AI 프로그래밍 059분반

HW10 보고서

2020년 11월 24일

정보컴퓨터공학과

201824481 박지우

목차

코드가 점점 많아짐에 따라 모든 코드를 싣지 않고 저번 과제와 대비해 추가되거나 수정된 부분만 설명하였다.

1. Setup.py

```python
class Setup:
    def __init__(self):
        self.delta = 0.01
        self.updateRate = 0.01
        self.dx = 10 ** (-4)
        self.aType = 0

    def setVariables(self, parameters):
        self.delta = parameters['delta']
        self.updateRate = parameters['alpha']
        self.dx = parameters['dx']
        self.aType = parameters['aType']

    def getAType(self):
        return self.aType
```

알고리즘의 타입을 정의하는 aType변수를 추가하였다. 더해 setVariables로 Setup Class에 정의된 변수들을 parameter의 값으로 저장해주는 함수를 추가하였다. getAType은 이번 과제의 Main.py Skeleton 코드에서 사용되는 함수로, aType의 값을 반환한다.

2. Problem.py

2-1. Class Problem

```python
import random
from setup import Setup
import math


class Problem(Setup):
    def __init__(self):
        super().__init__()
        self.solution = 0
        self.minimum = 0
        self.NumEval = 0
        self.bestSolution = 0
        self.bestMinimum = 0
        self.avgMinimum = 0
        self.avgNumEval = 0
        self.sumNumEval = 0
        self.avgWhen = 0
```

```
def report(self):
    print("Best value: {0:,.5f}".format(self.bestMinimum))
    print()
    print("Total number of evaluations: {0:,}".format(self.sumNumEval))

def getSolution(self):
    return self.solution

def getNumEval(self):
    return self.NumEval

def getValue(self):
    return self.minimum

def storeExpResult(self, results):
    self.bestSolution = results[0]
    self.bestMinimum = results[1]
    self.avgMinimum = results[2]
    self.avgNumEval = results[3]
    self.sumNumEval = results[4]
    self.avgWhen = results[5]
```

이번 과제에서 제공된 Main.py의 Skeleton 코드에서 restart 알고리즘이 추가되며 bestSolution, bestMinimum, avgSolution, avgMinimum, avgNumEval, sumNumEval, avgWhen이 추가되었다. getSolution, getNumEval, getValue는 Main에서 쓰이는 함수로 solution, NumEval, minimum값을 반환한다. storeExpResult 역시 Main에서 쓰이는 함수로 정해진 restart 값만큼 프로그램을 실행하고 나서 Main함수에서 계산한 각 변수의 값을 전달받아 저장한다.

2-2. Class Numeric

```
def coordinate(self, solution):
    c = [round(value, 3) for value in solution]
    return tuple(c)

def report(self):
    print()
    print("Average objective value: {0:,.3f}".format(self.avgMinimum))
    print("Average number of evaluations: {0:,}".format(self.avgNumEval))
    if self.avgWhen != 0:
        print("Average iteration when the best solution first appears: {0:,}".format(self.avgWhen))
    print()
    print("Best solution found:")
    print(self.coordinate(self.bestSolution))
    Problem.report(self)
```

출력에 avgMinimum, avgNumEval, avgWhen, bestSolution을 출력하는 코드를 추가하였다. 또한 coordinate 함수가 전과 달리 bestSolution을 출력하므로 self.solution을 출력하

는 대신 인자를 받아 출력하는 것으로 수정하였다.

2-3. Class Tsp

```
def tenPerRow(self, solution):
    for i in range(len(solution)):
        print("{0:>5}".format(solution[i]), end='')
        if i % 10 == 9:
            print()
```

```
def report(self):
    print()
    print("Average objective value: {0:,.3f}".format(self.avgMinimum))
    print("Average number of evaluations: {0:,}".format(self.avgNumEval))
    if self.avgWhen != 0:
        print("Average iteration when the best solution first appears: {0:,}".format(self.avgWhen))
    print()
    print("Best solution found:")
    self.tenPerRow(self.bestSolution)
    Problem.report(self)
```

Tsp도 위의 Numeric Class의 수정과 유사하다.

