Filippo_Casari_MSCAI22_hw3

December 6, 2022

1 Homework 3 (Tutorial 7)

##MA course in Artificial Intelligence 2022/2023

@author: Filippo Casari

[ ]:
```
#!rm -r AI2022MA/
#!git clone https://github.com/UmbertoJr/AI2022MA.git &> /dev/null
```

[ ]:
```
# Imports

#from AI2022MA.IO_manager.io_tsp import TSP_Instance_Creator
# if you are running from your local remove the prefix AI2020 (comment the␣
 ↪previous line and uncomment the following line)
from IO_manager.io_tsp import TSP_Instance_Creator

ic = TSP_Instance_Creator("standard", 'eil76.tsp')
ic.print_info()
#ic.plot_data()
```

```
name: eil76
nPoints: 76
best_sol: 538.0
```

[ ]:
```
ic = TSP_Instance_Creator("standard", 'ch130.tsp')
ic.print_info()
#ic.plot_data()
```

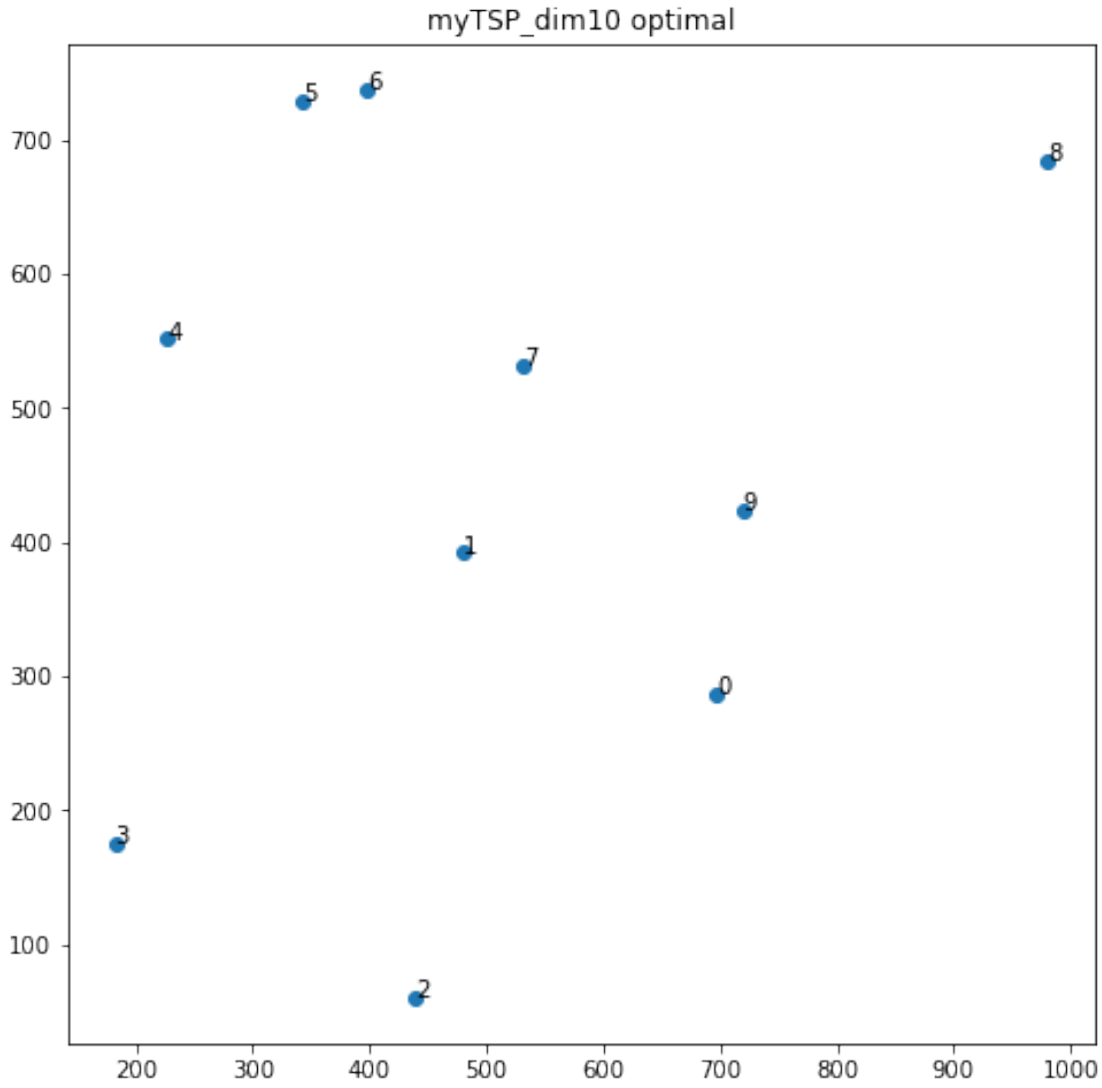
```
name: ch130
nPoints: 130
best_sol: 6110.0
```

[ ]:
```
ic = TSP_Instance_Creator("standard", 'd198.tsp')
ic.print_info()
#ic.plot_data()
```

```
name: d198
nPoints: 198
best_sol: 15780.0
```

[ ]:
```
ic = TSP_Instance_Creator("standard", 'myTSP_dim10.tsp')
ic.print_info()
ic.plot_data()
```

```
name: myTSP_dim10
nPoints: 10
best_sol: 2732.0
```



[ ]:
```
import time
from IPython import display
from matplotlib import pyplot as plt
%matplotlib notebook
```

```
def plot_tour(instance, tour, ant):
    """plot iterativerly the tour for that ant

    Args:
        instance (TSP_Instance_Creator): TSP problem
        tour (list): tour of the ant
        ant (int): ant number
    """
    plt.figure(figsize=(8, 8))
    plt.grid()
    plt.title(f"Tour Ant # {ant}")
    plt.scatter(instance.points[:, 1], instance.points[:, 2])
    for t in range(len(tour)-1):
        xy1, xy2 = tour[t], tour[t+1]
        plt.plot([instance.points[xy1, 1], instance.points[xy2, 1]], [
                 instance.points[xy1, 2], instance.points[xy2, 2]],␣
 ↪color="blue")
        display.clear_output(wait=True)
        display.display(plt.gcf())
        time.sleep(0.1)
```

