AIP

HW 07 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202255535

이름 : 김진우

1. 서론

이번 실습의 목표는 각각의 main에 존재하는 redundancy한 코드들을 제거하는 것이다.

우선 steepest-ascent, first-choice, gradient descent 알고리즘은 모두 Hill-climbing 알고리즘이므로 공통적인 요소를 묶어 Hill-climbing class를 만들고, Numeric과 tsp는 problem 종류이기 때문에 공통 요소를 묶어 problem class를 만든다. 그리고 모든 변수를 migrate해서 이를 Setup이라는 superclass로 만든다. 그 다음 main함수에 필요한 selectProblem, selectAlgorithm, invalid와 같은 함수를 만들어 코드를 구현했다.

2. 본론

1. HW07 코드를 완성하고 Steepest Ascent, First-Choice, gradient descent 알고리즘을 사용하여 해결한 결과 terminal 스크린샷. (steepest ascent, first-choice는 tsp, numeric 두가지 유형에 대해 각각 실행 필요)

2.1 main.py

여기에서는 main(), selectProblem(), selectAlgorithm(pType), invalid(pType, aType)의 4가지 함수를 구현해야 한다. 함수 하나씩 살펴보자면

2.1.1 main()

Main()함수에서는 selectProblem()을 통해 문제유형을 numeric/tsp 중 하나로 고르고 어떤 problem type인지를 얻는다. 그 다음 selectAlgorithm를 통해 Steepest-ascent/ first-choice/gradient-descent 중 하나의 알고리즘을 선택한다. selectAlgorithm 내부에서 구체적으로 코드가 어떻게 수행되는지는 밑에서 다시 언급할 것이다. selectAlgorithm을 통해 alg을 구하면 run methode에서 p을 인자로 받아 정해진 알고리즘대로 문제를 풀고 그 후 터미널에 내용을 출력한다.

```
def main():
    p, pType = selectProblem()
    alg = selectAlgorithm(pType)
    # Call the search algorithm
    alg.run(p)
    # Show the problem solved
    p.describe()
    # Show the algorithm settings
    alg.displaySetting()
    # Report results
    p.report()
```

2.1.2 selectProblem()

이 함수에서는 numeric과 tsp 중 어떤 타입의 문제를 풀 것인지 입력을 받고 입력값에 따라 각각에 맞는 class를 problem.py로부터 불러온다. 그 다음 setVariables를 통해 값을 읽어와 저장한 뒤 p와 pType을 return 한다.

```
def selectProblem():
    print("Select the problem type:")
    print(" 1. Numerical Optimization")
    print(" 2. TSP")
    # 1 (Numeric) 또는 2 (TSP)를 입력 받아서 대응되는 Problem Class를 초기화해서 반환하기
    pType = int(input("Enter the number: "))
    if pType == 1:
        p = Numeric()
    elif pType == 2:
        p = Tsp()
    p.setVariables()
    return p, pType
```

2.1.3 selectAlgorithm(pType)

이 함수에서는 steepest-ascent, first-choice, gradient descent 중 어떤 알고리즘으로 문제를 해결할 것인지를 결정한다. 알고리즘 타입(aType)을 입력을 받은 후 invalid함수를 통해 선택한 문제를 해당 알고리즘을 통해 풀 수 있는지 없는지 여부를 확인한 후 alg 객체를 생성한 후 어떤 type으로 문제를 풀 것인지를 저장한 뒤 alg를 리턴한다.

```
def selectAlgorithm(pType):
    print()
    print("Select the search algorithm:")
    print(" 1. Steepest-Ascent")
    print(" 2. First-Choice")
    print(" 3. Gradient Descent")
    # pType == 2 (TSP)일 경우, Gradient Descent를 입력 받으면 사용자로부터 재입력 받도록 구현
    # pType과 aType이 올바르게 설정 됐는지 확인하기 위한 invalid(pType, aType) 함수 추가 구현
    while True:
        aType = int(input("Enter the number: "))
        if not invalid(pType, aType):
            break
    optimizers = { 1: 'SteepestAscent()', 2: 'FirstChoice()', 3: 'GradientDescent()' }
    alg = eval(optimizers[aType])
    alg.setVariables(pType)
    return alg
```