3. Optimizer.py

3-1. Class Optimizer

```
class Optimizer(Setup):
    def __init__(self):
        super().__init__()
        self.pType = 0
        self.numExp = 0

    def setVariables(self, parameters):
        self.pType = parameters['pType']
        self.numExp = parameters['numExp']
        Setup.setVariables(self, parameters)

    def getNumExp(self):
        return self.numExp

    def displayNumExp(self):
        print()
        print("Number of experiments:", self.numExp)
```

HillClimbing Class와 새로 추가된 MetaHeuristics Class에서 공통으로 사용하는 변수와 함수를 모아둔 Class이다. 기존에는 HillClimbing Class가 바로 Setup Class를 Super Class로 두었는데 이번에는 Optimizer Class를 거치게 된다. Optimizer에 문제 Type을 저장하는 pType, numExp을 정의하고 두 변수의 값을 parameter 값으로 저장하는 setVariables함수와 Main에서 사용되는 getNumExp, displayNumExp 함수를 정의하였다.

3-2. Class HillClimbing

```
class HillClimbing(Optimizer):
    def __init__(self):
        super().__init__()
        self.limitStuck = 100
        self.numRestart = 0

    def displaySetting(self):
        if self.pType == 1:
            print()
            print("Number of random restarts:", self.numRestart)
            print()
            print("Mutation step size:", self.delta)
        if self.aType == 2 or self.aType == 3:
            print("Max evaluations with no improvement: {0:,} iterations".format(self.limitStuck))
```

```
def randomRestart(self, p):
    minimum = 0
    solution = 0
    for i in range(self.numRestart):
        self.run(p)
        if p.getValue() < minimum or i == 0:
            minimum = p.getValue()
            solution = p.getSolution()
    p.saveResult(solution, minimum)

def setVariables(self, parameters):
    self.limitStuck = parameters['limitStuck']
    self.numRestart = parameters['numRestart']
    Optimizer.setVariables(self, parameters)
```

새로 numRestart 변수가 추가되었다. setVariables 함수로 limitStuck과 numRestart 변수 값을 저장하며 displaySetting으로 numRestart 또한 출력한다. randomRestart는 numRestart 수만큼 알고리즘이 수행된다. 각 알고리즘이 끝날 때마다 solution과 minimum을 저장하여 main함수에서 bestSolution, bestMinimum, avgMinimum 등을 계산하게 된다.

3-3. Class FirstChoice

```
class FirstChoice(HillClimbing):
    def run(self, p):
        file = open('FirstChoiceTSP100.txt', 'w')
        current = p.randomInit()
        valueC = p.evaluate(current)
        file.writelines(str(valueC) + '\n')
        i = 0
        while i < self.limitStuck:
            successor = p.randomMutant(current)
            valueS = p.evaluate(successor)
            file.writelines(str(valueS) + '\n')
            if valueS < valueC:
                current = successor
                valueC = valueS
                i = 0
            else:
                i += 1
        file.close()
        p.saveResult(current, valueC)
```

Plot.py를 통해 사용할 FirstChoiceTSP100.txt 파일을 열어 evaluate된 값을 모두 저장하는 코드를 추가하였다. 모드를 w로 하였으므로 restart 횟수 등을 무시하고 마지막 1번의 run에 대해서만 저장한다.

3-4. Class Stochastic

```
class Stochastic(HillClimbing):
    def run(self, p):
        current = p.randomInit()
        valueC = p.evaluate(current)
        i = 0
        while i < self.limitStuck:
            neighbors = p.mutants(current)
            successor, valueS = self.stochasticBest(neighbors, p)
            if valueS < valueC:
                current = successor
                valueC = valueS
                i = 0
            else:
                i += 1
        p.saveResult(current, valueC)
```

```
    def displaySetting(self):
        print()
        print("Search algorithm: Stochastic Hill Climbing")
        HillClimbing.displaySetting(self)
```

```
def stochasticBest(self, neighbors, p):
    # Smaller valuse are better in the following list
    valuesForMin = [p.evaluate(indiv) for indiv in neighbors]
    largeValue = max(valuesForMin) + 1
    valuesForMax = [largeValue - val for val in valuesForMin]
    # Now, larger values are better
    total = sum(valuesForMax)
    randValue = random.uniform(0, total)
    s = valuesForMax[0]
    for i in range(len(valuesForMax)):
        if randValue <= s: # The one with index i is chosen
            break
        else:
            s += valuesForMax[i+1]
    return neighbors[i], valuesForMin[i]
```

새로 추가된 알고리즘이다. 과제에서 주어진 stochasticBest 함수를 추가하였고, 다른 알고리즘과 동일하게 displaySetting 함수를 추가하였다. Run은 FirstChoice의 run 함수를 참고하였다. stochasticBest 함수는 현재보다 더 좋은 value를 갖는 근처 neighbors 중 value값에 따라 확률적으로 선택한 neighbor과 value를 반환한다.

