AI 프로그래밍 059분반

HW8 보고서

2020년 11월 11일

정보컴퓨터공학과

201824481 박지우

목차

1. Defining Classes

1-1. N (First-choice / Steepest-ascent)

1-1-1. Problem.py

```python
import random
import math


class Problem:
    def __init__(self):
        self.solution = None
        self.minimum = None
        self.NumEval = 0
        self.DELTA = 0.01
        self.LIMIT_STUCK = 100

    def saveResult(self, solution, minimum):
        self.solution = solution
        self.minimum = minimum
```

지난 과제에서 사용하였던 random, math를 사용하기 위해 import한다. 또 과제에서 제시된 대로 Problem class를 선언하였다. __init__함수로 Problem의 Class에 정의된 solution, minimum, NumEval, DELTA, LIMIT_STUCK을 초기화한다. Solution, minimum, NumEval은 Numeric과 TSP 모두에서 사용되는 변수이기 때문에 각 Class가 아닌 Problem Class에 정의하고, DELTA와 LIMIT_STUCK은 define된 변수이지만 역시 두 문제에서 모두 사용되므로 Problem에 정의한다. saveResult함수는 각 python 파일에 solution, minimum으로 변수를 정의하여 기존 함수의 결과값을 저장하는 것을 대신해 Problem 변수에 저장할 수 있도록 한다.

```
class Numeric(Problem):
    def __init__(self):
        super().__init__()
        self.domain = None
        self.expression = None
        self.updateRate = 0.0000001

    def createProblem(self):
        varNames = []
        low = []
        up = []
        filename = str(input("Enter the file name of a function: "))
        infile = open(filename, 'r')
        lines = infile.readlines()
        infile.close()

        for i in range(len(lines)):
            lines[i] = lines[i].rstrip()
        self.expression = lines[0]

        for i in range(1, len(lines)):
            data = lines[i].split(',')
            varNames.append(data[0])
            low.append(float(data[1]))
            up.append(float(data[2]))
        self.domain = [varNames, low, up]
```

```
def randomInit(self):
    init = []
    for i in range(0, 5):
        data = random.uniform(self.domain[1][i], self.domain[2][i])
        init.append(data)
    return init

def evaluate(self, current):
    self.NumEval += 1
    expr = self.expression
    varNames = self.domain[0]
    for i in range(len(varNames)):
        assignment = varNames[i] + '=' + str(current[i])
        exec(assignment)
    return eval(expr)

def mutate(self, current, i):
    curCopy = current[:]
    l = self.domain[1][i]
    u = self.domain[2][i]
    if l <= (curCopy[i] + self.DELTA) <= u:
        curCopy[i] += self.DELTA
    return curCopy
```

```
def describeProblem(self):
    print()
    print("Objective function:")
    print(self.expression)
    print("Search space:")
    varNames = self.domain[0]
    low = self.domain[1]
    up = self.domain[2]
    for i in range(len(low)):
        print(" " + varNames[i] + ":", (low[i], up[i]))

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

def displayResult(self):
    print()
    print("Solution found:")
    print(self.coordinate())
    print("Minimum value: {0:,.3f}".format(self.minimum))
    print()
    print("Total number of evaluations: {0:,}".format(self.NumEval))

def randomMutant(self, current):
    i = random.randint(0, 4)
    return self.mutate(current, i)
```

```
def mutants(self, current):
    neighbors = []
    for i in range(0, 5):
        neighbors.append(self.mutate(current, i))
    return neighbors

def bestOf(self, neighbors):
    for i in range(len(neighbors)):
        value = self.evaluate(neighbors[i])
        if i == 0:
            bestValue = value
            best = neighbors[i]
        elif value < bestValue:
            bestValue = value
            best = neighbors[i]
    return best, bestValue
```

Problem Class를 Super Class로 하여 Numeric Class를 정의한다. Problem의 __init__과 함께 Problem Class의 변수인 domain, expression, updateRate를 정의한다. Domain과 expression 변수는 기존 p의 tuple로 정의된 값이었다. updateRate는 뒤의 Gradient-descent에 사용되는 값이다. Numeric문제에서 First-choice와 steepest-ascent방식 모두 사용하는 함수는 createProblem, randomInit, evaluate, mutate, describeProblem, coordinate, displayResult로 Numeric Class의 함수로 선언한다. 또, randomMutant, mutants, bestOf함수는 각 상황에 맞추어 사용되지만 encapsulation을 위해 Numeric Class 내부에 선언한다. 그후 기존 domain과 expression을 저장하던 변수 p를 함수 인자로 주는 대신 모두 self.domain과 self.expression으로 사용한다.

