Solving CVRP with Simulated Anealing (SA)

Importing Packages

In [3]:

```
import numpy as np
import pandas as pd
import time
```

SA Class to Perform Simulated Anealing Algorithm

In [6]:

```
#This class, consists of every operation and attribute related to SA algorithm
class SA:
    #cooling rate
    alpha = 0.99
    #initial tempreture
    t_init = 5000
    #number of iterations per tempreture
    n_iter = 5
    #capacity of vehicle
    capacity = 0
    #number of vehicles
    n_vehicle = 0
    #probability of performing move operation to find neighbor
    q_method = 0.2
    #limit of tempreture
    limit = 10
    #best overall solution
    best_s = np.array([])
    #best overall cost
    best_cost = 0
    #number of candidates customers in finding neighbors
    neighbor_candidates = 5
    #Array of Answer
    pop = np.array([])
    #List of costs in every iteration
    energy_list = np.array([])

    # Reading data and initilizing variables
    def __init__(self, directory, start_line, end_line, end_demand_line, end_file, capacity, n_vehicle):
        '''
            directory: directory of file
            start_line: number of line to start reading location of customers
            end_line: number of line to finith reading location of customers
            end_demand_line: number of line to finith reading demand of customers
            end_file: line of EOF
            capacity: capacity of each vehicle
            n_vehicle: number of vehicles
        '''
        self.data = pd.read_csv(directory, sep = ' ', skipinitialspace = True, names = ['CUST', 'X', 'Y'],\
                                index_col = 'CUST',  skiprows = lambda x: x in range(start_line) or\
                                x in range(end_line, end_file))
        demand = pd.read_csv(directory, sep = ' ', skipinitialspace = True, names = ['CUST', 'D'],\
                             index_col = 'CUST',  skiprows = lambda x: x in range(end_line + 1) or\
                             x in range(end_demand_line, end_file))
        self.data = self.data.merge(right = demand, on = 'CUST')
        self.best_cost = self.energy(self.best_s)
        self.n_vehicle = n_vehicle
        self.capacity = capacity

    #Init answer array
    def init_pop(self):
        checked = [0 for i in range(self.data.shape[0])]
        first_row = self.data.loc[1].to_frame().T
        self.data = self.data.loc[np.random.permutation(np.arange(2, self.data.shape[0]+1))]
        self.data = pd.concat([first_row, self.data], ignore_index = False)
        for j in range(self.n_vehicle):
            c = 0
            for i in range(self.data.shape[0]):
                if ((self.data.iloc[i]['D'] + c) <= self.capacity) and (checked[i] == 0 or i == 0):
                    c += self.data.iloc[i]['D']
                    self.pop = np.append(self.pop, int(self.data.iloc[i].name))
                    checked[i] = 1
        self.pop = np.append(self.pop, 1)
        self.data.index.names = ['CUST']

    #Check capacity constraint, returns True if constraints are observed
    def check_cap(self, array):
        '''
            array: array of answer, pass to function to be checked for capacity constraint
        '''
        c = 0
        for i in range(len(array)):
            if array[i] == 1:
                c = 0
            else:
                c += self.data.loc[array[i], 'D']
```

```
            if c > self.capacity:
                return False
        return True

    #Move operation to find neighbor and returns this neighbor
    def move(self, array):
        '''
            array: array of answer, pass to function to find neighbor around it
        '''
        dis = self.find_dis(array)
        dis.sort_values('dis', inplace = True)
        dis.reset_index(drop = True, inplace = True)
        best = dis.loc[:self.neighbor_candidates-1, 'CUST']
        random_cities = np.random.choice(np.arange(2, self.data.shape[0]), self.neighbor_candidates,\
                                         replace = False)
        while random_cities[np.isin(random_cities, best)].size != 0:
            random_cities = np.random.choice(np.arange(2, self.data.shape[0]), self.neighbor_candidates,\
                                             replace = False)
        i = 0
        while i < random_cities.shape[0]:
            new = array[array != random_cities[i]]
            idx = np.random.randint(1, len(array)-1, 1)
            new = np.insert(new, idx, random_cities[i])
            if self.check_cap(new):
                array = new
                i += 1
        return array

    #Highest operation to find neighbor and returns this neighbor
    def highest(self, array):
        '''
            array: array of answer, pass to function to find neighbor around it
        '''
        dis = self.find_dis(array)
        left = dis.iloc[:-1].reset_index(drop = True)
        right = dis.iloc[1:].reset_index(drop = True)
        left['m'] = (left['dis'] + right['dis']) / 2
        left = left[left['CUST'] != 1]
        left.sort_values('m', ascending = False, inplace = True)
        left.reset_index(drop = True, inplace = True)
        cities = left.loc[:self.neighbor_candidates-1, 'CUST']
        i = 0
        j = 0
        while i < self.neighbor_candidates:
            new = array[array != cities[i]]
            idx = np.random.randint(1, len(array)-1, 1)
            new = np.insert(new, idx, cities[i])
            if self.check_cap(new):
                if j == 0 or self.energy(new) < self.energy(array):
                    array = new
                j += 1
                if j > (self.neighbor_candidates - 1) :
                    j = 0
                    i += 1
        return array


    #Using move operation and highest operation to find neighbors and returns this neighbor
    def find_neighbor(self, arr):
        '''
            array: array of answer, pass to function to find neighbor around it
        '''
        if np.random.uniform(0, 1) <= self.q_method:
            arr = self.move(arr)
            return arr
        arr = self.highest(arr)
        return arr


    #Find distance every 2 consecutive cities in a rout and return this distance concated to destination
    def find_dis(self, array):
        '''
            array: array of answer, pass to function to the distances in consecutive cities
        '''
        positions1 = self.data.loc[array[:-1], ['X', 'Y']].reset_index()
        positions2 = self.data.loc[array[1:], ['X', 'Y']].reset_index()
        positions2['dis'] = np.sqrt((positions1['X'] - positions2['X']) ** 2 +\
                                    (positions1['Y'] - positions2['Y']) ** 2)
        return positions2

    #Cost function
    def energy(self, array):
        '''
            array: array of answer, pass to function to calculate its cost
        '''
        return np.sum(self.find_dis(array)['dis'])

    #Function to combine all steps and calculate the optimal answer along with its cost
    def play(self):
        temp = self.t_init
        self.init_pop()
        energy = self.energy(self.pop)
        self.best_cost = energy
        self.best_s = self.pop
        start = time.time()
        while temp > self.limit:
            for i in range(int(self.n_iter)):
                neighbor = self.find_neighbor(self.pop)
```

```
                if(self.energy(neighbor) < energy):
                    self.pop = neighbor
                    energy = self.energy(self.pop)
                else:
                    d_energy = energy - self.energy(neighbor)
                    if(np.random.uniform(0, 1) < np.exp(d_energy / temp) and (d_energy > (-0.5 * energy))):
                        self.pop = neighbor
                        energy = self.energy(self.pop)
                if energy<self.best_cost or self.best_cost == 0:
                    self.best_s = self.pop
                    self.best_cost = energy
            temp *= self.alpha
        print('Runtime: ', (time.time() - start)/60, 'min')
        print('Best answer: ', self.best_s)
        print('Best cost: ', self.best_cost)
```

