# Homework #6


제출 일자 : 2020-11-08
강의 : AI프로그래밍 (CP35655-060)
담당 교수님 : 감진규 교수님
학번 : 201824633
이름 : 김유진

1) problem.py

class Problem : 자식 클래스에서 오버라이딩 될 부분이므로, 함수의 내부 내용은 임의로 채워넣었다.

```python
import random
import math


class Problem:

    # Numeric 클래스와 Tsp 클래스에서 사용
    LIMIT_STUCK = 100
    NumEval = 0

    # 두 class 모두 solution 리스트와 minimum 값을 구하는 클래스이므로,
    # parent class 에서 정의함.
    def __init__(self, solution=[], minimum=0):
        self._solution = solution
        self._minimum = minimum

    # LIMIT_STUCK 값을 얻기 위함
    def getLIMIT_STUCK(self):
        return self.LIMIT_STUCK

    # 각각 solution 값과 minimum 값을 얻기 위함
    def setSolution(self, solution):
        self._solution = solution

    def setMin(self, minimum):
        self._minimum = minimum

    # 하위 6개의 함수 : Numeric 클래스와 Tsp 클래스에서 오버라이딩
    # 임의로 함수 내부 채워넣음.
    def setVariables(self):
        print("Problem Class")

    def describe(self):
        print()

    def report(self):
        print()

    def randomInit(self):
        return random.randint(1, 5)

    def evaluate(self, current):
        sum(current)

    def randomMutant(self, current):
        return random.randint(1, 5)
```

class Numeric

hw05에서 사용한 기존 (n)으로 끝나는 py 파일에 있는 함수들을 매개변수만 변경해서 복사 피피티에 명시되어있는 각각의 함수를 제외한 모든 함수들을 클래스로 넣어줬다.

```
49  class Numeric(Problem):
50      DELTA = 0.01
51
52      def __init__(self, p=[], expression=''):
53          self._p = p
54          self._expression = expression
55
56      def setVariables(self):
57          # file에서 읽어온 값을 클래스의 변수에 할당해주는 함수
58          fileName = input("Enter the file name of a function: ")
59          infile = open(fileName, 'r')
60          self._expression = infile.readline().rstrip()
61
62          varNames = list()
63          low = list()
64          up = list()
65
66          while True:
67              tmp = infile.readline().rstrip()
68              if not tmp:
69                  break
70              tmp = tmp.split(",")
71              varNames.append(tmp[0])
72              low.append(int(tmp[1]))
73              up.append(int(tmp[2]))
74
75          self._p = [varNames, low, up]
76          infile.close()
77
78      def describe(self):
79          # 파일로 읽어온 값을 print 해주는 함수
80          print()
81          print("Objective function:")
82          print(self._p[0])
83          print("Search space:")
84          varNames = self._p[0]
85          low = self._p[1]
86          up = self._p[2]
87          for i in range(len(low)):
88              print(" " + varNames[i] + ":", (low[i], up[i]))
```

```
90      def report(self):
91          # 결과값을 print 해주는 함수
92          print()
93          print("Solution found:")
94          print(tuple([round(value, 3) for value in self._solution]))
95          print("Minimum value: {0:,.3f}".format(self._minimum))
96          print()
97          print("Total number of evaluations: {0:,}".format(self.NumEval))
98
99      def randomInit(self):
100         low = self._p[1]
101         up = self._p[2]
102
103         init = [random.uniform(low[i], up[i]) for i in range(len(low))]
104         return init
105
106     def evaluate(self, current):
107         self.NumEval += 1
108         expr = self._expression
109         varNames = self._p[0]
110         for i in range(len(varNames)):
111             assignment = varNames[i] + '=' + str(current[i])
112             exec(assignment)
113         return eval(expr)
114
115     def getDELTA(self):
116         return self.DELTA
117
118     def randomMutant(self, current):
119         i = random.randint(0, len(current) - 1)
120         d = random.sample((-1 * self.DELTA, self.DELTA), 1)[0]
121         return self.mutate(current, i, d)
122
123     def mutants(self, current):
124         neighbors = list()
125         for i in range(len(current)):
126             neighbors.append(self.mutate(current, i, self.DELTA))
127             neighbors.append(self.mutate(current, i, -1 * self.DELTA))
128
129         return neighbors
130
131     def mutate(self, current, i, d):
132         curCopy = current[:]
133         l = self._p[1][i]
134         u = self._p[2][i]
135         if l <= (curCopy[i] + d) <= u:
136             curCopy[i] += d
137         return curCopy
```