3-5. Class MetaHeuristics

```
class MetaHeuristics(Optimizer):
    def __init__(self):
        super().__init__()
        self.limitEval = 0
        self.whenBestFound = 0
        self._numSample = 100

    def setVariables(self, parameters):
        self.limitEval = parameters['limitEval']
        Optimizer.setVariables(self, parameters)

    def displaySetting(self):
        print()
        print("Limit Evaluation:", self.limitEval)
        print("Number of Samples:", self._numSample)

    def getWhenBestFound(self):
        return self.whenBestFound
```

새로 추가된 Class이다. Loop를 도는 횟수를 제한하는 limitEval, 가장 좋은 값을 찾았을 때의 iteration을 저장하는 whenBestFound, 초기값을 설정해줄 때 sample 개수를 저장하는 _numSample 변수를 정의하였다. setVariables 함수로 변수의 값을 parameter에 따라 저장하며 displaySetting으로 limitEval과 _numSample을 출력한다. getWhenBestFound는 main에서 사용하는 함수로 whenBestFound 변수값을 반환한다.

3-6. Class SimulatedAnnealing

```
class SimulatedAnnealing(MetaHeuristics):
    def run(self, p):
        file = open('SimulatedAnnealingTSP100.txt', 'w')
        current = p.randomInit()
        valueC = p.evaluate(current)
        file.write(str(valueC) + '\n')
        T = self.initTemp(p)
        for i in range(self.limitEval):
            T = self.tSchedule(T)
            if T == 0:
                p.saveResult(current, valueC)
                break
            else:
                next = p.randomMutant(current)
                valueS = p.evaluate(next)
                file.write(str(valueS) + '\n')
                E = valueS - valueC
                if E < 0:
                    self.whenBestFound = i
                    current = next
                    valueC = p.evaluate(current)
                else:
                    if random.random() <= math.exp(-E / T):
                        current = next
                        valueC = p.evaluate(current)
        file.close()
        p.saveResult(current, valueC)
```

```
def tSchedule(self, t):
    return t * (1 - (1 / 10**4))

def initTemp(self, p):# To set initial acceptance probability to 0.5
    diffs = []
    for i in range(self._numSample):
        c0 = p.randomInit()     # A random point
        v0 = p.evaluate(c0)     # Its value
        c1 = p.randomMutant(c0)# A mutant
        v1 = p.evaluate(c1)     # Its value
        diffs.append(abs(v1 - v0))
    dE = sum(diffs) / self._numSample  # Average value difference
    t = dE / math.log(2)        # exp(-dE/t) = 0.5
    return t

def displaySetting(self):
    print()
    print("Search algorithm: Simulated Annealing Meta Heuristics")
    MetaHeuristics.displaySetting(self)
```

SimulatedAnnealing은 MetaHeuristics을 Super Class로 가진다. tSchedule과 initTemp는 주어진 코드를 참고하였으며 tSchedule은 현재 온도 T를 계산하는 함수이고 initTemp는 온도 T의 초기값을 설정해주는 함수이다. displaySetting으로 알고리즘을 출력한다. Run 함수에서는 plot.py에서 사용하기 위한 SimulatedAnnealingTSP100.txt 파일을 열어 evaluate된 값을 저장한다. 위의 FirstChoice와 같이 마지막 1번의 run 값만 저장하게 된다. Run의 구현은 강의자료 알고리즘을 참고하였다.