[ ]:
```
import numpy as np
import random
from solvers import local_search
from solvers.constructive_algorithms import nn
from solvers.two_opt_with_candidate import twoOpt_with_cl
from threading import Thread
from time import sleep
# nn takes as input the distance matrix and returns
# the tour and the length constructed with nearest neighbor, i.e.   tour, len_t␣
 ↪= nn(dist_mat)

# twoOpt takes as input the solution, the actual_len and the distance matrix
# and returns the tour and the length created with 2-opt, i.e.     tour, lent_t␣
 ↪= twoOpt(solution, actual_len, dist_mat)


class ACS:
    m = 10
    beta = 2
    alpha = rho = 0.1
    cl = 15
    q0 = 0.98
    stop_after_secs = 0

    @staticmethod
```

```
    def take_candidates(j, dist_mat):
        """take candidate list

        Args:
            j (_type_): city name
            dist_mat (_type_): distance matrix

        Returns:
            _type_: np.array containing candidate list
        """
        return (np.argsort(dist_mat[j])[1:ACS.cl+1])

    @staticmethod
    def take_other_cities(j, dist_mat):
        """return cities not in the candidate list

        Args:
            j (_type_): city
            dist_mat (_type_): distance matrix

        Returns:
            _type_: list of cities close to city but not in candidate list
        """
        return (np.argsort(dist_mat[j])[ACS.cl+1:])

    def __init__(self, instance, q0=None, boost=False, timeStop=180):
        """Constructor

        Args:
            instance (_type_):problem to solve
            q0 (float, optional): . Defaults to None.
            boost (bool, optional): applying 2opt or not. Defaults to False.
            timeStop (int, optional): time after which stopping the algorithm.␣
↪Defaults to 180.
        """
        self.boost = boost
        self.stop_after_secs = timeStop
        if (q0 != None):
            self.q0 = q0
        self.instance = instance
        self.n = instance.nPoints
        self.dist_mat = instance.dist_matrix
        _, self.L_nn = nn(instance.dist_matrix,
                          starting_node=np.random.choice(self.n))
        self.tau0 = 1./(float(self.n) * self.L_nn)

        # position collector for the Ants, TO BE UPDATED during the steps
```

```
        self.position = {i: None for i in range(ACS.m)}
        self.tour = {i: []
                     for i in range(ACS.m)}  # tour collector for the Ants
        #self.pheromone = {r: [self.tau0]*ACS.cl for r in range(self.n)}
        self.pheromone = {r: [self.tau0]*(self.n) for r in range(self.n)}
        self.candidate_list = {r: ACS.take_candidates(r, instance.dist_matrix)␣
↪for r in range(
            self.n)}  # per tutte le citta', prende le citta vicine
        #self.eta = {r: [1/self.dist_mat[r, s] for s in ACS.take_candidates(r,␣
↪instance.dist_matrix)] for r in range(self.n)}
        self.eta = {r: [1./(self.dist_mat[r, s]+np.finfo(np.double).eps)
                        for s in range(self.n)] for r in range(self.n)}
        for k in self.eta:
            self.eta[k][k] = 0.
        self.tour_len = {i: 0. for i in range(ACS.m)}
        self.best_tour = []
        self.best_ant = 0
        self.best_tour_len = 10e10
        self.iterations = 1000000
        self.tour_len_global = []
        self.current_iteration = 0
        self.tour_len_over_iters = []
        self.count_exploitation = 0
        self.count_exploration = 0
        self.best_len = 0.
        self.best_tour_tmp = []

    def global_update(self):
        """Global pheromone updating
        """
        for k1, val1 in self.pheromone.items():

            for k2, val2 in self.pheromone.items():

                if (k1 == k2):
                    self.pheromone[k1][k2] = 0.
                else:
                    delta_tau = 0.
                    if (k1 in self.best_tour_tmp and k2 in self.best_tour_tmp):
                        delta_tau = 1./self.best_len

                    self.pheromone[k1][k2] = (
                        1. - self.alpha) * self.pheromone[k1][k2] + (self.alpha␣
↪* delta_tau)
                    self.pheromone[k2][k1] = (
                        1. - self.alpha) * self.pheromone[k2][k1] + (self.alpha␣
↪* delta_tau)
```

```
    def solve(self):
        """Solving ACS
        """
        threeMinutes = Thread(target=lambda: sleep(self.stop_after_secs))
        threeMinutes.start()
        for iteration in range(self.iterations):
            self.current_iteration = iteration

            self.loop()
            best_tour_ant = min(self.tour_len, key=self.tour_len.get)

            self.best_len = self.tour_len[best_tour_ant]

            self.best_tour_tmp = self.tour[best_tour_ant]
            if (self.best_len < self.best_tour_len):
                self.best_tour_len = self.best_len
                self.best_ant = best_tour_ant
                self.best_tour = self.best_tour_tmp

            if (self.boost):
                self.two_Opt_Improvement()
            self.global_update()
            self.tour_len_over_iters.append(self.best_len)
            self.tour_len_global.append(np.mean(list(self.tour_len.values())))

            if not threeMinutes.is_alive():
                break
            self.tour = {i: [] for i in range(ACS.m)}
            self.position = {i: None for i in range(self.m)}
            self.tour_len = {i: 0. for i in range(self.m)}

    def local_update(self, k):
        """local pheromone update

        Args:
            k (int): ant number
        """
        old_city = self.tour[k][-2]
        next_city = self.position[k]
        old_ph = self.pheromone[old_city][next_city]
        pheromon = ((1.-self.rho) * old_ph) + (self.rho * self.tau0)
        self.pheromone[old_city][next_city] = pheromon
        self.pheromone[next_city][old_city] = pheromon
        self.tour_len[k] += self.dist_mat[old_city, next_city]

    def loop(self):
```

```
        """each ant contructs its own tour
        """

        for node_idx in range(self.n):

            if (node_idx < self.n-1):

                for k in range(self.m):
                    if (self.position[k] == None):
                        self.position[k] = np.random.randint(
                            low=0, high=self.n-1)

                        self.tour[k].append(self.position[k])
                    q = np.random.random()

                    eta_beta = np.power(self.eta[self.position[k]], self.beta)

                    s_list = eta_beta * \
                        np.array(self.pheromone[self.position[k]])
                    s_list_cities = np.argsort(s_list)

                    if (q < self.q0):
                        self.count_exploitation += 1
                        #next_city = max(s_dict, key=s_dict.get)

                        for i in range(self.n):
                            if (s_list_cities[self.n-1-i] not in self.tour[k]):
                                next_city = s_list_cities[self.n-1-i]
                                break

                        self.position[k] = next_city
                        self.tour[k].append(next_city)

                    else:

                        possible_cities = []
                        new_s_list = []
                        for i in range(self.n):
                            if (i not in self.tour[k]):
                                new_s_list.append(s_list[i])

                                possible_cities.append(i)
                        sum_val = sum(new_s_list)
                        prob = new_s_list/sum_val

                        next_city = random.choices(
                            possible_cities, weights=prob, k=1)[0]
```

```
                        self.tour[k].append(next_city)
                        self.position[k] = next_city

                        self.count_exploration += 1

                    self.local_update(k)
            else:
                for k in range(self.m):
                    self.tour[k].append(self.tour[k][0])
                    self.tour_len[k] += self.dist_mat[self.tour[k]
                                                      [-2], self.tour[k][-1]]

    def two_Opt_Improvement(self):
        """two opt with CL
        """
        best_tour_new, best_tour_len_new = twoOpt_with_cl(
            self.best_tour_tmp, self.best_len, self.dist_mat, self.
 ↪candidate_list)
        if (best_tour_len_new < self.best_len):
            self.best_len, self.best_tour_tmp = best_tour_len_new, best_tour_new
            if (self.best_len < self.best_tour_len):

                self.best_tour, self.best_tour_len = best_tour_new,␣
 ↪best_tour_len_new
```