2.1.4 invalid(pType, aType)

tsp문제는 gradient descent방식으로 풀 수 없기 때문에 pType으로 tsp를 선택하고 aType으로 gradient descent를 선택한 경우 True를 아니면 False를 리턴하는 함수이다.

```
def invalid(pType, aType):
    if pType == 2 and aType == 3:
        print("You cannot choose Gradient Descent")
        print("   unless your want a function optimization")
        return True
    else:
        return False
```

2.2 optimizer.py

Optimizer.py에서는 HillClimbing, FirstChoice, SteepestAscent, GradientDescent class를 구현한다. 이곳에서는 알고리즘을 실행하는 method인 run() 메소드와 화면에 값이 출력되게 하는 displaySetting()과 같은 메소드들을 구현한다. 이때 Hillclimbing class는 Setup class를 superclass로, 나머지 세개의 class는 Hillclimbing class를 superclass로 갖게 한다.

2.2.1 HillClimbing(Setup)

이 class는 init, setVariables, displaySetting, runmethod를 가진다

1) __init__()

Superclass인 Setup이 가진 값들을 통해 초기화를 하고 pType과 limitStuck의 값도 초기화한다.

이때 limitStuck은 first-choice알고리즘에서 사용될 값이다.

```
class HillClimbing(Setup):
    def __init__(self):
        # 1. Setup에 정의된 delta, alpha, dx에 접근하기 위해 Setup 초기화
        # 2. self._pType 정의하기 (Tsp인지, Numeric인지 구분하기 위한 Integer 변수 선언)
        # 3. self._limitStuck 정의하기 (지금은 First-choice에서만 사용하지만, 앞으로 추가될
        # 다른 hillclimbing 알고리즘에서 사용함)
        Setup.__init__(self)
        self._pType = 0
        self._limitStuck = 100
```

2) setVariables(pType)

pType값을 설정한다.

```
def setVariables(self, pType):
    # 1. pType을 인자로 받아서 self._pType에 assign
    self._pType = pType
```

3) displaySetting()

이 메소드는 numeric 문제일때만 step size를 출력하는 메소드이다. 이때 Setup Class를 superclass로 사용하기 때문에 이전과는 달리 getDelta()를 사용하지 않는다.

```
def displaySetting(self):
    # 1. pType==1 (Numeric) 일 때만, Mutation step size를 출력하는 함수
    # first-choice.py 코드의 'print( "Mutation step size: ", p.getDelta())' 부분 활용
    if self._pType == 1:
        print("Mutation step size:", self._delta)
```

4) run()

하위 class에서 구현될 것이라 굳이 하위 class의 superclass인 여기서는 구현할 필요가 없다.

```
def run(self):
    pass
```

2.2.2 FirstChoice(HillClimbing)

이 클래스는 HillClimbing을 superclass로 가지며 displaySetting, run 메소드를 구현해야한다.

1) displaySetting()

알고리즘과 관련 setting 내용을 출력해주는 메소드이다. 이때 Hillclimbing class에서 정의해둔 displaySetting 메소드를 사용하고 limitStuck도 함께 출력해준다

```
def displaySetting(self):
    # first-choice.py 코드의 displaySetting 부분 활용
    # HillClimb에 정의했던 displaySetting을 Super를 통해 호출해서 구현하기
    print()
    print("Search algorithm: First-Choice Hill Climbing")
    super().displaySetting()
    print("Max evaluations with no improvement: {0:,} iterations".format(self._limitStuck))
```

2) run(p)

P를 통해 정해진 알고리즘에 따라 문제를 해결하는 메소드이다. 코드는 이전 과제에서 쓰인 코드와 동일하다.

```
def run(self, p):
    # first-choice.py에 정의했던 firstchoice 함수를 활용해서 구현
    current = p.randomInit()
    valueC = p.evaluate(current)
    i = 0
    while i < self._limitStuck:
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    p.storeResult(current, valueC)
```

2.2.3 SteepestAscent(HillClimbing)

1) displaySetting()

알고리즘과 관련 setting 내용을 출력해주는 메소드이다. 이때 Hillclimbing class에서 정의해둔 displaySetting 메소드를 사용한다.

```
def displaySetting(self):
    print()
    print("Search algorithm: Steepest-Ascent Hill Climbing")
    return super().displaySetting()
```