4. Main.py

```
def createProblem(parameters): ###
    # Create a problem instance (a class object) 'p' of the type as
    # specified by 'pType', set the class variables, and return 'p'.
    if parameters['pType'] == 1:
        p = Numeric()
        p.setVariables(parameters['pFileName'])
    else:
        p = Tsp()
        p.setVariables(parameters['pFileName'])
    return p
```

```
def createOptimizer(parameters): ###
    # Create an optimizer instance (a class object) 'alg' of the type
    # as specified by 'aType', set the class variables, and return 'alg'.
    if parameters['aType'] == 1:
        alg = SteepestAscent()
    elif parameters['aType'] == 2:
        alg = FirstChoice()
    elif parameters['aType'] == 3:
        alg = Stochastic()
    elif parameters['aType'] == 4:
        alg = GradientDescent()
    else:
        alg = SimulatedAnnealing()
    alg.setVariables(parameters)
    return alg
```

Main.py는 주어진 Skeleton 코드를 사용하였다. createProblem함수는 exp.txt파일에서 읽어온 parameters에 따라 변수 p를 생성한다. pType이 1이면 Numeric으로 생성하고 2이면 Tsp로 생성한다. 또한 setVariables 함수로 사용할 변수 값을 parameters 값으로 초기화한다. createOptimizer 함수는 parameters에 저장된 알고리즘의 종류를 저장하는 aType에 따라 알고리즘을 생성한다. 1이면 SteepestAscent, 2이면 FirstChoice, 3이면 Stochastic, 4이면 GradientDescent, 그 외이면 SimulatedAnnealing으로 생성하고 setVariables 함수로 사용할 변수 값을 parameters 값으로 초기화한다.

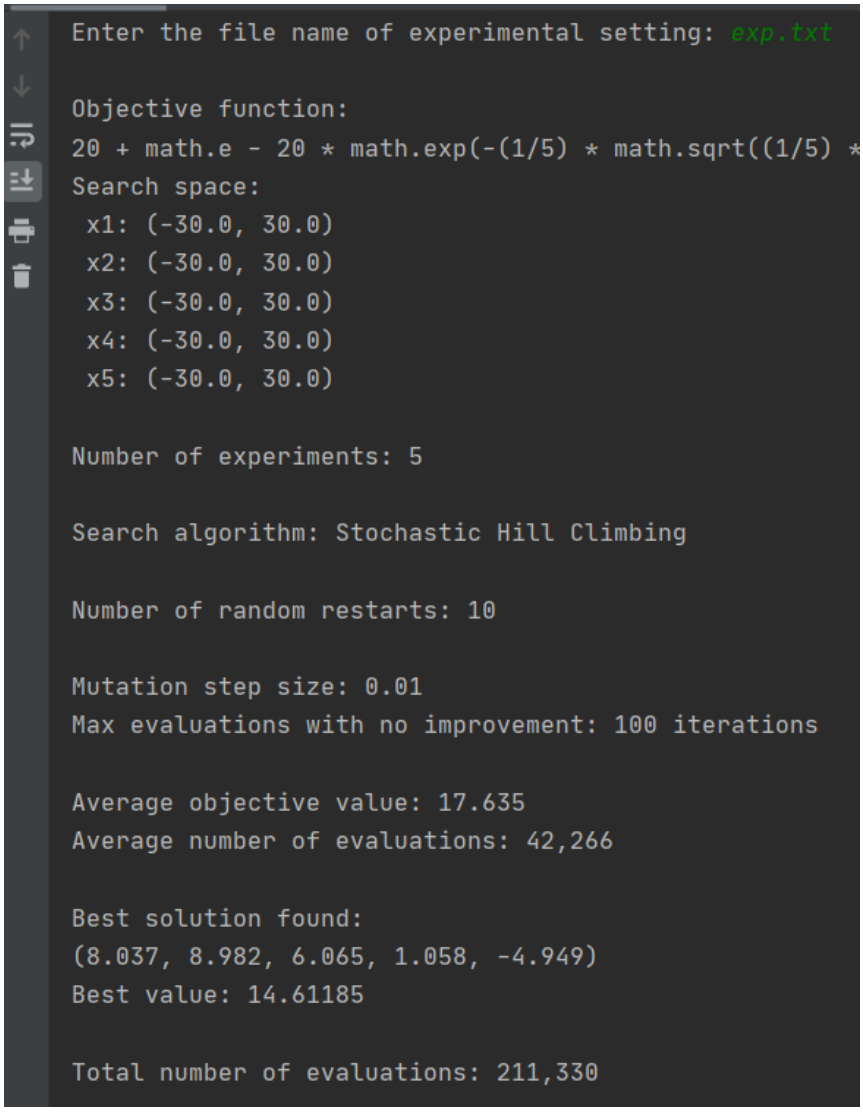
5. Plot.py

```
file1 = open('FirstChoiceTSP100.txt', 'r')
file2 = open('SimulatedAnnealingTSP100.txt', 'r')

y1 = []
numLine = 0
for line in file1:
    numLine += 1
    y1.append(float(line))
file1.close()
x1 = np.arange(numLine)

y2 = []
numLine = 0
for line in file2:
    numLine += 1
    y2.append(float(line))
file2.close()
x2 = np.arange(numLine)

plt.plot(x1, y1)
plt.plot(x2, y2)
plt.xlabel('Number of Evaluations')
plt.ylabel('Tour Cost')
plt.title('Search Performance (TSP-100)')

plt.legend(['First-Choice HC', 'Simulated Annealing'])
plt.show()
```