[ ]:
```
import pandas as pd
%matplotlib tk
seeds = [0, 42, 123]
q_s = [0.5, 0.98, 1.]
instances = ["eil76.tsp", "ch130.tsp", "d198.tsp"]
results = pd.DataFrame(columns=['q', 'boost', 'best integer len',
                        '# iterations', 'AVG integer len', 'STD ', 'Optimum',␣
 ↪'Rel error'])
counter = -1
results.to_csv("CSV/results5.csv")
list_len_over_iter = []
list_of_len_global = []
for q in q_s[1:2]:

    if (q == 1.):
        q = q-(13./ic.nPoints)
    print(f"-------  Q0 = {q} ---------")
    for seed in seeds:
        print(f"-------  SEED = {seed} ---------")
        random.seed(seed)
        np.random.seed(seed)
```

```
        for boost in [True, False]:
            print(f"-------  Boost/2opt = {boost} ---------")

            results = {i: [] for i in instances}
            for instance in instances:

                print(f"-------  INSTANCE = {instance} ---------")
                ic = TSP_Instance_Creator("standard", instance)

                acs = ACS(ic, q, boost, 180)
                # print(acs.pheromone)
                # print(acs.candidate_list)
                acs.solve()
                results[instance].append(q)
                results[instance].append(boost)
                results[instance].append(acs.best_tour_len)
                results[instance].append(acs.current_iteration)
                results[instance].append(np.mean(acs.tour_len_global))
                results[instance].append(np.std(acs.tour_len_global))
                results[instance].append(ic.best_sol)
                results[instance].append(
                    ((acs.best_tour_len-ic.best_sol)*100.)/ic.best_sol)
                list_of_len_global.append(np.mean(acs.tour_len_global))
                list_len_over_iter.append(acs.tour_len_over_iters)

                print("best gap (%) = ",
                      ((acs.best_tour_len-ic.best_sol)*100.)/ic.best_sol)
                print("best cost = ", acs.best_tour_len)
                print("numbers of tours generated = ",
                      acs.current_iteration*acs.m)

            counter += 1
            results = pd.DataFrame(results).T
            results.columns = ['q', 'boost', 'best integer len',
                               '# iterations', 'AVG integer len', 'STD ',␣
↪'Optimum', 'Rel error']
            results.to_csv("CSV/results5.csv", mode='a', header=False)
            print(results)
```

```
Warning: Cannot change to a different GUI toolkit: tk. Using notebook instead.
-------  Q0 = 0.98 ---------
-------  SEED = 0 ---------
-------  Boost/2opt = True ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  2.973977695167286
best cost =  554.0
numbers of tours generated =  26380
```

```
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  4.25531914893617
best cost =  6370.0
numbers of tours generated =  10480
-------  INSTANCE = d198.tsp ---------
best gap (%) =  7.522179974651458
best cost =  16967.0
numbers of tours generated =  5420
              q boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  True            554.0         2638      762.828837
ch130.tsp  0.98  True           6370.0         1048     8686.563584
d198.tsp   0.98  True          16967.0          542    22155.424678

                 STD   Optimum Rel error
eil76.tsp   12.241814    538.0  2.973978
ch130.tsp  145.095826   6110.0  4.255319
d198.tsp   349.936575  15780.0   7.52218
-------  Boost/2opt = False ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  10.594795539033457
best cost =  595.0
numbers of tours generated =  90850
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  17.626841243862522
best cost =  7187.0
numbers of tours generated =  25370
-------  INSTANCE = d198.tsp ---------
best gap (%) =  15.481622306717364
best cost =  18223.0
numbers of tours generated =  8650
              q  boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  False            595.0         9085      761.982468
ch130.tsp  0.98  False           7187.0         2537      8685.18357
d198.tsp   0.98  False          18223.0          865    22182.932333

                 STD   Optimum  Rel error
eil76.tsp   12.267007    538.0  10.594796
ch130.tsp  137.854559   6110.0  17.626841
d198.tsp     345.9232  15780.0  15.481622
-------  SEED = 42 ---------
-------  Boost/2opt = True ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  2.7881040892193307
best cost =  553.0
numbers of tours generated =  27050
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  3.6824877250409167
best cost =  6335.0
```

```
numbers of tours generated =  10810
-------  INSTANCE = d198.tsp ---------
best gap (%) =  7.414448669201521
best cost =  16950.0
numbers of tours generated =  5450
              q boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  True            553.0         2705      762.646452
ch130.tsp  0.98  True           6335.0         1081     8682.119131
d198.tsp   0.98  True          16950.0          545    22183.670696

                STD   Optimum Rel error
eil76.tsp   12.385073    538.0  2.788104
ch130.tsp  145.839661   6110.0  3.682488
d198.tsp   358.742756  15780.0  7.414449
-------  Boost/2opt = False ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  14.312267657992566
best cost =  615.0
numbers of tours generated =  90750
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  15.793780687397708
best cost =  7075.0
numbers of tours generated =  25290
-------  INSTANCE = d198.tsp ---------
best gap (%) =  16.134347275031686
best cost =  18326.0
numbers of tours generated =  8630
              q  boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  False            615.0         9075      762.041935
ch130.tsp  0.98  False           7075.0         2529     8681.988498
d198.tsp   0.98  False          18326.0          863    22180.026736

                STD   Optimum  Rel error
eil76.tsp   12.200081    538.0  14.312268
ch130.tsp  139.869648   6110.0  15.793781
d198.tsp    352.19132  15780.0  16.134347
-------  SEED = 123 ---------
-------  Boost/2opt = True ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  2.41635687732342
best cost =  551.0
numbers of tours generated =  27050
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  4.5499181669394435
best cost =  6388.0
numbers of tours generated =  10750
-------  INSTANCE = d198.tsp ---------
best gap (%) =  9.024081115335868
```

```
best cost =  17204.0
numbers of tours generated =  5490
              q boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  True            551.0         2705      762.657354
ch130.tsp  0.98  True           6388.0         1075     8688.969238
d198.tsp   0.98  True          17204.0          549    22162.425091

                 STD   Optimum Rel error
eil76.tsp    12.33162    538.0  2.416357
ch130.tsp  150.051927   6110.0  4.549918
d198.tsp   364.426275  15780.0  9.024081
-------  Boost/2opt = False ---------
-------  INSTANCE = eil76.tsp ---------
best gap (%) =  14.312267657992566
best cost =  615.0
numbers of tours generated =  90620
-------  INSTANCE = ch130.tsp ---------
best gap (%) =  12.553191489361701
best cost =  6877.0
numbers of tours generated =  25310
-------  INSTANCE = d198.tsp ---------
best gap (%) =  15.17743979721166
best cost =  18175.0
numbers of tours generated =  8620
              q  boost best integer len # iterations AVG integer len  \
eil76.tsp  0.98  False            615.0         9062      762.262992
ch130.tsp  0.98  False           6877.0         2531     8686.274882
d198.tsp   0.98  False          18175.0          862    22198.765933

                 STD   Optimum  Rel error
eil76.tsp   12.353686    538.0  14.312268
ch130.tsp  140.373739   6110.0  12.553191
d198.tsp    349.61374  15780.0   15.17744
```