2) run(p)

P를 통해 정해진 알고리즘에 따라 문제를 해결하는 메소드이다. 코드는 이전 과제에서 쓰인 코드와 동일하다.

```
def run(self, p):
    current = p.randomInit() # 'current' is a list of values
    valueC = p.evaluate(current)
    while True:
        neighbors = p.mutants(current)
        successor, valueS = self.bestOf(neighbors, p)
        if valueS >= valueC:
            break
        else:
            current = successor
            valueC = valueS
    p.storeResult(current, valueC)
```

3) bestOf(neighbors, p)

SteepestAscent 문제를 풀 때 사용되는 함수였는데 SteepestAscent 안에서 메소드로 사용되기 위해 transformation을 진행하였다

```
def bestOf(self, neighbors, p):
        best = neighbors[0]
        bestValue = p.evaluate(best)
        for i in range(1, len(neighbors)):
            newValue = p.evaluate(neighbors[i])
            if newValue < bestValue:
                best = neighbors[i]
                bestValue = newValue # 새로운 값이 더 작으면(더 좋은 것이기때문에) 바꿔주기

        return best, bestValue
```

2.2.4 GradientDescent(HillClimbing)

1) displaySetting()

알고리즘과 관련 setting 내용을 출력해주는 메소드이다. 이때 getAlpha()와 getDx() 메소드를 사용할 필요없이 바로 값을 사용할 수 있는데 왜냐하면 부모 class인 HillClimbing class가 Setup class를 상속받기 때문이다.

```
def displaySetting(self):
    print()
    print("Search algorithm: Gradient Descent")
    print()
    print("Udate rate:", self._alpha)
    print("Increment for calculating derivative:", self._dx)
```

2) run(p)

P를 통해 정해진 알고리즘에 따라 문제를 해결하는 메소드이다. 코드는 이전 과제에서 쓰인 코드와 동일하다.

```
def run(self,p):
    currentP = p.randomInit()  # Current point
    valueC = p.evaluate(currentP)
    while True:
        nextP = p.takeStep(currentP, valueC)
        valueN = p.evaluate(nextP)
        if valueN >= valueC:
            break
        else:
            currentP = nextP
            valueC = valueN
    p.storeResult(currentP, valueC)
```

2.3 problem.py

이전 과제의 problem.py와 거의 똑같기 때문에 따로 코드를 캡처해서 하나하나 설명하는 것은 생략할 것이다. 이전과 다른 점만 서술해보자면,

1. Setup class를 상속받는다

```
class Problem(Setup):
    def __init__(self):
        Setup.__init__(self)
```

2. 이전 과제에서 사용했던 getDelta, getAlpha, getDx 메소드들을 사용하지 않는데 왜냐하면 Problem Class가 Setup Class를 상속받기 때문이다.

2.4 setup.py

Setup.py에서는 Setup class를 정의하는데 여기서는 delta, alpha, dx와 같은 variable을 정의한다.

이때 Setup class는 HillClimbing class와 Problem class의 superclass로서의 역할을 한다.

```
class Setup:
    def __init__(self):
        self._delta = 0.01
        self._alpha = 0.01
        self._dx = 10 ** (-4)
```

[전체 코드 캡쳐]

1. Main.py

```
from problem import *
from optimizer import *

def main():
    p, pType = selectProblem()
    alg = selectAlgorithm(pType)
    # Call the search algorithm
    alg.run(p)
    # Show the problem solved
    p.describe()
    # Show the algorithm settings
    alg.displaySetting()
    # Report results
    p.report()


def selectProblem():
    print("Select the problem type:")
    print(" 1. Numerical Optimization")
    print(" 2. TSP")
    # 1 (Numeric) 또는 2 (TSP)를 입력 받아서 대응되는 Problem Class를 초기화해서 반환하기
    pType = int(input("Enter the number: "))
    if pType == 1:
        p = Numeric()
    elif pType == 2:
        p = Tsp()
    p.setVariables()
    return p, pType

def selectAlgorithm(pType):
    print()
    print("Select the search algorithm:")
    print(" 1. Steepest-Ascent")
    print(" 2. First-Choice")
    print(" 3. Gradient Descent")
    # pType == 2 (TSP)일 경우, Gradient Descent를 입력 받으면 사용자로부터 재입력 받도록 구현
    # pType과 aType이 올바르게 설정 됐는지 확인하기 위한 invalid(pType, aType) 함수 추가 구현
    while True:
        aType = int(input("Enter the number: "))
        if not invalid(pType, aType):
            break
    optimizers = { 1: 'SteepestAscent()', 2: 'FirstChoice()', 3: 'GradientDescent()' }
    alg = eval(optimizers[aType])
    alg.setVariables(pType)
    return alg

def invalid(pType, aType):
    if pType == 2 and aType == 3:
        print("You cannot choose Gradient Descent")
        print("   unless your want a function optimization")
        return True
    else:
        return False

main()
```