# grediant Descent를 지원하기 위한 함수들이다.
createNextP가 현재 위치하고 있는 currentP에서 가장 최적화된 곳으로 이동하도록 유도해 주는 함수이다.

```
# 하위의 두개 함수가 grediantDescent를 지원하기 위한 함수들이다.
def createNextP(self, currentP):
    g = self.getGrediant(currentP)      # step 1 get gradient
    resultV = self.evaluate(currentP)   # 만약, 아래에서 구하는 모든 점들이 조건에 부합하지 않은 경우,
    resultP = currentP[:]               # currentP를 리턴해 gradientDescent 함수의 작동을 멈추기 위함
    for i in range(len(currentP)):
        tmpP = currentP[:]
        tmpP[i] -= (self.DELTA * g[i])      # step 2 get nextP,
                                            # it should be in domain
        low = self._p[1]
        up = self._p[2]

        if low[i] <= tmpP[i] <= up[i]:
            valueN = self.evaluate(tmpP)
            if valueN < resultV:
                resultV = valueN
                resultP = tmpP[:]
    return resultP, resultV
    # value가 가장 작은 점의 정보와 해당 value를 리턴해준다.
    # 이 때 value는 evaluate 연산을 더이상 하지 않기 위함.

# 각각의 x[k]에 대해서 lerning rate 만큼 이동한 뒤 (DELTA값 사용)
# currentP 와 새롭게 생성된 newP의 기울기를 구해서 value에 넣어줬다.
def getGrediant(self, currentP):
    value = []
    valueP = self.evaluate(currentP)
    for i in range(len(currentP)):
        newP = self.mutate(currentP, i, self.DELTA)
        value.append((self.evaluate(newP)-valueP)/self.DELTA)
    return value
```

Tsp class 또한 위의 Numeric 클래스와 똑같은 방식으로 복사해서 넣어줬다.

```
162  class Tsp(Problem):
163      def __init__(self, numCities=0, locations=[], table=[]):
164          self._numCities = numCities
165          self._locations = locations
166          self._table = table
167
168      def setVariables(self):
169          fileName = input("Enter the file name of a TSP: ")
170          infile = open(fileName, 'r')
171
172          self._numCities = int(infile.readline())
173          self._locations = []
174          line = infile.readline()
175          while line != '':
176              self._locations.append(eval(line))
177              line = infile.readline()
178          infile.close()
179          self._table = self.calcDistanceTable()
180
181      def calcDistanceTable(self):
182          table = [[0] * self._numCities for i in range(self._numCities)]
183          for i in range(self._numCities):
184              for j in range(i + 1, self._numCities):
185                  table[i][j] = math.sqrt(math.pow(self._locations[i][0] - self._locations[j][0], 2)
186                                          + math.pow(self._locations[i][1]-self._locations[j][1], 2))
187                  table[j][i] = table[i][j]
188          return table
189
190      def describe(self):
191          print()
192          print("Number of cities:", self._numCities)
193          print("City locations:")
194          for i in range(self._numCities):
195              print("{0:>12}".format(str(self._locations[i])), end='')
196              if i % 5 == 4:
197                  print()
198
199      def report(self):
200          print()
201          print("Best order of visits:")
202          for i in range(len(self._solution)):
203              print("{0:>5}".format(self._solution[i]), end='')
204              if i % 10 == 9:
205                  print()
206          print("Minimum tour cost: {0:,}".format(round(self._minimum)))
207          print()
208          print("Total number of evaluations: {0:,}".format(self.NumEval))
```

```
210     def randomInit(self):
211         n = self._numCities
212         init = list(range(n))
213         random.shuffle(init)
214         return init
215 
216     def evaluate(self, current):
217         self.NumEval += 1
218 
219         numCites = self._numCities
220         distanceTable = self._table
221 
222         cost = 0
223         for i in range(numCites - 1):
224             cost += distanceTable[current[i]][current[i+1]]
225         cost += distanceTable[numCites-1][current[0]]
226 
227         return cost
228 
229     def randomMutant(self, current):
230         while True:
231             i, j = sorted([random.randrange(self._numCities)
232                            for _ in range(2)])
233             if i < j:
234                 curCopy = self.inversion(current, i, j)
235                 break
236         return curCopy
237 
238     def inversion(self, current, i, j):
239         curCopy = current[:]
240         while i < j:
241             curCopy[i], curCopy[j] = curCopy[j], curCopy[i]
242             i += 1
243             j -= 1
244         return curCopy
245 
246     def mutants(self, current):
247         n = self._numCities
248         neighbors = []
249         count = 0
250         triedPairs = []
251         while count <= n:
252             i, j = sorted([random.randrange(n) for _ in range(2)])
253             if i < j and [i, j] not in triedPairs:
254                 triedPairs.append([i, j])
255                 curCopy = self.inversion(current, i, j)
256                 count += 1
257                 neighbors.append(curCopy)
258         return neighbors
```