[ ]:
```
%matplotlib inline
n_plots = len(list_len_over_iter)
num_rows = int(n_plots/3)
fig, axes = plt.subplots(nrows=num_rows, ncols=3 , figsize=(50, 80))
plt.grid()
row =-1
#print(axes.size)
cols = ['TSP Problem {}'.format(instances[col]) for col in range(3)]
rows = ['TSP {}'.format(row) for row in range(int(len(list_len_over_iter)/3))]
# ((np.array(list_len_over_iter[i])-ic.best_sol)*100.)/ic.best_sol
count =-3
for ax, col in zip(axes[0], cols):
    ax.set_title(col)
```

```
boost_list=["2OPT\nMEAN=RED LINE", "NO 2 OPT\nMEAN=RED LINE"]

print(np.array(list_len_over_iter[0]).flatten())
#print("len of array of statistics", np.array(list_len_over_iter))
opt_solutions = [TSP_Instance_Creator("standard", instances[0]).best_sol,␣
 ↪TSP_Instance_Creator("standard", instances[1]).best_sol,␣
 ↪TSP_Instance_Creator("standard", instances[2]).best_sol]

for i in range(3):
    count_step = 3
    count = -3
    step_mean = int(300./(i+1))
    legend_index =0
    for ax, row in zip(axes[:,i], rows):
        print("row # ", row)
        ax.set_ylabel("GAP", rotation=0, size='large')
        ax.set_xlabel("Iterations", rotation=0, size='large')
        ax.grid()

        #ax.plot(list(range(len(list_len_over_iter[count+3]))),␣
 ↪list_len_over_iter[count+3])
        #ax.plot([np.argmin(list_len_over_iter[count+3])],␣
 ↪min(list_len_over_iter[count+3]), color='red')
        means =[]

        count_m = -step_mean
        lenght_x = []
        #print("len of the array", len(list_len_over_iter[i+count+count_step]))
        if(count+count_step+i<len(list_len_over_iter)):
            for j in np.array(list_len_over_iter[i+count+count_step])[::
 ↪step_mean]:
                means.append((np.mean(np.
 ↪array(list_len_over_iter[i+count+count_step]).flatten()[count_m+step_mean:
 ↪count_m+2*step_mean])-opt_solutions[i])*100./opt_solutions[i])
                lenght_x.append(count_m+step_mean)
                count_m+=step_mean
            #print("len mean ", lenght_x, "values mean ", means)


            ax.
 ↪plot(list(range(int(len(list_len_over_iter[i+count+count_step])))),( np.
 ↪array(list_len_over_iter[i+count+count_step])-opt_solutions[i])*100./
 ↪opt_solutions[i], label="GAP")
            try:
                ax.plot(lenght_x, means, color='red',␣
 ↪label=boost_list[legend_index%2])
```

