ASSIGNMENT 4

ADVANCE ANALYTICS & METAHEURISTICS

SUBMITTED BY

SAI SAKETH BOYANAPALLI
SAI SRIKANTH KOLA

Question 1: Strategies for the problem (14 points)

a) (2 points) Define and explain a strategy for determining an initial solution to this knapsack problem for a neighborhood-based heuristic.

- We have used different initial solutions based on the problem structure for the Best improvement and First accept we have taken initial solution as $x = [0] * 100$ that is not selecting any items for the first iteration. This will enable neighborhood based heuristic to search through all the available solutions.
- And for the Random walk and Random Restart we have set up initial solution to zero for the first iteration and later changed it to set 10 values to 1 i.e., taking 10 items at the start of problem. This will enable our problem to have different instance at the start of the restart.

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

1. **1 Flip neighborhood:** In this neighborhood 1 element is flipped at a time for every instance. This will enable us to have randomness in our problem.
2. **2 Flip neighborhood:** This is very similar to 1 flip neighborhood but here instead of flipping just 1 element we flip 2 elements for every instance of the problem.
3. **First half flip:** Here the first $\frac{n}{2}$ elements will be intact and the latter part $\frac{n}{2} - n$ we undergo 1 flip.

c) (3 points) What is the size of each of the neighborhoods you recommended?

- The size of each of the neighborhood is set to n.

d) (4 points) Identify 2 neighborhood structure definitions that you think would NOT work well with the example knapsack problem in this assignment. Explain why.

1. **Add neighborhood:** In this neighborhood, we add a no to the elements. In this case our solution is in binary so adding elements will change this so we cannot use this for a knapsack problem.
2. **Subtract neighborhood:** This is similar to the above-mentioned neighborhood. But here subtracting elements we make our solutions take negative values making it in feasible.

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

1. We are setting such that if the total weight is greater than maximum weight, than total value is set to [-1,-1] this will make sure that instance of solution is discarded.
2. Other case is to set the total value to 0, if the total weight is greater than the maximum weight. This will make sure that instance of solution is discarded.

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

Ans:

The Local search with Best Improvement identifies the Best-found improvement and Stores it as current value. The python code provided is for best improvement in which the number of instances were 100, values randomly generated from range 10 to 100 and weight is generated randomly from 5 to 15. The maximum weight is a set of 5 times the original weight.

The pseudo code to get the best improvement is:

Pseudo code:

```
for each s ∈N(current) do
   if f (s) < f (best neighbor) then
     best  neighbor ⇐ s
  end if
end for
```

Strategy to determine an initial solution:

we create an empty list which is named as 'x' and we iterate for 'i' times from 0 to 100 range using a For loop in which for each iteration 'x' takes a value of 0 into its list. The python code for the initial solution is:

Python code:

```
def initial_solution():
   x = []
   for i in range(0,100):
      x.append(0)
      print(x)
      return x
```

Here append functions helps to keep 0's in to the list for every iteration.

Strategy for handling infeasibility:

When the infeasibility condition is met which in this case is if total weight of elements selected exceeds 500 then we return -1 value using if condition to exit the present process and to take new values and python code for this is:

Python code:

```
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]

else:
    return [totalvalue, totalweight]
```

Strategy for neighborhood structure:

Here we use one flip neighborhood where after selecting the random solution for 'x' this function will change the solution in such a manner that it will convert 1's into 0's and if it has 0's then it will change to 1.

Python code:

```
def neighborhood(x):
    nbrhood = []
    for i in range(0, n):
        nbrhood.append(x[:])
        if nbrhood[i][i] == 1:
            nbrhood[i][i] = 0
        else:
            nbrhood[i][i] = 1
    return nbrhood
```

Now on running this python code for Local search for Best Improvement we get results as:

Results:

Final number of solutions checked: 4000

Best value found: 3465.46182268

Weight is: 495.272698677

Total number of items selected: 39

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

Ans:

The Local search with First Improvement identifies the First found improvement and Stores it as current value. The python code provided is for best improvement in which the number of instances were 100, values randomly generated from range 10 to 100 and weight is generated randomly from 5 to 15. The maximum weight is a set of 5 times the original weight.

As the python code given is for best improvement we should change it to first improvement with pseudo code.

Pseudo code:

```
for each s ∈N(current) do
   if f (s) < f (best neighbor) then
     best  neighbor ⇐ s
       exit the for-loop
  end if
end for
```

Running this case on Python, we get different results, because the for-loop exits when it spots the first improvement to the current value.

