Improvement (just clearly state your choice). Make sure to include the following:
. Make the probability of random walk an easily modifiable variable.
Apply the technique to the random problem instance and determine the best solution and objective value using your revised algorithm.

Solution:

- Choice: Used Best Improvement to implement Random Walk

Make the probability of random walk an easily modifiable variable:

- Probability of random walk in code is 0.75 (using variable 'prob', which can be easily modifiable).

Stopping Criteria:

- Stopping criteria is met when solutions checked reaches 40,000. Then it returns the best value found so far.

Strategy for determining an initial solution:

- Initial solution is considering first 10 elements and not considering the other elements, because maximum weight of each element can be 20 and thus total maximum weight can be 200, which is less than maximum weight limit of 500. So that the initial solution falls in feasible region.

Handling infeasibility:

- We can handle infeasibility by checking the solution weight falls within the maximum weight limit. If the weight obtained is greater than maximum weight (infeasible) return[-1,-1] statement gets executed and it comes out of loop.

Neighborhood structure used:

- Neighborhood structure used is 1-flip neighborhood.

Output: (probability =0.75)

Final number of solutions checked: 3206

Best value found: 3514.51189194

Weight is: 495.892783447

Total number of items selected: 43

Random Walk with Best Improvement solution: [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Output: (probability =0.80)

Final number of solutions checked: 3206

Best value found: 3514.51189194

Weight is: 495.892783447

Total number of items selected: 43

Random Walk with Best improvement solution is: [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Output: (probability =0.90)

Final number of solutions checked: 3302

Best value found: 3454.93137899

Weight is: 499.381384307

Total number of items selected: 42

Random Walk with Best improvement solution is: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Question 6: Simulated Annealing (20 points)

Using the completed Python code as a base, implement Simulated Annealing. Make sure to include the following:
. Explanation of how you determined the initial temperature.
. Well-defined the temperature schedule (the temperature update procedure, the number of iterations performed at a given temperature, etc.)
. Explanation of the stopping criterion.
Apply the technique to the random problem instance and determine the best solution and objective value using your revised algorithm.

Solution:

Explanation of how you determined the initial temperature:

- The initial temperature is found by using the formula, $p = e^{-\frac{\Delta max}{T}}$
- In the above equation probability is kept high i.e., p= 0.9 and Δ_{max} is obtained using difference in the values of evaluation function. Initial temperature (T) is found by the above formula.

Well-defined the temperature schedule:

- **Cauchy Cooling Schedule:**

$$T_{k+1} = \frac{T_k}{1 + K}$$

```
def getCurrentTemperature(temperature_maximum, n):
    temperature_maximum = temperature_maximum /(1 + n) # Cauchy cooling schedule
    return temperature_maximum, n * 40
```

- **Linear Cooling Schedule:**

```
def getCurrentTemperature(temperature_maximum, n):
    temperature_maximum = temperature_maximum-(n*2) #linear cooling schedule
    return temperature_maximum, n * 40
```

Explanation of the stopping criterion:

- Stopping criteria used is number of iterations to be completed, which is fixed at 300.
- I have also used stopping criteria as final temperature, process stops when the final temperature has arrived. But I have found better solution considering number of iterations as stopping criteria.

Output: Cauchy Cooling Schedule

number of solutions checked: 299

Final Temperature: 12.666666666666666

Best value found: 3681.86565313

Weight is: 498.54269249282129

Item selected: 47

Simulated Annealing Cauchy Cooling solution: [1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Output: Linear Cooling Schedule

number of solutions checked: 299

Final Temperature 3202

Best value found 3130.27897273

Total weight: 498.77805484886881

Items selected 45

Simulated Annealing Linear Cooling solution: [0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0]

Question 7: Tabu Search or Variable Neighborhood Search (30)

Using the completed Python code as a base, implement either a Tabu Search or Basic VNS to solve the knapsack problem.

Tabu Search:

For TS, make sure to include the following:
. Explain the tabu criterion, tabu tenure, aspiration criterion, etc. and any other parameters you include.
. Long-term memory is optional.

Solution:

- Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available (like when the search is stuck at a strict local optimum). In addition, prohibitions (hence the term tabu) are introduced to discourage the search from coming back to previously-visited solutions.
- Tabu criteria is met with the help of tabu tenure.

Solution parameters:

- Maximum number of iterations: 1200
- Candidates chosen from neighborhood: 60
- Tabu tenure: 5
- Aspiration Criterion: evaluate(current) > evaluate(best)
- Tabu Criterion: Decrement tabu life, if the objective value appears again

Output:

Best value found: 3687.24894681

Total weight: 499.172651497

Items selected: 44

TABU solution: [0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0]

Variable Neighborhood Search (Extra Credit)

For Basic VNS, make sure to include the following:
. You must define and use at least 3 different neighborhood structures.
. Define the local search component.

Solution:

You must define and use at least 3 different neighborhood structures:

- Variable Neighborhood Search is implemented using 3 different neighborhood structures. Three neighborhood structures used are 1-Flip neighborhood of solution, 2-Flip neighborhood of solution, 3-Flip neighborhood of solution

```
def Variable_Neighbor(x, k): # k refers to number of flips
  neighboorhood = []
  no_flips = []
  for i in range(0,n):
    neighboorhood.append(x[:])
    no_flips = [myPRNG.randint(0, n - 1) for u in range(0, k)]
    for j in no_flips:
      if neighboorhood[i][j] == 1:
        neighboorhood[i][j] = 0
      else:
        neighboorhood[i][j] = 1
  return neighboorhood
```

Output:

Final number of solutions checked: 800

Best value found: 3208.47493148

Weight is: 499.875289076

Total number of items selected: 42

Best solution: [0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0]

Question 8: Summary (5 points)

What are your thoughts regarding the performance of the neighborhood-based heuristics that you implemented? Which technique would you recommend for this problem? Did you try any variations (e.g. allowing infeasible moves, changing the initial solution strategy, different cooling processes, different stopping criteria, etc.?) If so, what seemed to be most effective?

Solution:

- Local search with first improvement gives quicker solution than Local search with best improvement.
- For this problem, Tabu search, Simulated Annealing gives better solution.
- For Simulated Annealing I tried two cooling techniques, they are Cauchy cooling technique, Linear cooling technique. I observed that Cauchy cooling technique gives better optimum solution when compared to Linear cooling technique.
    1. Cauchy Cooling process gives optimum value of 3681.9
    2. Linear Cooling process gives optimum value of 3130.3
- Local search with Random restarts, Random walk performed better with Best improvement than with first improvement.
- Stopping criteria used is number of iterations to be completed, which is fixed at 300. I have also used stopping criteria as final temperature, process stops when the final temperature has arrived. But I have found better solution considering number of iterations as stopping criteria.
    1. Stopping criteria with 300 iterations gives optimum of 3681.9
    2. Stopping criteria with final temperature as 15 gives optimum of 3568.8
- In Simulated Annealing initial solution is given by:

```
def initial_solution():
    for i in range(0, n):
        if myPRNG.random() < 0.9: # if random number generated between 0 & 1 is < 0.9 then item is not selected.
            x_curr.append(0)
        else:
            x_curr.append(1)
    return x_curr
```

1. myPRNG.random() < 0.9 gives optimum value of 3681.9
2. myPRNG.random() < 0.99 gives optimum value of 3561.1
3. myPRNG.random() < 0.8 gives optimum value of 3665.4

- Local search with random Rewalk:

Probability	Objective value
0.75	3514.5
0.80	3514.5
0.90	3454.9