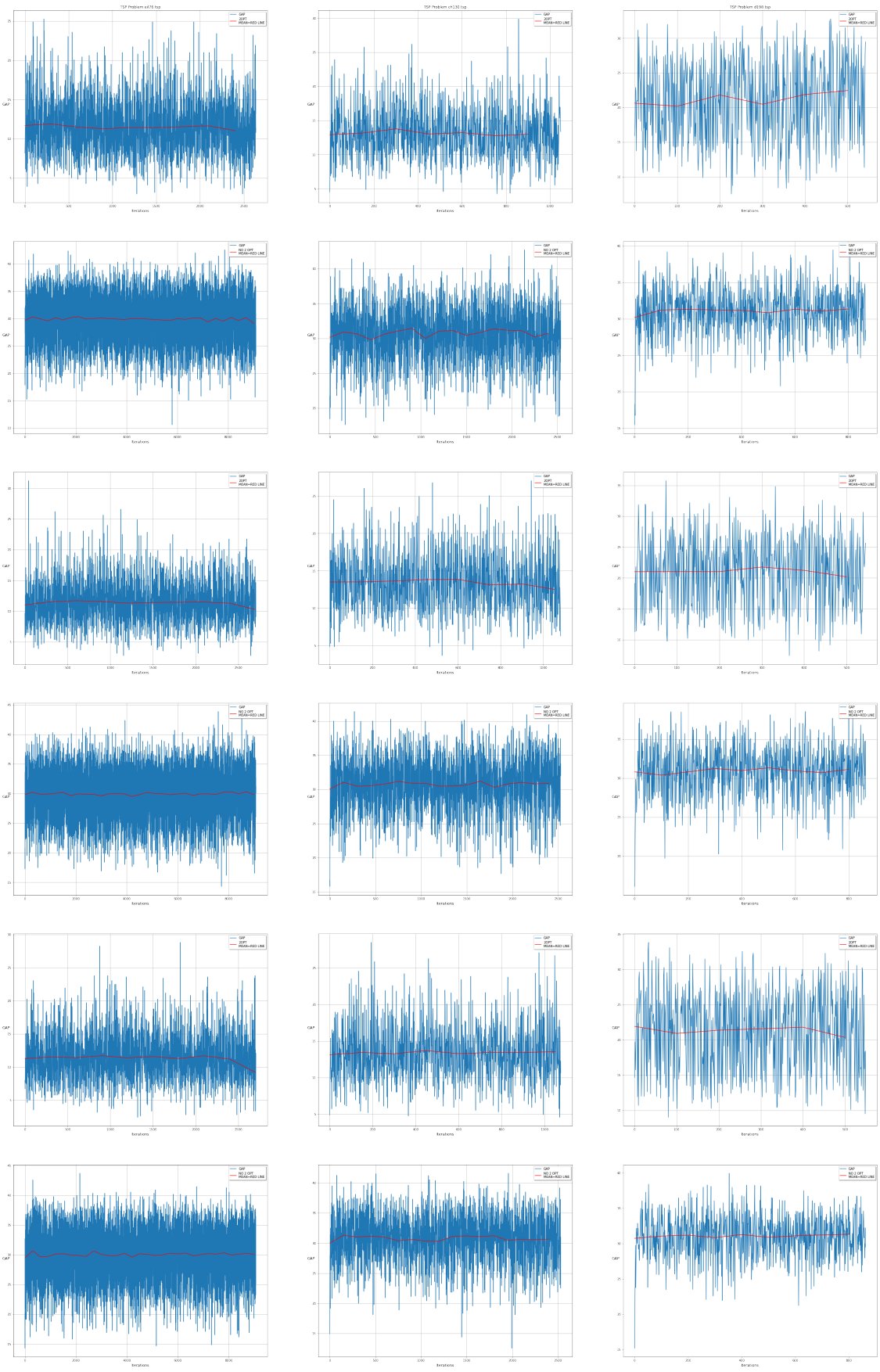
```
                ax.legend(loc="upper right", shadow=True)
                legend_index+=1

            except:
                print("out of range")

            count+=count_step


    #plt.plot([np.argmax(acs.tour_len_over_iters), np.argmin(acs.
 ↪tour_len_over_iters)], [max(acs.tour_len_over_iters), min(acs.
 ↪tour_len_over_iters)], color="blue")
plt.show()
```

```
[582. 585. 589. … 606. 596. 599.]
row #  TSP 0
row #  TSP 1
row #  TSP 2
row #  TSP 3
row #  TSP 4
row #  TSP 5
row #  TSP 0
row #  TSP 1
row #  TSP 2
row #  TSP 3
row #  TSP 4
row #  TSP 5
row #  TSP 0
row #  TSP 1
row #  TSP 2
row #  TSP 3
row #  TSP 4
row #  TSP 5
```


TSP Problem st70.tsp
TSP Problem ch130.tsp
TSP Problem d198.tsp
GAP
NO 2 OPT
MEAN+RED LINE
Iterations

1.1 ANT TOUR (REALTIME)

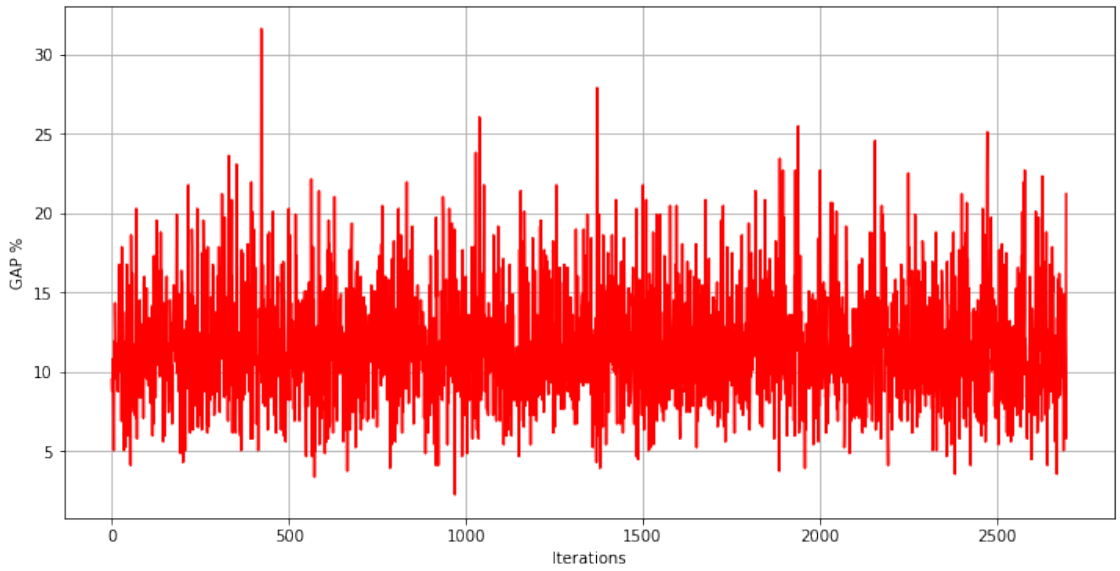
I will show how the best ant will build its own tour on the TSP eil76

[ ]:
```
instance = instances[0]
print(instance)
ic = TSP_Instance_Creator("standard", instance)
np.random.seed(43)
acs = ACS(ic, 0.98, boost=True, timeStop=180)
#print(acs.tour_len_over_iters)
#print(acs.eta)
acs.solve()
```