Results:

Final number of solutions checked: 1008
Best value found: 2791.52170883
Weight is: 498.667885204
Total number of items selected: 42

Now comparing those results with best improvement

Solution parameters	Local Search with Best Improvement	Local Search with First Improvement
Total no. of Solutions Checked	4000	1008
Best Objective value	3465.46182268	2791.52170883

Table 01 Solution Parameters for Local Search Algorithms

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

Ans:

The code provide is for Local search for Best Improvement so I tuned the same code to get Local search for random restarts. So, the code that used was from ***Local Search for Best Improvement.*** The pseudo code for Random restarts is:

Pseudo code:

```
for restart with restart _ num
  for each s ∈N(current) do
    if f (s) < f (best _ neighbor) then
      best _ neighbor ⇐ s
    end if
  end for
end for
```

This pseudo code explains the how the for loop works where in the code the variable '*restart _ num'* is set right before the for loop starts and it is an easily changeable variable which can be set to any number depending up on how many times a loop has to restart.

Results:

Now for different number of restarts the best values are given in the below table.

Restart Number	**Solutions Checked**	**Best Value Found**
50	200000	3465.40
80	320000	3465.46
100	400000	3465.4618

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

Ans:

The code provide is for Local search for Best Improvement so I tuned the same code to get Local search for random restarts. So, the code that used was from ***Local Search for Best Improvement.*** The Random Walk is Implemented using the rand_int variable. If probability is less than 0.75

then best search is implemented else random walk is implemented. Python code used is as follows:

Python code:

```
done = 0
location=0
rand_num = 0.75
solutions = []

while done == 0:
    solutions.append(f_best)
    Neighborhood = neighborhood(x_curr)  # create a list of all neighbors in the
neighborhood of x_curr
    step= myPRNG.uniform(0,1) #probabilty = 0.5 to select 0 and probability = 0.5
to select 1
    if step < rand_num:
        for s in Neighborhood:  # evaluate every member in the neighborhood of
x_curr
            solutionsChecked = solutionsChecked + 1
            if evaluate(s)[0] > f_best[0]:
                x_best = s[:]  # find the best member and keep track of that
solution
                f_best = evaluate(s)[:]  # and store its evaluation
    else:
        walk = 0
        while walk == 0:
            solutionsChecked = solutionsChecked + 1
            rand_index = myPRNG.randint(0,98)
            s = Neighborhood[rand_index]
            if evaluate(s)[0] >= 0:
                x_best = s[:]  # find the best member and keep track of that
solution
                f_best = evaluate(s)[:]  # and store its evaluation
                walk = 1

    if f_best == f_curr:  # if there were no improving solutions in the
neighborhood
            done = 1
    else:
        x_curr = x_best[:]  # else: move to the neighbor solution and continue
        f_curr = f_best[:]  # evaluate the current solution
```

Here you can see in the code that variable ‘rand_num’ is the variable for which different probabilities can be assigned.

Results:

Now when the probability value is 0.75 then the results are:

Final number of solutions checked: 3706
Best value found: 3496.6377855
Weight is: 499.318267198
Total number of items selected: 40

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

Extra credit:

Now for different probabilities the best value found is given in below table

Random walk probabilities	Solutions Checked	Best value found
0.5	2638	3264.90511977
0.65	3016	3372.49067557
0.75	3706	3496.6377855

Question 6: Simulated Annealing (20 points):

Ans:

The instance of the problem used is best Improvement.

- The initial temperature can be found by using $p = e^{(f(s_2)-f(s_1))/T)}$ here we need to find the value of T initially we take $p = 0.9$ than after evaluating we get $T = 3800$ which is our initial temperature setting.
- The temperature is scheduled for 4 different temperature values.

Update Procedure

1. Cauchy cooling schedule $Cauchy = \left(\frac{T}{1+n}\right)$
2. Boltzmann cooling schedule $B = \left(\frac{T}{math.\log(1+n)}\right)$
3. Very fast temperature cooling schedule $FT = T \times math.\exp(-cn^q)\ where\ c = 2,\ q = 1$
4. Geometric cooling schedule $T_{k+1} = \alpha Tk\ for\ \alpha\ value\ 0.8\ linear = T \times alpha, alpha = 0.95$

No of iterations

- This is set to $n \times 40 = 4000$. For the Cauchy and blotzmann and it is set to 100 for the other 2.

Stopping criteria

- And the stopping criteria is 300. Because if we increase this value than the time taken to run increases significantly and if we decrease than we may not get the best solution.

