Solving CVRP with Immune System (IS)

Importing Packages

In [17]:

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

Immune Class to Perform Immune System Algorithm

In [28]:

```
class Immune:
    #Number of population members
    n_pop = 5
    #Number of customers
    n_cust = 0
    #Coefficiet, used for mutation operation probability
    p_mutation2 = -1/9
    #Number of vehicles
    n_vehicle = 0
    #Capacity of vehicles
    capacity = 0
    #Array of Answer
    pop = np.array([])
    #best overall solution
    best_s = np.array([])
    #best overall cost
    best_cost = 0
    #limits of iteration (number of iteraton which best cost remains constant)
    iteration_limit = 2000

    # 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.n_cust = self.data.shape[0]
        self.best_cost = self.energy(self.best_s)
        self.n_vehicle = n_vehicle
        self.capacity = capacity

    #Initilizing population
    def init_pop(self):
        for k in range(self.n_pop):
            pop = np.array([])
            checked = [0 for i in range(self.data.shape[0])]
            first_row = self.data.loc[1]
            first_row = first_row.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']
                        pop = np.append(pop, int(self.data.iloc[i].name))
                        checked[i] = 1
            pop = np.append(pop, 1)
            self.add_pop(pop)
        self.data.index.names = ['CUST']
        self.sort_pop()

    #Add array to population
    def add_pop(self, pop):
        '''
            pop: array, that will be added to population
        '''
        energy = self.energy(pop)
        pop = np.append(pop, energy)
        if self.pop.size != 0:
            self.pop = np.append(self.pop, np.reshape(pop, (1, -1)), axis = 0)
        else:
            self.pop = np.reshape(pop, (1, -1)).copy()
        if self.best_cost == 0 or self.best_cost > energy:
            self.best_s = pop
            self.best_cost = energy

    #Sort population based on their cost in ascending order
    def sort_pop(self):
        self.pop = self.pop[self.pop[:, self.pop.shape[1]-1].argsort()]

    #Copy from arrays based on their cost (bigger costs, have less copies) and perform mutation on
    #based on their costs
    def copy(self):
        self.sort_pop()
        for i in range(self.n_pop):
            for j in range(self.n_pop - i):
                new = self.pop[i][:self.pop.shape[1]-1].copy()
                if np.random.uniform(0, 1) < np.exp(self.p_mutation2 * (self.n_pop - i)):
                    new = self.mutation(new)
                self.add_pop(new)

    #Select new generation among copies and original arrays
    def select(self):
        selected = np.random.choice(np.arange(self.n_pop+3), self.n_pop, replace = False)
        self.sort_pop()
        self.pop = self.pop[selected]
        self.sort_pop()

    #Mutation operator on generated copies, returns the manipulated array
    def mutation(self, array):
        '''
            array: array, pass to function to perform mutation that
        '''
        cities = np.random.randint(2, self.data.shape[0]+1, 1)
        i = 0
        j = 0
        while i < 1:
            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):
                array = new
                i += 1
        return array

    #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:
                #print(c)
                return False
        return True

    #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):
        self.init_pop()
        b = self.best_cost
        i = 0
        start = time.time()
        while i < self.iteration_limit:
            i += 1
            self.copy()
            self.select()
            if b != self.best_cost:
                b = self.best_cost
                i = 0
        print('Runtime: ', (time.time() - start)/60, 'min')
        print('Best answer: ', self.best_s[:-1])
        print('Best cost: ', self.best_cost)
```

E-n51-k5 Data Set

Test1

In [36]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  7.330989476044973 min
Best answer:  [ 1.  8. 44. 25.  7. 28.  2.  4. 37. 36. 30. 50. 39. 47.  1. 14.  6. 17.
 51. 35. 22. 21. 29. 33.  1. 13. 12.  3. 23. 32. 27.  9. 49. 24.  1. 48.
 19.  5. 18. 38. 16. 11. 40. 31. 10.  1. 34. 46. 45. 43. 20. 41. 42. 26.
 15.  1.]
Best cost:  714.7841873611636
```

Test2

In [37]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  4.039527610937754 min
Best answer:  [ 1. 21. 17. 31. 40. 45. 41. 42. 13.  1. 47. 39.  3.  2. 28.  9. 44. 26.
 14.  1. 15. 25.  8. 27. 29.  4. 10. 50. 34.  6.  1. 48. 18. 16. 46. 11.
 35. 51. 22. 30. 36. 37. 32. 23. 33.  1. 49. 24.  7. 19.  5. 20. 43. 38.
 12.  1.]
Best cost:  812.507555587564
```

Test3

In [38]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  3.336073672771454 min
Best answer:  [ 1. 47.  6. 16. 46. 40. 11. 10. 51. 17.  3. 23. 49.  1. 38. 45. 41. 42.
 14. 15.  9.  2. 12.  1. 48.  5. 20. 19. 33. 29. 27.  8. 28.  1. 26. 25.
 44.  7. 21. 35. 31. 34.  1. 13. 18. 43. 50. 39. 22. 30. 36. 37.  4. 32.
 24.  1.]
Best cost:  841.523468921067
```

Test4

In [39]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  4.853357863426209 min
Best answer:  [ 1. 48. 18. 45. 43. 20. 41. 42. 26.  7.  1. 28. 49.  8.  9.  2. 35.  5.
 19.  1. 13. 38. 16. 11. 50. 51. 22. 30. 36. 37.  3. 33.  1. 15. 25. 44.
 24. 27. 32. 29. 17. 31. 10. 12. 47.  1. 14. 46. 34. 40.  6. 39. 21.  4.
 23.  1.]
Best cost:  757.423282923279
```