2.Optimizer.py

```python
from setup import Setup

class HillClimbing(Setup):
    def __init__(self):
        # 1. Setup에 정의된 delta, alpha, dx에 접근하기 위해 Setup 초기화
        # 2. self._pType 정의하기 (Tsp인지, Numeric인지 구분하기 위한 Integer 변수 선언)
        # 3. self._limitStuck 정의하기 (지금은 First-choice에서만 사용하지만, 앞으로 추가될
        # 다른 hillclimbing 알고리즘에서 사용함)
        Setup.__init__(self)
        self._pType = 0
        self._limitStuck = 100

    def setVariables(self, pType):
        # 1. pType을 인자로 받아서 self._pType에 assign
        self._pType = pType

    def displaySetting(self):
        # 1. pType==1 (Numeric) 일 때만, Mutation step size를 출력하는 함수
        # first-choice.py 코드의 'print( " Mutation step size: " , p.getDelta())' 부분 활용
        if self._pType == 1:
            print("Mutation step size:", self._delta)

    def run(self):
        pass


class FirstChoice(HillClimbing):
    def displaySetting(self):
        # first-choice.py 코드의 displaySetting 부분 활용
        # HillClimb에 정의했던 displaySetting을 Super를 통해 호출해서 구현하기
        print()
        print("Search algorithm: First-Choice Hill Climbing")
        super().displaySetting()
        print("Max evaluations with no improvement: {0:,} iterations".format(self._limitStuck))


    def run(self, p):
        # first-choice.py에 정의했던 firstchoice 함수를 활용해서 구현
        current = p.randomInit()
        valueC = p.evaluate(current)
        i = 0
        while i < self._limitStuck:
            successor = p.randomMutant(current)
            valueS = p.evaluate(successor)
            if valueS < valueC:
                current = successor
                valueC = valueS
                i = 0
            else:
                i += 1
        p.storeResult(current, valueC)
```

```python
class SteepestAscent(HillClimbing):
    def displaySetting(self):
        print()
        print("Search algorithm: Steepest-Ascent Hill Climbing")
        return super().displaySetting()

    def run(self, p):
        current = p.randomInit() # 'current' is a list of values
        valueC = p.evaluate(current)
        while True:
            neighbors = p.mutants(current)
            successor, valueS = self.bestOf(neighbors, p)
            if valueS >= valueC:
                break
            else:
                current = successor
                valueC = valueS
        p.storeResult(current, valueC)

    def bestOf(self, neighbors, p):
            best = neighbors[0]
            bestValue = p.evaluate(best)
            for i in range(1, len(neighbors)):
                newValue = p.evaluate(neighbors[i])
                if newValue < bestValue:
                    best = neighbors[i]
                    bestValue = newValue # 새로운 값이 더 작으면(더 좋은 것이기때문에) 바꿔주기

            return best, bestValue


class GradientDescent(HillClimbing):

    def displaySetting(self):
        print()
        print("Search algorithm: Gradient Descent")
        print()
        print("Udate rate:", self._alpha)
        print("Increment for calculating derivative:", self._dx)

    def run(self,p):
        currentP = p.randomInit()  # Current point
        valueC = p.evaluate(currentP)
        while True:
            nextP = p.takeStep(currentP, valueC)
            valueN = p.evaluate(nextP)
            if valueN >= valueC:
                break
            else:
                currentP = nextP
                valueC = valueN
        p.storeResult(currentP, valueC)
```

3.problem.py

>> 지난주와 거의 유사하기 때문에 코드 캡처는 따로하지 않고 같이 첨부된 코드 참고

4. setup.py

```python
class Setup:
    def __init__(self):
        self._delta = 0.01
        self._alpha = 0.01
        self._dx = 10 ** (-4)
```