Cauchy cooling schedule	
Best value found	**3532.91**

Final Temperature	**10**
Best weight	**498.16**
No of items selected	**46**

Boltzmann cooling schedule	
Best value found	**3179.96**
Final Temperature	**525.96**
Best weight	**498.311**
No of items selected	**45**

Very fast temperature cooling schedule	
Best value found	**2508.07**
Final Temperature	**5.87**
Best weight	**497.75**
No of items selected	**41**

Geometric cooling schedule	
Best value found	**3039.77**
Final Temperature	**2400**
Best weight	**497.65**
No of items selected	**43**

From the above tables, we can see that **Cauchy** cooling schedule is providing us with the **best** value among the other schedules followed by Boltzmann, Geometric and finally very fast temperature cooling schedule.

With the best value $\mathbf{3532.91}$ $\mathit{selecting}$ $\mathbf{46}$ items.

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

Ans:

Variable Neighborhood Search:

This method uses Tabu Search itself but it iteratively searches for the best solution with help of Tabu search and updates its neighborhoods randomly where it is selected from a pool of choices. Here the strategy is to search for larger neighborhoods once the solution had reached a local maxima and is stuck there. According to Hansen *et al.* , the VNS is led by the 3 principle that (a) a local minimum for one neighborhood may not be a local minimum for another and (b) a global

minimum is a local minimum for all possible neighborhood structures, and (c) local minimum is relatively close to global minimum is most cases. The pseudo code for VNS is given below replicated from [3]:

Pseudo code:

Input: Neighborhoods
Output: *Sbest*
while (stop condition is reached)
  for (Neighborhood*i* ∈ Neighborhoods)
    Neighborhood*curr* ← CalculateNeighborhood(*Sbest*, Neighborhood*i*)
    Scandidate ← RandomSolutionInNeighborhood(Neighborhood*curr*)
    if (*Scandidate* > *Sbest*)
      Sbest ← *Scandidate*
      Break

    End

  End
return(*Sbest*)

The three Neighborhoods:

The first neighborhood is called full flip. The code snippet is shown below.

```
def neighborhood(x):
    nbrhood = []
    for i in range(0, n):
        nbrhood.append(x[:])
        if nbrhood[i][i] == 1:
            nbrhood[i][i] = 0
        else:
            nbrhood[i][i] = 1
    return nbrhood
```

The second neighborhood is called 2-flip neighborhood. The code snippet is shown below.

```
def neighborhood2(x):
    nbrhood = []
    temp_nhbr = []
    for i, j in itr.combinations(range(0, 100), 2):
        temp_nhbr = x[:]
        temp_nhbr[i] = 1
        temp_nhbr[j] = 1
```

```
            nbrhood.append(temp_nhbr)
        return nbrhood
```

The second neighborhood is called 3-flip neighborhood. The code snippet is shown below.

```
    def neighborhood3(x):
        nbrhood = list()
            for i, j, k in itr.combinations(range(0, 100), 3):
                tempx = x[:]
                tempx[i] = 1 if tempx[i] == 0 else 0
                tempx[j] = 1 if tempx[j] == 0 else 0
                tempx[k] = 1 if tempx[k] == 0 else 0
    nbrhood.append(tempx)
return nbrhood
```

Prior to performing tabu search, the pointer randomly selects one of the neighborhoods and proceeds with finding the best candidate, moving along as it does so.

Results:

Final number of solutions checked: 2000
Best value found: 3727.56748
Weight is: 499.4637
Total number of items selected: 47

Extra credit:

When we try some other local search neighborhoods like

The three Neighborhoods:

The first neighborhood is called full flip. The code snippet is shown below.

```
def neighborhood(x):
    nbrhood = []
    for i in xrange(0,n):
        nbrhood.append(x[:])
            if nbrhood[i][i] == 1:
                nbrhood[i][i] = 0
            else:
                nbrhood[i][i] = 1
return nbrhood
```

The idea is simple, easy to use, and retains maximum variability a random process can provide. Good to explore
spaces. The second neighborhood is called First Half Flip. The code snippet is given below:

```
def NeighFirstHalf(x):
    nbrhood = []
    nbrhood.append(x[:])
```

```
    for i in xrange(0,n):
        nbrhood.append(x[:])
        for i in xrange(0,n/2):
            if nbrhood[i][i] == 1:
                nbrhood[i][i] = 0
            else:
                nbrhood[i][i] = 1
return nbrhood
```