E-n51-k5 Data Set

Test1

In [15]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n51-k5.vrp',\
        start_line = 7, end_line = 58, end_demand_line = 110, end_file = 115, capacity = 160, n_vehicle = 5)
sa.play()
```

```
Runtime:  8.859964815775554 min
Best answer:  [ 1. 28. 24.  8. 44. 25. 19. 26.  7.  1. 48. 38. 45. 16. 46. 18.  5. 14.
 15. 49.  1. 39. 50. 40. 11. 34. 43. 20. 42. 41. 13.  1. 33. 32. 29. 37.
 36. 21.  4. 22. 35.  6.  1. 47. 10. 51. 31. 17. 30.  3. 23. 27.  9.  2.
 12.  1.]
Best cost:  701.7119310893696
```

Test2

In [16]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n51-k5.vrp',\
        start_line = 7, end_line = 58, end_demand_line = 110, end_file = 115, capacity = 160, n_vehicle = 5)
sa.play()
```

```
Runtime:  8.758769631385803 min
Best answer:  [ 1. 33. 47.  6. 39.  3. 12. 11. 31. 50. 10.  1.  7. 26.  5. 18. 38. 40.
 46. 16. 43. 45. 48.  1. 28. 44.  8. 27. 32. 23. 29.  4. 37. 36. 21.  1.
 13. 19. 14. 42. 41. 20. 34.  1. 15. 25. 49. 24.  9.  2. 30. 22. 17. 35.
 51.  1.]
Best cost:  764.5595870622635
```