1-1-2. First-choice (n)

```
import problem


def main():
    p = problem.Numeric()
    p.createProblem()
    firstChoice(p)
    p.describeProblem()
    displaySetting(p)
    p.displayResult()
```

```
def firstChoice(p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    i = 0
    while i < p.LIMIT_STUCK:
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    p.saveResult(current, valueC)


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


main()
```

저번 과제 코드와 크게 다르지 않지만, problem.py에 정의된 class를 사용하기 위해 import한다. 또, class 내부에 선언된 함수를 사용하기 위해서 모두 p.(함수이름)(함수인자)로 바꾼다.

1-1-3. Steepest-ascent (n)

```
import problem


def main():
    p = problem.Numeric()
    p.createProblem()
    steepestAscent(p)
    p.describeProblem()
    displaySetting(p)
    p.displayResult()
```

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


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


main()
```

Steepest-ascent 파일도 위와 유사하다. Numeric class를 사용하기 위해 problem.py를 import한 후 class 내부로 선언된 함수를 사용할 때 변수 이름을 붙여준다.

1-2. TSP (First-choice / Steepest-ascent)

1-2-1. problem.py

```
class Tsp(Problem):
    def __init__(self):
        super().__init__()
        self.numCities = 0
        self.locations = []
        self.table = []

    def createProblem(self):
        fileName = input("Enter the file name of a TSP: ")
        infile = open(fileName, 'r')
        self.numCities = int(infile.readline())
        line = infile.readline()
        while line != '':
            self.locations.append(eval(line))
            line = infile.readline()
        infile.close()
        self.calcDistanceTable()

    def calcDistanceTable(self):
        for i in range(self.numCities):
            data = []
            for j in range(self.numCities):
                data.append(math.sqrt(math.pow((self.locations[i][0] - self.locations[j][0]), 2)
                                      + math.pow((self.locations[i][1] - self.locations[j][1]), 2)))
            self.table.append(data)
```

```
def randomInit(self):
    init = list(range(self.numCities))
    random.shuffle(init)
    return init

def evaluate(self, current):
    self.NumEval += 1
    cost = 0
    for i in range(self.numCities - 1):
        a = current.index(i)
        b = current.index(i + 1)
        cost += self.table[a][b]
    return cost

def inversion(self, current, i, j):
    curCopy = current[:]
    while i < j:
        curCopy[i], curCopy[j] = curCopy[j], curCopy[i]
        i += 1
        j -= 1
    return curCopy

def describeProblem(self):
    print()
    print("Number of cities:", self.numCities)
    print("City locations:")
    for i in range(self.numCities):
        print("{0:>12}".format(str(self.locations[i])), end='')
        if i % 5 == 4:
            print()
```

```
def displayResult(self):
    print()
    print("Best order of visits:")
    self.tenPerRow()
    print("Minimum tour cost: {0:,}".format(round(self.minimum)))
    print()
    print("Total number of evaluations: {0:,}".format(self.NumEval))

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

def randomMutant(self, current):
    while True:
        i, j = sorted([random.randrange(self.numCities)
                       for _ in range(2)])
        if i < j:
            curCopy = self.inversion(current, i, j)
            break
    return curCopy
```

