AI Programming
[Week 12] Practice

2024. 11. 21.

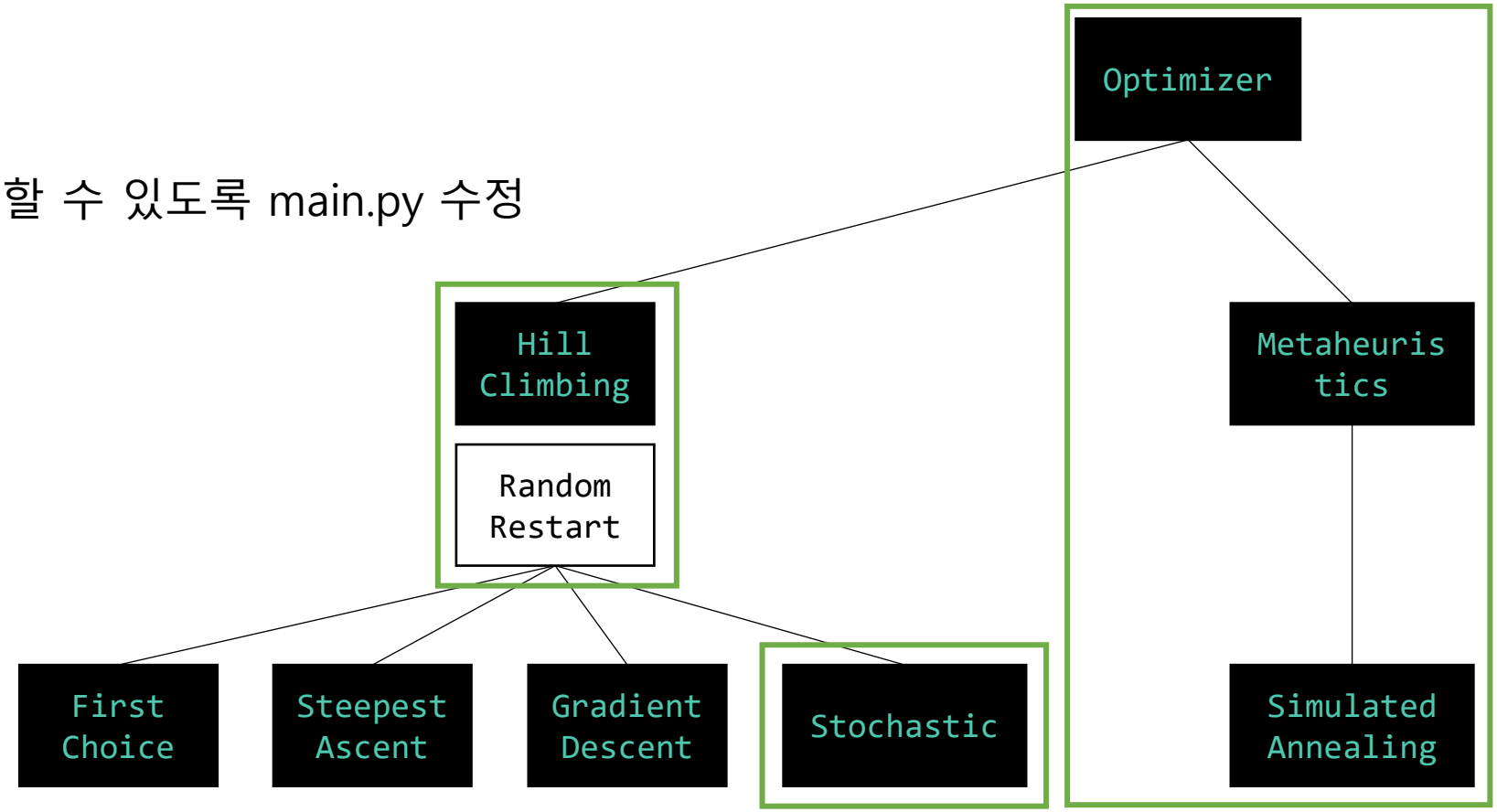
목차

HW07 본인 제출 파일 준비

- main.py
- optimizer.py
- problem.py
- setup.py

실습 목표

1. Stochastic hill climbing 추가
2. Random Restart 추가
3. Simulated Annealing 추가
4. exp.txt를 사용해서 실험을 수행할 수 있도록 main.py 수정



optimizer.py 실습

Inference 구성하기

```
class HillClimbing(Optimizer):
    ...

    def __init__(self):
        Optimizer.__init__(self)
        self._pType = 0
        self._limitStuck = 100
```