2) first choice(n).py

```python
from problem import Numeric

def main():
    p = Numeric()
    p.setVariables()
    firstChoice(p)
    p.describe()
    displaySetting(p)
    p.report()

def firstChoice(p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    i = 0
    while i < p.getLIMIT_STUCK():
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    p.setSolution(current)
    p.setMin(valueC)

def displaySetting(p):
    print()
    print("Search algorithm: First-Choice Hill Climbing")
    print()
    print("Mutation step size:", p.getDELTA())

main()
```

3) steepest ascent (n).py

```python
from problem import Numeric

def main():
    p = Numeric()
    p.setVariables()
    steepestAscent(p)
    p.describe()
    displaySetting(p)
    p.report()

def steepestAscent(p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    p.mutants(current)
    while True:
        neighbors = p.mutants(current)
        successor, valueS = bestOf(neighbors, p)
        if valueS >= valueC:
            break
        else:
            current = successor
            valueC = valueS
    p.setSolution(current)
    p.setMin(valueC)

def bestOf(neighbors, p):
    best = neighbors[0]
    bestValue = p.evaluate(neighbors[0])
    for i in range(1, 10):
        tmpValue = p.evaluate(neighbors[i])
        if bestValue > tmpValue:
            bestValue = tmpValue
            best = neighbors[i]
    return best, bestValue

def displaySetting(p):
    print()
    print("Search algorithm: Steepest-Ascent Hill Climbing")
    print()
    print("Mutation step size:", p.getDELTA())

main()
```

4) first choice(tsp).py

```python
from problem import Tsp

def main():
    p = Tsp()
    p.setVariables()
    firstChoice(p)
    p.describe()
    displaySetting()
    p.report()

def firstChoice(p):
    current = p.randomInit()
    valueC  = p.evaluate(current)
    i = 0
    while i < p.getLIMIT_STUCK():
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    p.setSolution(current)
    p.setMin(valueC)

def displaySetting():
    print()
    print("Search algorithm: First-Choice Hill Climbing")

main()
```

5) steepest ascent (tsp).py

```python
from problem import Tsp

def main():
    p = Tsp()
    p.setVariables()
    steepestAscent(p)
    p.describe()
    displaySetting()
    p.report()

def steepestAscent(p):
    current = p.randomInit()   # 'current' is a list of city ids
    valueC = p.evaluate(current)
    while True:
        neighbors = p.mutants(current)
        (successor, valueS) = bestOf(neighbors, p)
        if valueS >= valueC:
            break
        else:
            current = successor
            valueC = valueS
    p.setSolution(current)
    p.setMin(valueC)

def bestOf(neighbors, p):
    best = neighbors[0]
    bestValue = p.evaluate(neighbors[0])

    for i in range(1, len(neighbors)):
        newValue = p.evaluate(neighbors[i])
        if newValue < bestValue:
            best = neighbors[i]
            bestValue =  newValue
    return best, bestValue

def displaySetting():
    print()
    print("Search algorithm: Steepest-Ascent Hill Climbing")

main()
```

6) grediant descent.py

```python
from problem import Numeric

def main():
    p = Numeric()
    p.setVariables()
    gradientDescent(p)
    p.describe()
    displaySetting(p)
    p.report()

def gradientDescent(p):
    currentP = p.randomInit()       # 임의의 시작 점 선택
    valueC = p.evaluate(currentP)   # 시작 점 value 구함
    while True:
        nextP, valueN = p.createNextP(currentP)
        if valueN >= valueC:        # step 3 update
            break
        else:
            currentP = nextP
            valueC = valueN
    p.setSolution(currentP)
    p.setMin(valueC)

def displaySetting(p):
    print()
    print("Search algorithm: Gradient Descent")
    print()
    print("Learning Rate:", p.getDELTA())

main()
```