```
def mutants(self, current):
    neighbors = []
    count = 0
    triedPairs = []
    while count <= self.numCities:
        i, j = sorted([random.randrange(self.numCities) for _ in range(2)])
        if i < j and [i, j] not in triedPairs:
            triedPairs.append([i, j])
            curCopy = self.inversion(current, i, j)
            count += 1
            neighbors.append(curCopy)
    return neighbors

def bestOf(self, neighbors):
    best = []
    bestValue = 0
    for i in range(len(neighbors)):
        value = self.evaluate(neighbors[i])
        if i == 0:
            bestValue = value
            best = neighbors[i]
        elif value < bestValue:
            bestValue = value
            best = neighbors[i]
    return best, bestValue
```

Problem Class를 Super Class로 하여 Tsp Class를 정의한다. Problem의 __init__과 함께 Problem Class의 변수인 numCities, locations, table을 정의한다. 세 변수 모두 기존 p의 tuple로 정의된 값이었다. Tsp문제에서 First-choice와 steepest-ascent방식 모두 사용하는 함수는 createProblem, calcDistanceTable, randomInit, evaluate, inversion, describeProblem, displayResult, tenPerRow로 Tsp Class의 함수로 선언한다. 또, randomMutant, mutants, bestOf함수는 각 상황에 맞추어 사용되지만 encapsulation을 위해 Tsp Class 내부에 선언한다. 그후 기존 numCities, locations, table을 저장하던 변수 p를 함수 인자로 주는 대신 모두 self.numCities과 self.location, self.table으로 사용한다.

1-2-2. First-choice (tsp)

```
import problem


def main():
    p = problem.Tsp()
    p.createProblem()
    firstChoice(p)
    p.describeProblem()
    displaySetting()
    p.displayResult()
```

```
def firstChoice(p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    i = 0
    while i < p.LIMIT_STUCK:
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    p.saveResult(current, valueC)


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


main()
```

Tsp도 위와 유사하다. problem.py에 정의된 class를 사용하기 위해 import한다. 또, class 내부에 선언된 함수를 사용하기 위해서 모두 p.(함수이름)(함수인자)로 바꾼다.

1-2-3. Steepest-ascent (tsp)

```
import problem


def main():
    p = problem.Tsp()
    p.createProblem()
    steepestAscent(p)
    p.describeProblem()
    displaySetting()
    p.displayResult()
```

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


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


main()
```

Tsp class를 사용하기 위해 problem.py를 import한 후 class 내부로 선언된 함수를 사용할 때 변수 이름을 붙여준다.

2. Adding Gradient Descent

2-1. Gradient-descent (n)

```
import problem


def main():
    p = problem.Numeric()
    p.createProblem()
    gradientDescent(p)
    p.describeProblem()
    displaySetting(p)
    p.displayResult()
```

```
def gradientDescent(p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    while True:
        neighbors = p.gradientMutants(current)
        successor, valueS = p.bestOf(neighbors)
        if valueS >= valueC:
            break
        else:
            current = successor
            valueC = valueS
    p.saveResult(current, valueC)


def displaySetting(p):
    print()
    print("Search algorithm: Gradient-Descent Hill Climbing")
    print()
    print("Update Rate:", p.updateRate)


main()
```

Gradient-descent 방식은 위의 Steepest-ascent 방식과 유사하다. 다만 neighbors의 값을 discrete하지 않게 받는 것이므로 Numeric class에 새로 gradientMutants를 정의하여 사용하는 것을 제외하고는 크게 다르지 않다. 또, Steepest-ascent 방식 때 사용한 DELTA 값 대신 updateRate가 사용된다.

2-2. Problem.py

```
def gradientMutants(self, current):
    neighbors = []
    for i in range(0, 5):
        neighbors.append(self.gradientMutate(current, i))
    return neighbors
```

```
def gradientMutate(self, current, i):
    curCopy = current[:]
    l = self.domain[1][i]
    u = self.domain[2][i]
    curCopy2 = current[:]
    curCopy2[i] += pow(10, -4)
    curCopy2 = self.evaluate(curCopy2)
    curCopy3 = self.evaluate(current)
    if l <= (curCopy[i] - self.updateRate * (curCopy2 - curCopy3 / pow(10, -4))) <= u:
        curCopy[i] -= self.updateRate * (curCopy2 - curCopy3 / pow(10, -4))
    return curCopy
```