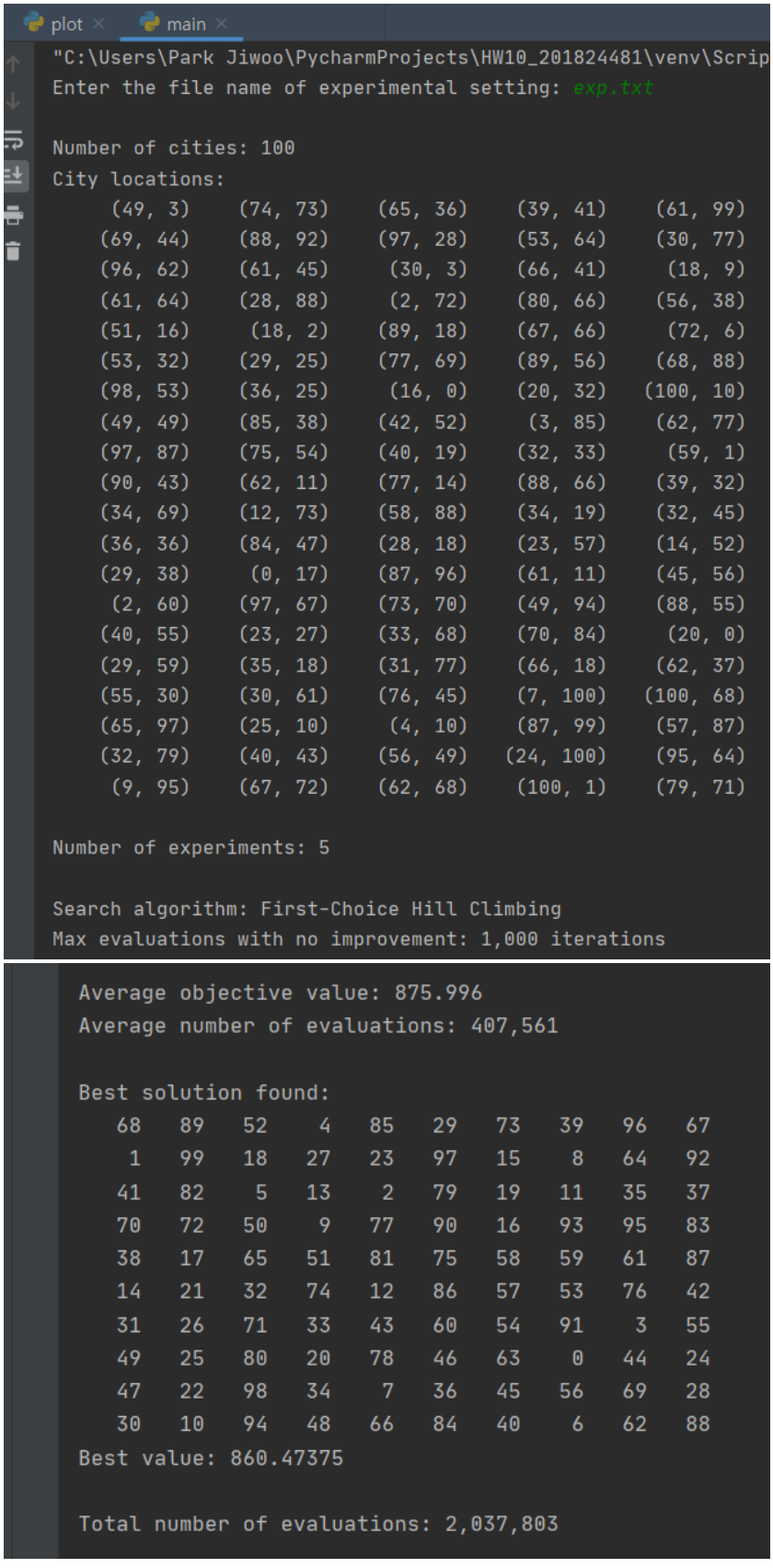
Plot.py는 강의자료 08 Numpy Matplotlib를 참고하였다. 단 x1과 x2의 길이가 달라질 수 있으므로 텍스트 파일에 저장된 값의 개수로 지정하였다.