✓ 결과값

**first-choice**

1) Convex.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Convex.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: First-Choice Hill Climbing

Mutation step size: 0.01

Solution found:
(1.998, 5.005, -7.995, -1.002, 7.0)
Minimum value: 0.000

Total number of evaluations: 12,377
```

2) Ackley.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Ackley.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: First-Choice Hill Climbing

Mutation step size: 0.01

Solution found:
(-29.998, -1.001, -15.003, 17.998, -0.001)
Minimum value: 19.337

Total number of evaluations: 597
```

3) Griewank.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Griewank.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: First-Choice Hill Climbing

Mutation step size: 0.01

Solution found:
(3.145, 4.436, 21.738, -6.275, -7.011)
Minimum value: 0.148

Total number of evaluations: 2,626
```

**steepest-ascent**

1) Convex.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Convex.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Steepest-Ascent Hill Climbing

Mutation step size: 0.01

Solution found:
(1.999, 5.002, -8.002, -1.004, 6.998)
Minimum value: 0.000

Total number of evaluations: 96,041
```

2) Ackley.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Ackley.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Steepest-Ascent Hill Climbing

Mutation step size: 0.01

Solution found:
(1.996, 26.999, -11.001, 25.997, -21.004)
Minimum value: 19.623

Total number of evaluations: 1,601
```

3) Griewank.txt

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Griewank.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Steepest-Ascent Hill Climbing

Mutation step size: 0.01

Solution found:
(18.843, 0.001, 0.003, -12.537, -0.003)
Minimum value: 0.128

Total number of evaluations: 4,721
```

first-choice (tsp)

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\tsp30.txt

Number of cities: 30
City locations:
     (8, 31)    (54, 97)    (50, 50)    (65, 16)    (70, 47)
   (25, 100)    (55, 74)    (77, 87)     (6, 46)    (70, 78)
    (13, 38)   (100, 32)    (26, 35)    (55, 16)    (26, 77)
    (17, 67)    (40, 36)    (38, 27)     (33, 2)     (48, 9)
    (62, 20)    (17, 92)     (30, 2)    (80, 75)    (32, 36)
    (43, 79)    (57, 49)    (18, 24)    (96, 76)    (81, 39)

Search algorithm: First-Choice Hill Climbing

Best order of visits:
   29   11   28   23    7    9    6   26    4   20
    3   13   19    2   16   17   27   22   18   24
   12   10    0    8   15   14   25    1    5   21
Minimum tour cost: 520

Total number of evaluations: 668
```

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\tsp50.txt

Number of cities: 50
City locations:
      (1, 7)    (14, 92)    (45, 97)    (17, 60)    (22, 44)
     (4, 38)    (13, 73)    (79, 68)    (76, 95)    (62, 14)
    (25, 75)     (26, 9)    (88, 81)    (56, 65)    (64, 71)
    (92, 20)     (7, 20)     (8, 20)    (61, 39)    (17, 11)
    (10, 40)    (18, 72)    (89, 72)    (58, 25)    (57, 57)
    (66, 70)    (36, 72)    (89, 91)    (18, 90)    (72, 49)
    (82, 38)    (22, 26)    (36, 56)    (23, 44)    (45, 45)
     (7, 27)     (84, 6)    (32, 78)     (0, 29)    (64, 63)
    (45, 24)    (21, 81)    (37, 16)    (86, 57)    (65, 99)
    (25, 53)    (98, 24)    (83, 81)     (50, 5)    (58, 80)

Search algorithm: First-Choice Hill Climbing

Best order of visits:
   49   14   25   39   24   29   18    4   45   33
   20    5   35   38   16   19   11   31   17    0
   42   40   48   23    9   36   15   46   30   43
    7   22   12   47   27    8   44    2   26   37
   10    1   28   41   21    3    6   32   34   13
Minimum tour cost: 700

Total number of evaluations: 1,483
```

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\제출\tsp100.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)

Search algorithm: First-Choice Hill Climbing

Best order of visits:
   97   67   23   41   56   19   35    5   82   45
    7   34   22   36   24   46   78   47   13   11
   92    8   15   18   30   10   99   29    4   68
   16   83   38   72   81   75   54   91    3   26
   42   80   79   37   64   70   60   55   49   25
    2   98   44    0   63   20   33   87   14   86
   21   32   57   31   53   43   76   12   74   61
   71   58   50   90   77    9   95   17   51   59
   65   93   52   89   39   96   27    1   88   85
   62    6   73   48   94   28   69   40   66   84
Minimum tour cost: 1,586