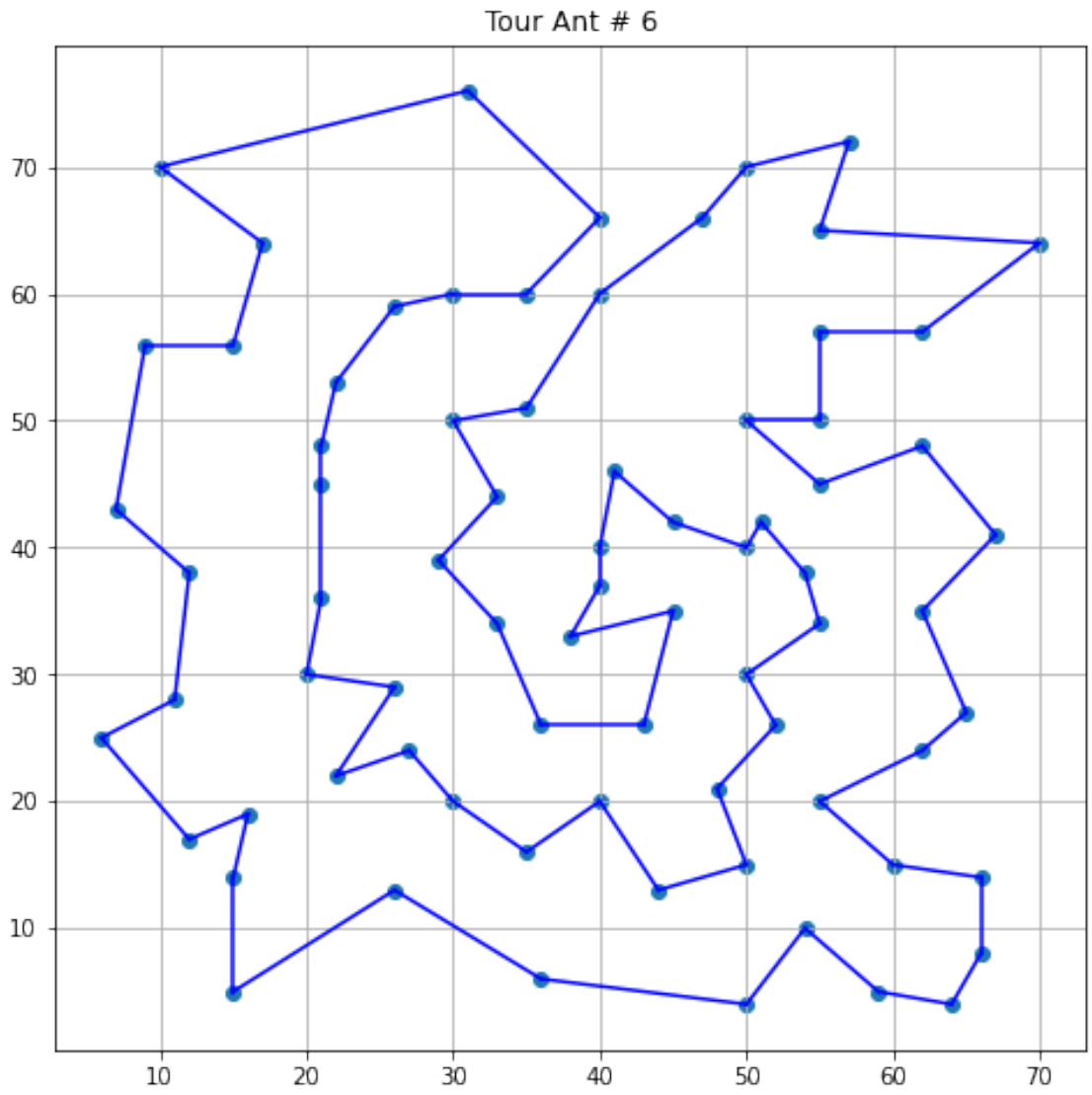
```
eil76.tsp
```

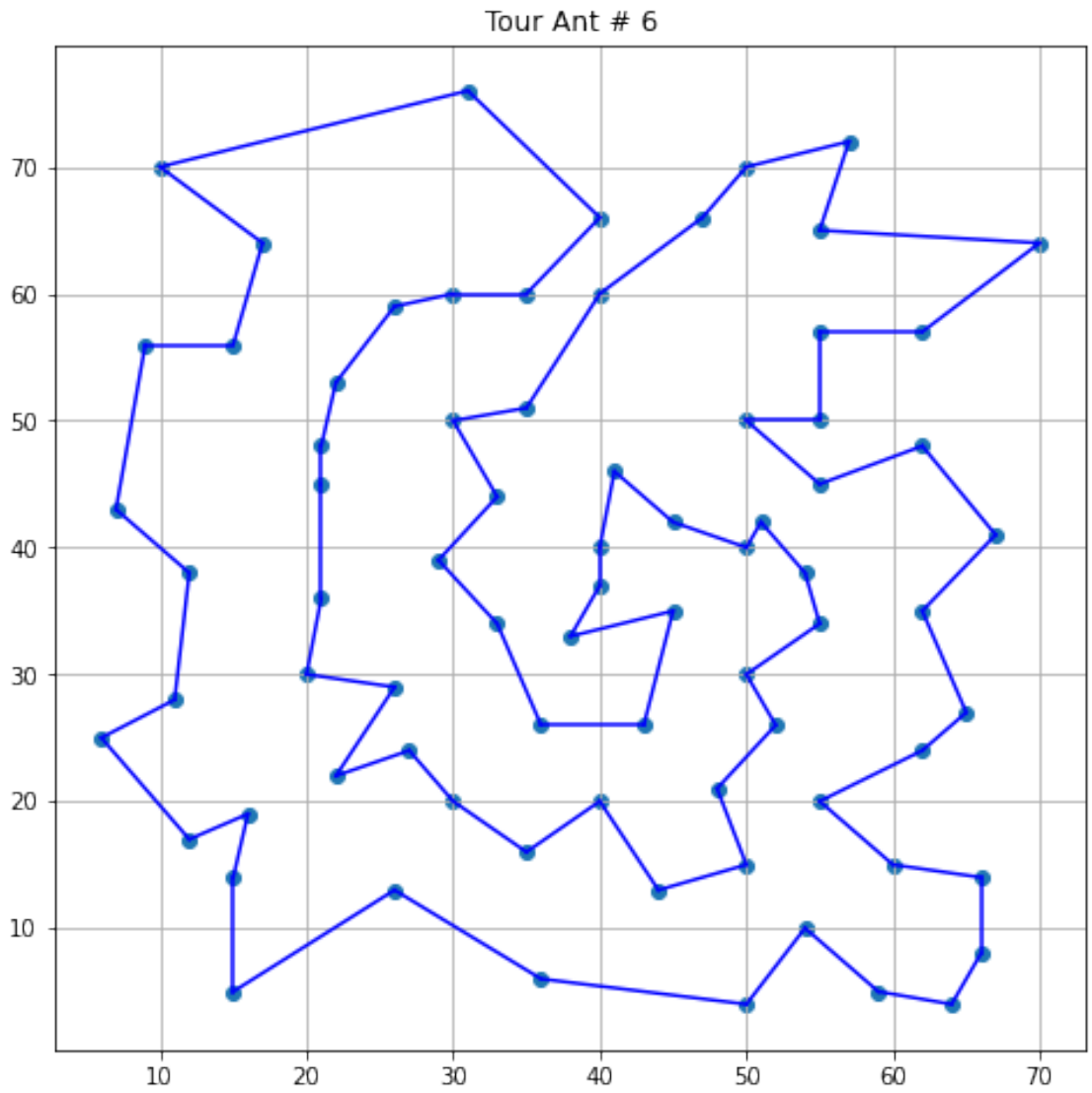
[ ]:
```
%matplotlib inline
print(np.argsort(acs.tour_len_over_iters[:]))
print(type(acs.tour_len_over_iters[0]))
total= float(acs.count_exploitation)+float(acs.count_exploration)
print("exploitations= ", float(acs.count_exploitation)*100/total, " % ")
print("solution NN: ", acs.L_nn, " mine: ", acs.best_tour_len)
fig= plt.figure(figsize=(12, 6))
plt.grid()
plt.ylabel("GAP %")
plt.xlabel("Iterations")
plt.plot(list(range(int(acs.current_iteration)+1)), ((np.array(acs.
 ↪tour_len_over_iters[::])-ic.best_sol)*100.)/ic.best_sol, color="red")
#plt.plot([np.argmax(acs.tour_len_over_iters), np.argmin(acs.
 ↪tour_len_over_iters)], [max(acs.tour_len_over_iters), min(acs.
 ↪tour_len_over_iters)], color="blue")
plt.show()
```

```
[ 969  572 2382 … 1038 1371  422]
<class 'numpy.ndarray'>
exploitations=  97.99090347299469  %
solution NN:  636.0  mine:  550.0
```