6. 출력 결과

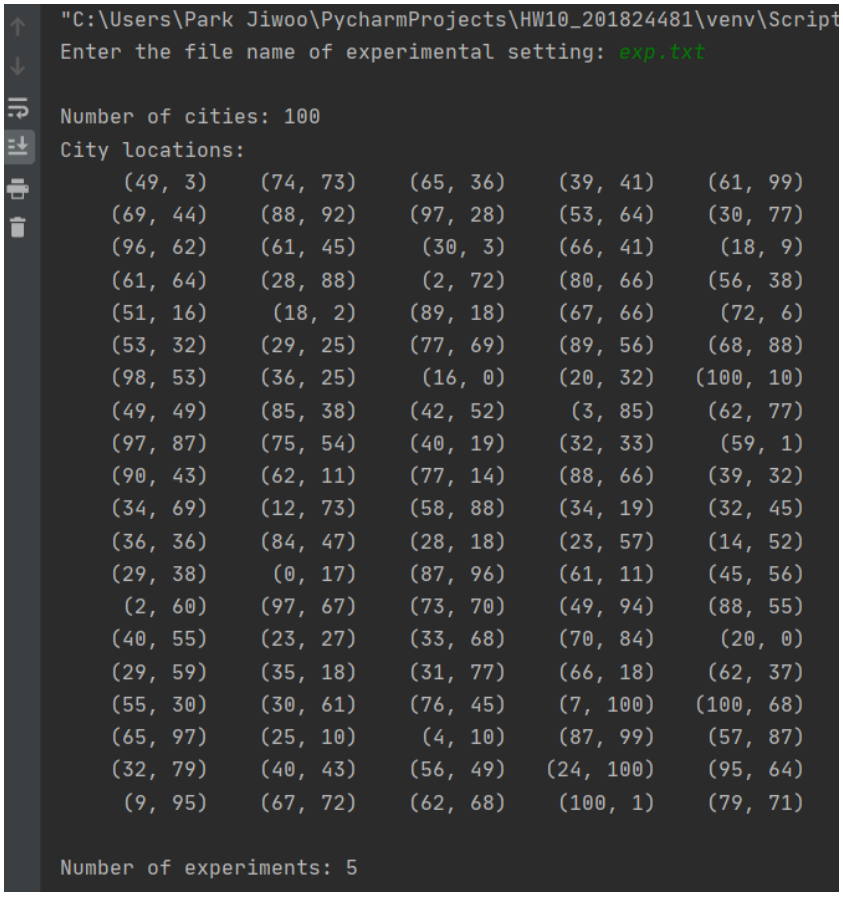
모든 알고리즘을 테스트하기엔 너무 양이 많아 새로운 알고리즘의 사진만 추가하였다.



Numeric 문제 중 Ackley.txt를 Stochastic 알고리즘으로 출력한 결과이다.