Total number of evaluations: 2,295
```

steepest-ascent(tsp)

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\제출\tsp30.txt

Number of cities: 30
City locations:
     (8, 31)    (54, 97)    (50, 50)    (65, 16)    (70, 47)
   (25, 100)    (55, 74)    (77, 87)     (6, 46)    (70, 78)
    (13, 38)   (100, 32)    (26, 35)    (55, 16)    (26, 77)
    (17, 67)    (40, 36)    (38, 27)     (33, 2)     (48, 9)
    (62, 20)    (17, 92)     (30, 2)    (80, 75)    (32, 36)
    (43, 79)    (57, 49)    (18, 24)    (96, 76)    (81, 39)

Search algorithm: Steepest-Ascent Hill Climbing

Best order of visits:
   11   28    7   23    4   29   20   17   12    0
   10    8   15   14    5   21   25    6    1    9
   26    2    3   13   19   18   22   27   24   16
Minimum tour cost: 574

Total number of evaluations: 776
```

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\제출\tsp50.txt

Number of cities: 50
City locations:
      (1, 7)    (14, 92)    (45, 97)    (17, 60)    (22, 44)
     (4, 38)    (13, 73)    (79, 68)    (76, 95)    (62, 14)
    (25, 75)     (26, 9)    (88, 81)    (56, 65)    (64, 71)
    (92, 20)     (7, 20)     (8, 20)    (61, 39)    (17, 11)
    (10, 40)    (18, 72)    (89, 72)    (58, 25)    (57, 57)
    (66, 70)    (36, 72)    (89, 91)    (18, 90)    (72, 49)
    (82, 38)    (22, 26)    (36, 56)    (23, 44)    (45, 45)
     (7, 27)     (84, 6)    (32, 78)     (0, 29)    (64, 63)
    (45, 24)    (21, 81)    (37, 16)    (86, 57)    (65, 99)
    (25, 53)    (98, 24)    (83, 81)     (50, 5)    (58, 80)

Search algorithm: Steepest-Ascent Hill Climbing

Best order of visits:
   49   14    7   44    8   47   43   22   12   27
   25   39   24   33    4   45    3   21    6    1
   41   10   37    2   28   26   20   38   35   17
    0   11   19   16    5   31   42   48    9   23
   34   40   32   13   29   30   46   15   36   18
Minimum tour cost: 849

Total number of evaluations: 2,143
```

```
Enter the file name of a TSP: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\제출\tsp100.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)

Search algorithm: Steepest-Ascent Hill Climbing

Best order of visits:
   27   23   69   94   66   84   40   30   48   28
   41   45    7   56   39   85   88   62    6   73
   52   97   96   70   35   64   54    9   75   81
   72   58   51   65   59   38   16   17   37    8
   15   67   10   18   99    1   29   89    4   68
   93   95   83   77   90   50   60   43   71   57
   14   87   32   74   61   21   86   12   76   26
   53   33    0   20   46   34   22   36   82   47
   63   44   24   78    3   91   49   55   31   42
   25    2   92   13    5   11   79   80   19   98
Minimum tour cost: 1,604

Total number of evaluations: 6,768
```

gradient descent.py

```
Enter the file name of a function: C:\Users\chris\OneDrive\바탕 화면\pnu\ai\hw05\sample problems\Ackley.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Gradient Descent

Learning Rate: 0.01

Solution found:
(-13.004, 9.994, -20.004, 13.0, 27.999)
Minimum value: 19.455

Total number of evaluations: 5,233
```

```
Enter the file name of a function: hw05\sample problems\Convex.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Gradient Descent

Learning Rate: 0.01

Solution found:
(2.0, 5.0, -8.0, -1.005, 7.0)
Minimum value: 0.000

Total number of evaluations: 13,477
```

```
Enter the file name of a function: hw05\sample problems\Griewank.txt

Objective function:
['x1', 'x2', 'x3', 'x4', 'x5']
Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Gradient Descent

Learning Rate: 0.01

Solution found:
(15.7, -13.315, -16.3, 18.812, -0.005)
Minimum value: 0.261

Total number of evaluations: 219,253
```