Test3

In [17]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n51-k5.vrp',\
        start_line = 7, end_line = 58, end_demand_line = 110, end_file = 115, capacity = 160, n_vehicle = 5)
sa.play()
```

```
Runtime:  8.42318510611852 min
Best answer:  [ 1. 47.  6. 50. 10. 40. 11. 16. 39.  3. 36.  1. 28.  2. 22. 30. 37.  4.
 32. 29. 17. 35. 51. 12.  1. 48. 18.  5. 19.  7. 49. 33. 21. 23.  1. 13.
 15.  8. 27.  9. 31. 34. 46. 38.  1. 45. 43. 20. 41. 42. 14. 26. 25. 44.
 24.  1.]
Best cost:  802.3626244805092
```

Test4

In [18]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n51-k5.vrp',\
        start_line = 7, end_line = 58, end_demand_line = 110, end_file = 115, capacity = 160, n_vehicle = 5)
sa.play()
```

```
Runtime:  8.575920073191325 min
Best answer:  [ 1. 14. 42. 20. 41. 43. 46. 11. 40. 50. 39. 17.  1. 19. 38. 34. 16. 45.
  5. 18.  7. 49. 28.  1. 26. 13.  6. 10. 31. 35. 22. 30. 51.  1. 47. 33.
  3. 12.  2. 24. 44. 25.  8.  9. 23.  1. 48. 15. 27. 32. 29.  4. 21. 37.
 36.  1.]
Best cost:  752.6709551167322
```

Test5

In [19]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n51-k5.vrp',\
        start_line = 7, end_line = 58, end_demand_line = 110, end_file = 115, capacity = 160, n_vehicle = 5)
sa.play()
```

```
Runtime:  8.6216139793396 min
Best answer:  [ 1. 19. 42.  5. 50.  3. 23. 12.  1.  7. 27. 44. 25. 15. 14. 20. 41. 43.
 13. 47.  1. 48. 38. 18. 10. 31. 35. 30. 22. 37. 36. 21.  1. 26. 24.  8.
 49. 28.  9. 32. 29.  4.  1.  6. 45. 16. 46. 34. 40. 11. 51. 39. 17. 33.
  2.  1.]
Best cost:  759.211177635011
```

E-n101-k8 Data Set

Test1

In [7]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n101-k8.vrp',\
        start_line = 7, end_line = 108, end_demand_line = 210, end_file = 215, capacity = 200, n_vehicle = 8)
sa.play()
```

```
Runtime:  11.732148325443267 min
Best answer:  [  1.  38.  94. 100.  62.  17.  87.  45.  86.  88.   3.   1.  53.  61.
  95.  41.  73.  75.  24.  76.  56.  26.  55.  13.  77.  78.   1.  96.
   7.  84.   6.  18.  85.  46.  83.  49.   9.  19.  28.  29.   2.   1.
   5.  68.  40.  57.  23.  42.  22.  74.  54.   1.  47.  37.  65.  50.
  48.  20.  12.  64.  63.  89.   8.   1.  27.  81.  79.  69.  30.  25.
  80.  35.  36.  66.  72.  52.  21.  31.  70.   1.  51.   4.  10.  34.
  82.  67.  91.  33.  71.  32.  11.   1.  14.  59.  58.  43.  16.  44.
  39.  15.  97.  90.  99.  98.  92. 101.  93.  60.   1.]
Best cost:  1191.149016739995
```

Test2

In [20]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n101-k8.vrp',\
        start_line = 7, end_line = 108, end_demand_line = 210, end_file = 215, capacity = 200, n_vehicle = 8)
sa.play()
```