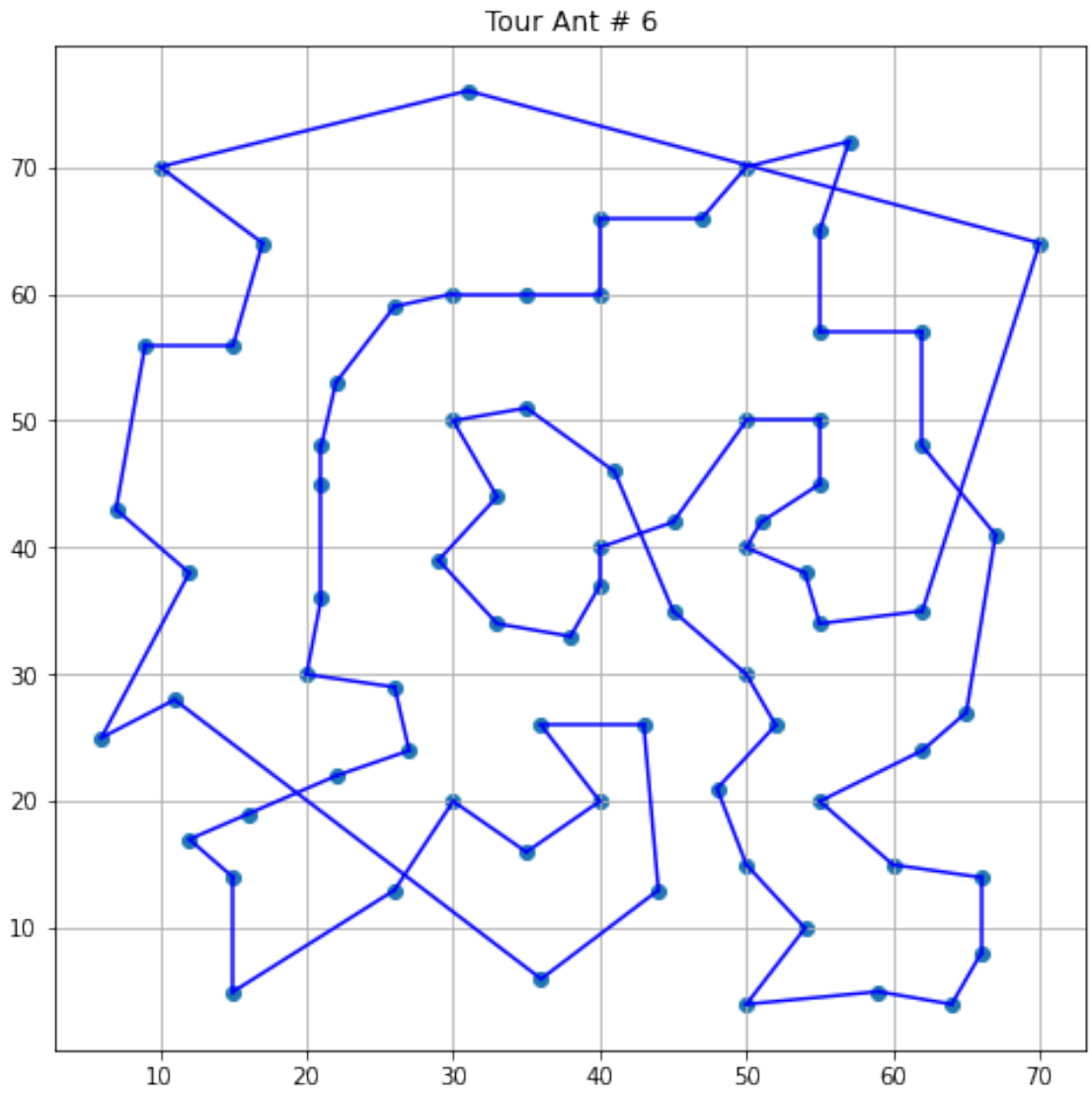
[ ]:
```
%matplotlib inline
plot_tour(acs.instance, acs.best_tour,acs.best_ant)
```