Test5

In [40]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  3.6749087810516357 min
Best answer:  [ 1. 47. 34. 11. 51. 36. 37.  9. 24. 25. 26.  7.  1. 13. 19.  5. 45. 38.
  6.  3.  1. 28.  8. 44. 15. 14. 42. 41. 20. 43. 46.  1. 49. 27. 32. 29.
 21. 22. 31. 17. 12. 33.  1.  2. 23.  4. 30. 35. 10. 39. 50. 40. 16. 18.
 48.  1.]
Best cost:  768.1907977535595
```

E-n101-k8 Data Set

Test1

In [33]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  9.712151718139648 min
Best answer:  [  1.  29.  77.  78.   4.  80.  79.  72.  67.  66.  36.  82.  34.  51.
   1.  90.  61.  84.   9.  47.  46.  18.  85.   6.  60.  96.  95.   1.
  14.  58.  16.  44.  87.  17.  62.  86.  94. 100.  97.   7.   1.  32.
  11.  33.  91.  64.  63.  89.   2.  69.  81.  13.   1.  73.  76.  57.
  24.  68.  40.  26.  56.   5.  75.  23.  42.   3.   1.  54.  59.  41.
  22.  74.  88.  43.  15.  39.  45.  92. 101.  99.  38.  93.  98.   1.
  27.  55.  25.  30.  35.  10.  52.  21.  31.  71.  70.  28.   1.  53.
   8.  20.  12.  65.  50.  37.  48.  49.  83.  19.   1.]
Best cost:  918.5313419276305
```

Test2

In [41]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  14.49408462047577 min
Best answer:  [  1.  19.  83.   8.  89.  32.   2.  69.  81.  25.  30.  78.  77.  29.
   1.  95.  99.  38. 101.  39.  87.  92.  86.  94. 100.  97.   1.  10.
  36.  72.  66.  67.  21.  33.  12.  20.  37.  47.   9.  84.  61.  90.
   1.   7.  62.  17.  45.  15.  43.  44.  16.  42.  57.  24.  68.  40.
  26.  56.   1.  79.  35.  80.   4.  13.  22.  74.  75.  76.  23.   3.
  58.  88.  14.   1.  28.  70.  71.  11.  63.  50.  65.  64.  91.  31.
  52.  82.  34.  51.   1.  53.  49.  48.  46.  18.  85.   6.  60.  93.
  98.  96.   1.  27.  55.   5.  73.  41.  59.  54.   1.]
Best cost:  1004.7069723718977
```

Test3

In [42]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  8.269693970680237 min
Best answer:  [  1.  54.  41.  74.  73.  79.  82.  34.  29.  95.  96.  58.  16.  44.
  43.   1.  53.   8.  83.  49.  48.  20.  12.  11.  32.   1.  27.  13.
  55.   5.  26.  56.  25.  30.  35.  36.  10.  51.  46.  18.  85.   6.
  61.   1.  93.  99.  38.  15.  39.  45.  92.  86.  94. 100.  66.  72.
  77.   1.  52.  67.  21.  71.  90.   7.  98.  60.  62.  84.  65.  50.
  37.  47.   9.  19.   1.  78.   4.  80.  69.  81.  88. 101.  87.  17.
  97.   1.  28.  70.   2.  31.  33.  91.  64.  63.  89.   1.  14.   3.
  42.  23.  75.  76.  24.  68.  40.  57.  22.  59.   1.]
Best cost:  1220.0810146618462
```

Test4

In [43]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  12.81115002632141 min
Best answer:  [  1.  28.  70.  52.  10.  82.  34.  51.   1.   7. 100.  86.  62.   6.
  61.  19.   8.  89.  71.  31.  67.  21.   2.   1.  27.   5.  40.  56.
  25.  30.  35.  36.  66.  72.  79.  80.  78.  29.   1.  53.  11.  63.
  12.  65.  50.  37.  47.  46.  18.  85.  99.  93.  98.  22.  74.  58.
  16.  43.   1.  32.  33.  91.  64.  20.  48.  49.  83.   9.  84.  90.
   1.  97.  92.  39.  15. 101.  38.  88.  75.  73.  81.  69.   4.  77.
   1.  95.  96.  14.   3.  42.  23.  76.  24.  57.  41.  59.  54.   1.
  13.  55.  26.  68.  44.  45.  87.  17.  94.  60.   1.]
Best cost:  1091.2710886042316
```

Test5

In [44]:

```
immune = Immune(\
        directory = '/media/amirabbas/287935d9-b220-4347-beed-981bb0f7821a/personal/university/6th term/biological compu
taion/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)
immune.play()
```

```
Runtime:  8.323703189690908 min
```

```
Runtime:  0.328708169090906 min
Best answer:  [  1.  28.  70.  71.  31.  51.   7. 100.  94.  97.   1.   8.  83.  49.
  48.  50.  37.  47.   9.  14.  74.  22.  41.  54.   1.  95.  60.  99.
  38. 101.  92.  86.   6.  18.  46.  84.  19.   1.  73.  75.  23.  42.
  76.  68.   5.  96.  93.  98.  88.  44.  16.  58.   1.  27.  55.  25.
  30.  10.  52.  21.  67.  66.  35.  79.  80.  77.  29.   1.  78.   4.
   2.  32.  11.  33.  91.  64.  65.  12.  20.  63.  89.  53.   1.  59.
   3.  43.  15.  45.  39.  87.  17.  62.  85.  61.  90.   1.  13.  56.
  26.  40.  24.  57.  81.  69.  36.  72.  82.  34.   1.]
Best cost:  1083.9617623363058
```