2.5 실행결과

1) problem/Convex.txt

Steepest Ascent

```
Select the problem type:
 1. Numerical Optimization
 2. TSP
Enter the number: 1
Enter the file name of a function: problem/Convex.txt

Select the search algorithm:
 1. Steepest-Ascent
 2. First-Choice
 3. Gradient Descent
Enter the number: 1

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, 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:
(2.002, 4.996, -8.002, -0.998, 7.004)
Minimum value: 0.000

Total number of evaluations: 84,911
```

First-choice

```
Select the problem type:
 1. Numerical Optimization
 2. TSP
Enter the number: 1
Enter the file name of a function: problem/Convex.txt

Select the search algorithm:
 1. Steepest-Ascent
 2. First-Choice
 3. Gradient Descent
Enter the number: 2

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, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: First-Choice Hill Climbing
Mutation step size: 0.01
Max evaluations with no improvement: 100 iterations

Solution found:
(1.998, 5.003, -8.003, -0.997, 7.001)
Minimum value: 0.000

Total number of evaluations: 33,755
```

Gradient Descent

```
Select the problem type:
 1. Numerical Optimization
 2. TSP
Enter the number: 1
Enter the file name of a function: problem/Convex.txt

Select the search algorithm:
 1. Steepest-Ascent
 2. First-Choice
 3. Gradient Descent
Enter the number: 3

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, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Search algorithm: Gradient Descent

Udate rate: 0.01
Increment for calculating derivative: 0.0001

Solution found:
(2.0, 5.0, -8.0, -1.0, 7.0)
Minimum value: 0.000

Total number of evaluations: 10,639
```

2) problem/tsp50.txt

Steepest Ascent

```
Select the problem type:
 1. Numerical Optimization
 2. TSP
Enter the number: 2
Enter the file name of a TSP: problem/tsp50.txt

Select the search algorithm:
 1. Steepest-Ascent
 2. First-Choice
 3. Gradient Descent
Enter the number: 1

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:
   24   14    7   43   22   47   12   27   44    8
    2    1   28   41   21    6    3   20    5    4
   33   45   10   37   26   32   34   42   31   19
   17   35   38   16    0   11   48   40   23    9
   36   46   15   30   18   25   49   13   39   29
Minimum tour cost: 757

Total number of evaluations: 2,653
```

Gradient Descent 선택 (실패 후) -> First-choice 선택

```
Select the problem type:
 1. Numerical Optimization
 2. TSP
Enter the number: 2
Enter the file name of a TSP: problem/tsp50.txt

Select the search algorithm:
 1. Steepest-Ascent
 2. First-Choice
 3. Gradient Descent
Enter the number: 3
You cannot choose Gradient Descent
   unless your want a function optimization
Enter the number: 2

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
Max evaluations with no improvement: 100 iterations

Best order of visits:
    7   22   12   27    8   47   44   49   13   39
    2   37   10   26   28    1   41   21    6    3
   32   33   45    4   31   20    5   38   35   16
   17    0   19   11   42   23   40   48    9   36
   18   30   46   15   29   24   34   14   25   43
Minimum tour cost: 782

Total number of evaluations: 1,255
```

3. 결론

이번주에 수행한 과제는 코드실행의 결과는 지난주와 크게 다르지 않아 크게 분석을 할 필요가 없었다. 그래도 한번 더 분석해보자면 numeric의 경우 평가 횟수는 First choice < Steepest ascent < Gradient descent 순서임을 알 수 있고, 이때 minimum value는 모두 0으로 동일하다는 것을 확인할 수 있다. tsp의 경우 평가 횟수는 First choice < Steepest ascent 순서인데 이 이유는 이전 과제에서부터 계속 설명했듯 first choice가 연산 횟수가 더 적기 때문이다.

또한 tsp문제에를 Gradient descent 방법으로 풀게 선택했을 때, "You cannot choose Gradient Descent"라는 문구를 나오게 해서 tsp 문제를 풀 수 있는 다른 알고리즘을 선택하도록 설계했다.

지난주에 했던 class화를 조금 더 체계적으로 설계 함으로서 코드의 redundancy를 줄였다.