Lince Rumainum & Ramkishore Rao
DSA 5113
Advanced Analytics & Metaheuristics
Group 9 - HW 4

Problem 1 – Strategies for the problem

Part a

For the initial solutions, we create a list of random picks for the solutions. At first it will randomly select the item but once the total weight is greater than the specified (random) upper limit, all the other items will not be added. **Note:** The combination of those items in the knapsack will always be less than a (random) percentage of the maximum weight. Below is the code of the initial solution function:

```
reate the initial solution
f initial_solution(restartsCounter):
  x = []   #i recommend creating the solution as a list

  #need logic here!
  #create temporary lists for weights and solutions(var x)
  tempWeights = []
  tempX = []

  for i in range(0,n):
      #creating new seed so solution will be picked at random
      solutionSeed = myPRNG.randint(0,(restartsCounter+1)*10)
      solutionSeed *= restartsCounter*100
      tempSolution = Random(solutionSeed)
      #add weights
      tempWeights.append(weights[i])
      #random selections
      tempX.append(tempSolution.randint(0,1))

      #put temporary lists into array
      a=np.array(tempX)
      c=np.array(tempWeights)

      totalWeight = np.dot(a,c)    #compute the weight value of the knapsack selection

      #create random percentage to use as constraint of weight upper limit
      percentageOfMaxWeight = i*5/100

      #make sure percentage at most 100%
      while percentageOfMaxWeight > 1:
          percentageOfMaxWeight = percentageOfMaxWeight/3

      #append the selection if total weight is less than x% of total weight
      if totalWeight < maxWeight*percentageOfMaxWeight:
          x.append(tempX[i])
      else:
          x.append(0)
  return x
```

Part b

Three neighborhood structures that can be used for this problem:

- The 1-flip method, where it generated from the set of solution where it will be modified one index from start to end selections from being selected to unselect it or vice versa. Size of neighborhood: 150.
- Flip between two to four (closest) indices. For example, index 0 will flip index 1 and index 2, index 2 would flip two indices before (index 0 and index 1) and two indices after (index 3 and

index 4), where those indices will be modified from being selected to unselect it or vice versa. This would have more variations than only 1-flip method. Size of neighborhood: 150.

- Flip an odd and an even index from start to end (i.e. flip index 0 and index 1, flip index 2 and index 3, and so on), where those indices will be modified from being selected to unselect it or vice versa. This will have smaller size neighborhood but more variation than the 1-flip method. Size of neighborhood: 75.

Part c

Since the goal of this knapsack problem is to maximize the total value without exceeding the weight limitation, two strategies to handle infeasible solution (weight exceed limitation):

- A simple solution would be setting the total value of the solution to zero so it will never be chosen as the best solution.
- We can sort the list by value (lowest to highest) and start to unpick the one(s) with the lowest value until the weight is finally less than or equal to the maximum weight.

Problem 2 – Problem 7

Algorithm	Iterations	# Items Selected	Weight	Objective
Local Search (Best Improvement)	1,500	20	2,499.80	23,006.20
Local Search (First Improvement)	214	19	2,494.90	13,742.80
Local Search (Best Improvement - Random Restarts: 100)	174,300	18	2,489.90	29,345.10
Local Search (Best Improvement - Random Restarts: 1,000)	1,737,000	19	2,499.00	29,957.90
Local Search (Best Improvement - Random Walk, p = 0.4)	1,518	20	2,499.80	23,006.20
Local Search (Best Improvement - Random Walk, p = 0.55)	1,836	20	2,493.90	24,369.20
Local Beam Search (10 parallel searches)	33,000	22	2,497.60	32,698.90
Local Beam Search (20 parallel searches)	122,250	30	2,494.10	37,799.00
Stochastic Beam Search (10 parallel searches)	46,050	23	2,497.30	33,734.30
Stochastic Hill Climbing	1,500	20	2,489.50	22,626.90

To determine the stochastic beam search probability, use the equation below:

$$p = 90\% \frac{current\ total\ value}{max\ total\ value} + 10\% \frac{current\ total\ weight}{max\ total\ weight}$$

Since the goal is to maximize the total value, setting the 90% of the probability depend on the total value and 10% of the probability depend on the total weight will make the probability to rely on the total value more than the total weight. **Note**: The maximum total value is actually an approximation. I set it as the sum of all of the items with value greater than 1,400.

Code files:

Group9_HW4_p2.py, Group9_HW4_p3.py, Group9_HW4_p4.py, Group9_HW4_p5.py, Group9_HW4_p6.py, Group9_HW4_p7a.py and Group9_HW4_p7b.py.