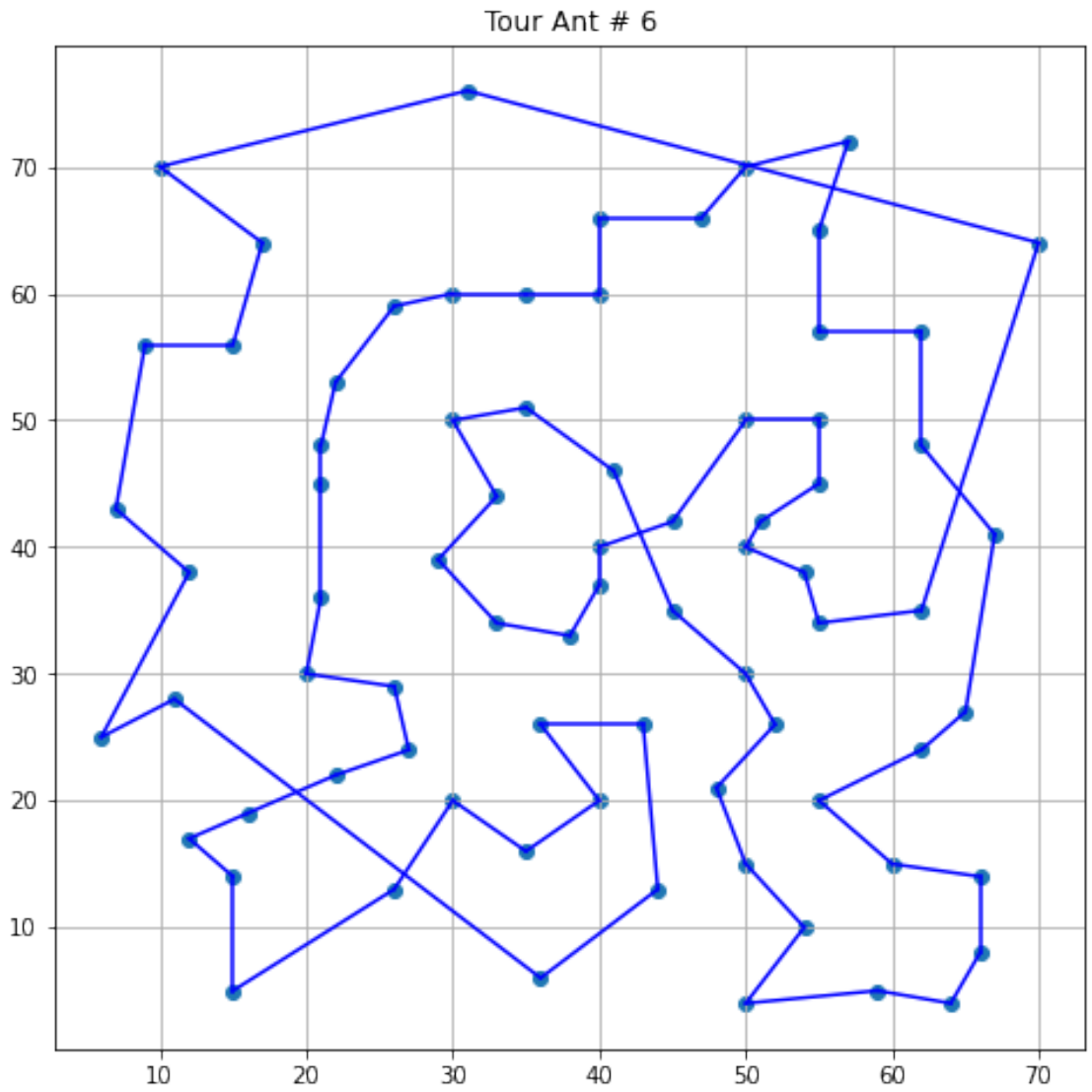

Tour Ant # 6
70
60
50
40
30
20
10
10
20
30
40
50
60
70



Comparing with NN tour solution

[ ]:
```
%matplotlib inline
t, L_nn = nn(acs.instance.dist_matrix,
                          starting_node=np.random.choice(acs.n))
t.append(t[0])
plot_tour(acs.instance, t,acs.best_ant)
```


Tour Ant # 6
70
60
50
40
30
20
10
10
20
30
40
50
60
70



2 REPORT

Overall, I got good results with the ANT colony system as shown in the plots above. Indeed, if one looks at the results called "results_final". I run different combinations: 3 different q0, 2 different variants, 3 different seeds, and 3 different instances.
For instance, for problem eil76 I got the minimum value equal to 2.97. For plots I only considered just ANTS with q0=0.98.

[ ]:
```
data = pd.read_csv("CSV/results_final.csv")
data = data.rename(columns={"Unnamed: 0": "Instance"})

print("Min gap % for instance eil76.tsp:\n")
```

```
print(data.loc[data["Instance"] =="eil76.tsp"].min())
print("\nMin gap % for instance ch130.tsp:\n")
print(data.loc[data["Instance"] =="ch130.tsp"].min())
print("\nMin gap % for instance d198.tsp:\n")
print(data.loc[data["Instance"] =="d198.tsp"].min())
array_min_gap = [data.loc[data["Instance"] =="eil76.tsp"]["Rel error"].min(),␣
 ↪data.loc[data["Instance"] =="ch130.tsp"]["Rel error"].min(), data.
 ↪loc[data["Instance"] =="d198.tsp"]["Rel error"].min()]
plt.figure(figsize=(12,8))
plt.bar(instances, array_min_gap)
plt.xlabel("INSTANCES")
plt.ylabel("GAP %")
plt.grid()
plt.title("BEST RESULTS")
plt.show()
```

```
Min gap % for instance eil76.tsp:

Instance             eil76.tsp
q                          0.5
boost                    False
best integer len         554.0
# iterations              392
AVG integer len     761.133842
STD                  12.307499
Optimum                  538.0
Rel error             2.973978
dtype: object

Min gap % for instance ch130.tsp:

Instance              ch130.tsp
q                           0.5
boost                     False
best integer len         6314.0
# iterations               217
AVG integer len     8660.843798
STD                  145.806526
Optimum                  6110.0
Rel error             3.338789
dtype: object

Min gap % for instance d198.tsp:

Instance              d198.tsp
q                           0.5
boost                     False
```

```
best integer len       16950.0
# iterations               54
AVG integer len      21978.785
STD                 393.788252
Optimum                15780.0
Rel error              7.414449
dtype: object
```