```
Runtime:  11.92137796377182 min
Best answer:  [  1.  77.  51.  70.   2.  71.  52.  72.  36.  35.  82.  79.  80.  78.
  34.  30.   4.   1.  90.  96.   7.  19.  84.   6.  45.  44. 101.  16.
  98.  95.   1.   3.  23.  42.  24.  68.  40.  26.  56.  69.  81.  25.
  27.  54.   1.  28.  11.  91.  12.  89.  31.  21.  33.  64.  65.  50.
  37.  20.  63.   1.  99.  38.  93.  92.  43.  15.  94.  61.  85.  47.
  83.  48.  49.  53.   8.  32.   1.  55.   5.  74.  76.  58.  59.  41.
  73.  75.  57.  22.  14.  88.   1.  13.  66.  67.  10.  29.   1.  60.
  62.  86.  17.  39.  87.  18.  97. 100.  46.   9.   1.]
Best cost:  1270.4869546991458
```

Test3

In [21]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n101-k8.vrp',\
        start_line = 7, end_line = 108, end_demand_line = 210, end_file = 215, capacity = 200, n_vehicle = 8)
sa.play()
```

```
Runtime:  11.557971612612407 min
Best answer:  [  1.   8.  49.  50.  37.  20.  48.  63.  32.  64.  65.  12.   1.  13.
  55.  25.   5.  73.  23.  87.   1.  77.   4.  82.  10.  66.  67.  33.
   1.  51.  69.  81.  79.  72.  35.  36.  52.  21.  34.   2.  31.  11.
  91.  89.  71.  53.   1.  59.  92.  94.  84.  97.  58.   3.  41.  22.
  27.  80.  30.  78.  70.  29.  28.   1.  90.   7.  19.  83.  47.  18.
  61. 100.  88.  98.  96.  17.  85.   6.  46.   9.   1.  95.  38.  99.
  44.  43.  45.  39.  15. 101.  86.  62.  93.  54.   1.  74.  57.  26.
  56.  68.  40.  24.  75.  76.  42.  16.  60.  14.   1.]
Best cost:  1338.2229810307388
```

Test4

In [22]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n101-k8.vrp',\
        start_line = 7, end_line = 108, end_demand_line = 210, end_file = 215, capacity = 200, n_vehicle = 8)
sa.play()
```

```
Runtime:  11.880941847960154 min
Best answer:  [  1.  98.  58.   3.  75.  73.  42.  24.  68.  56.  26.  40.  27.  54.
```

```
   1.   2.   4.  79.  35.  36.  80.  78.  81.  30.  25.  69.  13.  51.
  77.  29.   1.  86.  62.  17.  85.  18.  87.  39.  15.  44.  43.  16.
  23.  41.   1.  32.  71.  21.  11.  91.  33.  64.  12.  65.  89.  63.
   8.  53.   1.  96.  88.   7.  59.  95.  61.  46.  47.  83.  84.  19.
  97. 100.  92.  94.  93.   1.  28.  70.  31.  52.  34.  82.  10.  66.
  72.  67.   1.   9.  49.  20.  50.  37.  48.   6.   1.  22.  76.   5.
  55.  57.  74.  14.  90.  99.  60.  38.  45. 101.   1.]
Best cost:  1208.7467921381276
```

Test5

In [23]:

```
sa = SA(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological computaion/project/Vrp-All/E/E-n101-k8.vrp',\
        start_line = 7, end_line = 108, end_demand_line = 210, end_file = 215, capacity = 200, n_vehicle = 8)
sa.play()
```

```
Runtime:  11.708103342851002 min
Best answer:  [  1.  98.  96. 100.  15.  39. 101.  92.  99.  84.  97.  14.  58.   3.
  43.  16.  44.  42.  75.   1.  90.   6.   9.  46.  47.  48.  83.  49.
  20.  11.  63.   1.  61.  94.  19.   7.  59.  28.   2.  71.  89.   8.
  53.  70.  77.  13.  81.  69.   1.  74.  22.  57.  68.  26.  56.   5.
  40.  25.  55.  76.  73.  23.  41.   1.  85.  18.  62.  86.  87.  45.
  17.  38.  93.  60.  88.   1.  65.  50.  37.  12.  64.  91.  33.  67.
  21.  10.  52.  31.  32.   1.  95.  24.  27.   1.  82.  34.  80.  30.
  78.   4.  72.  66.  36.  35.  79.  51.  54.  29.   1.]
Best cost:  1268.3634650385065
```