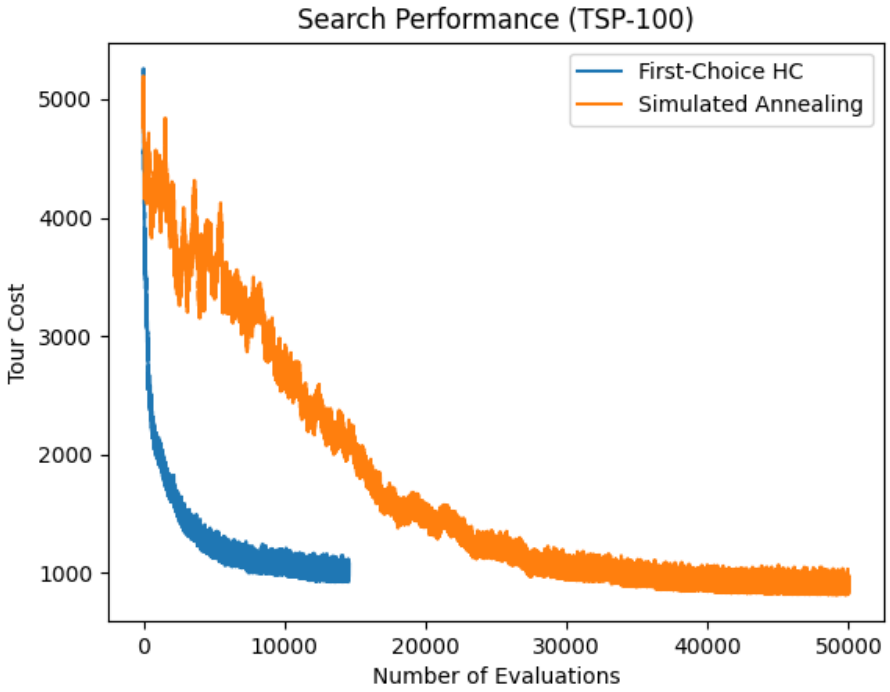
FirstChoice TSP100으로 설정해 텍스트 파일을 만들어주었다.



```
"C:\Users\Park Jiwoo\PycharmProjects\HW10_201824481\venv\Script
Enter the file name of experimental setting: exp.txt

Number of cities: 100
City locations:
     (49, 3)    (74, 73)    (65, 36)    (39, 41)    (61, 99)
    (69, 44)    (88, 92)    (97, 28)    (53, 64)    (30, 77)
    (96, 62)    (61, 45)     (30, 3)    (66, 41)     (18, 9)
    (61, 64)    (28, 88)     (2, 72)    (80, 66)    (56, 38)
    (51, 16)     (18, 2)    (89, 18)    (67, 66)     (72, 6)
    (53, 32)    (29, 25)    (77, 69)    (89, 56)    (68, 88)
    (98, 53)    (36, 25)     (16, 0)    (20, 32)   (100, 10)
    (49, 49)    (85, 38)    (42, 52)     (3, 85)    (62, 77)
    (97, 87)    (75, 54)    (40, 19)    (32, 33)     (59, 1)
    (90, 43)    (62, 11)    (77, 14)    (88, 66)    (39, 32)
    (34, 69)    (12, 73)    (58, 88)    (34, 19)    (32, 45)
    (36, 36)    (84, 47)    (28, 18)    (23, 57)    (14, 52)
    (29, 38)     (0, 17)    (87, 96)    (61, 11)    (45, 56)
     (2, 60)    (97, 67)    (73, 70)    (49, 94)    (88, 55)
    (40, 55)    (23, 27)    (33, 68)    (70, 84)     (20, 0)
    (29, 59)    (35, 18)    (31, 77)    (66, 18)    (62, 37)
    (55, 30)    (30, 61)    (76, 45)    (7, 100)   (100, 68)
    (65, 97)    (25, 10)     (4, 10)    (87, 99)    (57, 87)
    (32, 79)    (40, 43)    (56, 49)   (24, 100)    (95, 64)
     (9, 95)    (67, 72)    (62, 68)    (100, 1)    (79, 71)

Number of experiments: 5
```

```
Search algorithm: Simulated Annealing Meta Heuristics

Limit Evaluation: 50000
Number of Samples: 100

Average objective value: 849.465
Average number of evaluations: 170,680
Average iteration when the best solution first appears: 49,178

Best solution found:
   40    6   62   88   85    4   68   89   52   29
   73   39   96    1   99   27   67   23   97   15
    8   35   92   11   19   25   80   79    2   13
    5   82   41   18   48   66   84   94   10   30
   28   69   56   45   36    7   22   34   98   47
   24   78   63   46   44    0   20   42   31   49
   55   43   26   53   76   57   86   12   74   32
   21   14   87   61   71   33   60   54    3   91
   37   64   70   81   75   58   59   65   17   51
   38   95   83   93   16   90   77    9   50   72
Best value: 815.41844

Total number of evaluations: 853,398
```

SimulatedAnnealing TSP100으로 설정해 텍스트 파일을 만들어주었다.



위의 First-Choice와 Simulated Annealing을 통해 만들어진 텍스트 파일로 plot.py를 통해 그래프를 출력한 모습이다.