The next neighborhood is called Second Half Flip. The code snippet is provided below:

```
def NeighFirstHalf(x):
    nbrhood = []
    nbrhood.append(x[:])
    for i in xrange(0,n):
        nbrhood.append(x[:])
        for i in xrange(n/2,n):
            if nbrhood[i][i] == 1:
                nbrhood[i][i] = 0
            else:
                nbrhood[i][i] = 1
return nbrhood
```

Results:

Final number of solutions checked: 4228
Best value found: 3524.13704862
Weight is: 499.431233841
Total number of items selected: 40
Best solution: [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 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, 1, 0, 0, 0]

Tabu Search:

It can be easily seen that for small solution spaces, the search gets limited to a particular local maxima, and repeatedly goes to that value wasting CPU time. Tabu search was included, so that a particular space would be deemed taboo for certain number of iterations. The extra-credit problem solved here is diversification of the Tabu Search by penalizing maximum frequency of taboo occurrence.

Python code:

```
while (not stoppingCondition())
    candidateList ← []
    best ← null
    for (s in sNeighborhood)
```

```
            if ((not tabuList.contains(s)) and (evaluate(s) > evaluate(best)))
                best ← s
            end
        end
        s ← best
        if (evaluate(best) > evaluate(sBest))
            sBest ← bestCandidate
        end
        tabuList.push(best);
        if (tabuList.size > maxTabuSize)
            tabuList.removeFirst()
        end
    end
    return sBest
```

we gave the number of instances for the Knapsack problem is 100 and all the rest of the things like randomization of values etc., remains same as previous problems and even the neighborhoods are formed by flipping the solution values and the results are:

Results:

Best value found: 3610.4620526
Total weight: 499.790727911
Best solution: [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 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, 1, 0, 0, 0]

Solution parameters	Local Search with Best improvement
Maximum number of iterations	1000
Candidates chosen from neighborhood	42
Tabu tenure	5
Aspiration Criterion	evaluate(current) > evaluate(best)
Tabu Criterion	Decrement tabu life, if the objective value appears again

Tabu Search with Path Relinking:

Path relinking is done to link the current solution to the overall best solution by setting up a path and repair procedure through the solution space. Because the knapsack solution contains primarily of 0s and 1s, the best repair procedure to correctively guide the current solutions would be to flip the solution elements. It works the following way:

flip(a, i)

if $i = 0$:
    flip i to 1
else:
    $i = 0$

The pseudo code for path relinking procedure is a bit long, but the core procedure is discussed below:

if the $dist(a, b)$ > alpha:
    get the current and best value from TS
    repair the solution and make it best
    update the distance parameter

This pseudo code is kept in the Tabu search and the iteration parameter is changed to 60 and the results are obtained.

Results:

Best value found: 3687
Total weight: 499.657
Best solution: [1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 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?

- Neighborhood based heuristics are fast and give a result that's very close to the optimal solution for the instance of the problem. But the algorithm works best if only modeled properly or else we might not get the desired output. And if there is an error in the model than its very hard to detect it.
- For this problem, best technique that we would recommend is **VNS** because vns took less iterations than any other technique and gave us the best optimal solution when compared to other techniques.

$$VNS - 3727, solutions\ checked - 2000$$
$$Best\ improvement\ - 3465, solutions\ checked - 4000$$
$$Best\ accept - 2791, solutions\ checked - 1008$$
$$Tabu\ - 3610\ solutions\ checked - 1000$$

$$Simulated\ annealing - 3532\ solutions\ checked - 299$$

- I tried changing the initial solution to other value i.e., less than the maximum weight, what I had noticed was a change in no of iterations. That's because changing it from 0 I was submitting a solution with an initial value other than 0 this is allowing the algorithm to improve the provided solution thus taking less time to come up with the solution.

$$best\ improvement\ initial\ solution\ set\ to\ 0\ value - 3465\ solutions\ checked - 4000$$

$$initial\ solution\ set\ to\ generate\ random\ values\ of\ 0\ \&\ 1\ than\ value\ is\ 3114\ and\ solutions\ checked - 1400$$

- Tried giving infeasibility, in the initial solution but in this case the solution had [-1] in the final output with higher total value and iterations but it is not feasible to have [-1] in the solution.
- Given different cooling process Cauchy yield a better value among the other cooling schedules.
- Among the above-mentioned variations initial solution played a major part in reducing the time and arriving at optimal solution cause, providing an optimal solution as an input will only make the algorithm to stop. So closer the initial solution to optimal lesser the no of iterations to arrive at optimal.