Gradient-descent Algorithm을 위해 Numeric Class에 두 함수를 정의하였다. 두 함수는 각각 Mutants, Mutate함수와 유사한데 neighbor의 값을 계산하는 방식만 다르기 때문이다. gradientMutate함수를 보면, gradient를 curCopy2와 curCopy3로 f(x + ε)와 f(x)를 계산한 후 x값을 x – α(updateRate) * (f(x + ε) – f(x) / ε)로 저장하고 있다. ε의 값은 과제에 제시된 대로 10의 -4승을 사용하였다. 또, 프로그램을 실행시켜보고 적당한 α값을 찾았는데, NumEval이 100을 넘도록 하는 α값으로 설정하였다.

3. 실험 결과

```
Enter the file name of a function: Ackley.txt

Objective function:
20 + math.e - 20 * math.exp(-(1/5) * math.sqrt((1/5) * (x1 ** 2 + x2 ** 2 + x3 ** 2 + x4 ** 2 + x5 ** 2))) - ma
Search space:
 x1: (-30.0, 30.0)
 x2: (-30.0, 30.0)
 x3: (-30.0, 30.0)
 x4: (-30.0, 30.0)
 x5: (-30.0, 30.0)

Search algorithm: First-Choice Hill Climbing

Mutation step size: 0.01

Solution found:
(-3.736, -0.801, 12.001, -26.988, 7.997)
Minimum value: 19.545

Total number of evaluations: 305

Process finished with exit code 0
```

First-choice (n)의 결과이다. 계산 공식은 너무 길어 사진상으로는 모두 나타나지 않았다.

```
Enter the file name of a function: Convex.txt

Objective function:
(x1 - 2) ** 2 +5 * (x2 - 5) ** 2 + 8 * (x3 + 8) ** 2 + 3 * (x4 + 1) ** 2 + 6 * (x5 - 7) ** 2
Search space:
 x1: (-30.0, 30.0)
 x2: (-30.0, 30.0)
 x3: (-30.0, 30.0)
 x4: (-30.0, 30.0)
 x5: (-30.0, 30.0)

Search algorithm: Steepest-Ascent Hill Climbing

Mutation step size: 0.01

Solution found:
(2.004, 20.222, -7.999, -1.003, 7.0)
Minimum value: 1,158.621

Total number of evaluations: 32,031

Process finished with exit code 0
```

Steepest-ascent (n)의 결과이다.

```
Enter the file name of a TSP: 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:
   12    8   28   11    2    7    4   14   24    0
   18   21   13   10   25   26    3   19   17   22
    9    6   23   15   29    5   20   16   27    1
Minimum tour cost: 1,050

Total number of evaluations: 196

Process finished with exit code 0
```

First-choice (tsp)의 결과이다.

```
Enter the file name of a TSP: 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   22   29   34   41   31    3   19   20    4
    6   36   43    1   42   10   24   32   13   37
   48   30   18   11   46    7   45   16   23   15
   14   33   47   39   38   25   35    5   40   44
   12    2   26   21   28   27    9   17    8    0
Minimum tour cost: 1,614

Total number of evaluations: 766

Process finished with exit code 0
```

Steepest-ascent (tsp)의 결과이다.

```
Enter the file name of a function: Griewank.txt

Objective function:
1 + (x1 ** 2 + x2 ** 2 + x3 ** 2+ x4 ** 2 + x5 ** 2) / 4000 - math.cos(x1) * math.cos(x2 / math.sqrt(2))
Search space:
 x1: (-30.0, 30.0)
 x2: (-30.0, 30.0)
 x3: (-30.0, 30.0)
 x4: (-30.0, 30.0)
 x5: (-30.0, 30.0)

Search algorithm: Gradient-Descent Hill Climbing

Update Rate: 1e-07

Solution found:
(17.19, 17.362, 17.876, -11.998, -26.696)
Minimum value: 1.402

Total number of evaluations: 55,531

Process finished with exit code 0
```

Gradient-descent (n)의 결과이다. 계산 공식은 너무 길어 사진상으로는 모두 나타나지 않았다.