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

class MetaHeuristic(Optimizer):
    def __init__(self):
        Optimizer.__init__(self)

class SimulatedAnnealing(MetaHeurstic):
    def __init__(self):
        MetaHeurstic.__init__(self)

class Stochastic(HillClimbing):
    def __init__(self):
        pass
```

Stochastic Hill climbing 실습 (10분)

Stochastic Class 구현하기
*some useful codes 사용

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

    def run(self, p):
        # hint; Stochastic 알고리즘은 Steepest Ascent 알고리즘과 흐름이 유사함
        ...
        p.storeResult(current, valueC)

    def stochasticBest(self, neighbors, p):
        # some useful codes 사용하기
        return neighbors[i], valuesForMin[i]
```

optimizer.py 실습


```
def selectAlgorithm(pType):
    print()
    print("Select the search algorithm:")
    print("  1. Steepest-Ascent")
    print("  2. First-Choice")
    print("  3. Gradient Descent")
    print("  4. Stocahstic")

    while True:
        aType = int(input("Enter the number: "))
        if not invalid(pType, aType):
            break
    optimizers = { 1: 'SteepestAscent()',
                   2: 'FirstChoice()',
                   3: 'GradientDescent()',
                   4: 'Stochastic()' }
    alg = eval(optimizers[aType])
    alg.setVariables(pType)
    return alg
```

Main문에 추가하여
Stochastic 실행해보기

*일부 문제는 **시간이 오래 걸릴 수 있음**

Ackley.txt or
Griewank.txt or
tsp 문제 풀어보기

(5분)

Random Restart 실습 (10분)

Random Restart는 기존의 HillClimbing methods들을 주어진 실험 수 만큼 반복적으로 수행하여 더 좋은 결과를 찾는 방법으로,

HillClimbing methods들이 공통적으로 수행할 수 있도록 HillClimbing Class에 randomRestart Method를 추가

재대로 실행될 수 있도록 화살표된 부분 구현해주기

```
class HillClimbing(Optimizer):
    def __init__(self):
        ...
        self._numRestart = 1
        ...

    def randomRestart(self, p):
        i = 1
        self.run(p)
        bestSolution = p.getSolution()
        bestMinimum = p.getValue()
        numEval = p.getNumEval()
        while i < self._numRestart:
            self.run(p)
            newSolution = p.getSolution()
            newMinimum = p.getValue()
            numEval += p.getNumEval()
            if newMinimum < bestMinimum:
                bestSolution = newSolution
                bestMinimum = newMinimum
            i += 1
        p.storeResult(bestSolution, bestMinimum)
```

Random Restart 실습 (5분)

```
def main():
    p, pType = selectProblem()
    alg = selectAlgorithm(pType)
    alg.randomRestart(p)
    p.describe()
    alg.displaySetting()
    p.report()
```

run을 randomRestart로 변경해서

Numeric – Ackley.txt – Gradient Descent 방식으로 문제 풀기

Hillclimbing – self._numRestart = **1 and 10** 으로 각각 실행해서 결과(min value) 비교해보기

Simulated Annealing 실습

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state

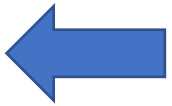
inputs: *problem*, a problem

schedule, a mapping from time to "temperature"

current ← MAKE-NODE(*problem*.INITIAL-STATE)

for $t \leftarrow 1$ **to** ∞ **do**

T ← *schedule*[*t*]



While 문으로 대체하시면 됩니다.

if $T = 0$ **then return** *current*

next ← a randomly selected successor of *current*

$\Delta E \leftarrow$ *next*.VALUE − *current*.VALUE

if $\Delta E < 0$ **then** *current* ← *next*

else *current* ← *next* only with probability $e^{-\Delta E/T}$

Simulated Annealing 실습 (10분)

```
class SimulatedAnnealing:
    def __init__(self):
        self._numSample = 100
        self._limitEval = 100000
        self._whenBestFound = 0

    def displaySetting(self):
        print("Search Algorithm: Simulated Annealing")
        print("Number of evaluations until termination: {0:,}"
              .format(self._limitEval))

    def run(self, p):
        current = p.randomInit()
        valueC = p.evaluate(current)
        best, valueBest = current, valueC
        whenBestFound = i = 1
        t = self.initTemp(p)
        while True:
            ... # Implement
        self._whenBestFound = whenBestFound
        p.storeResult(best, valueBest)
```

Some useful code 활용하여 SimulatedAnnealing 구현

Some useful code 안의 함수들은 SimulatedAnnealing Class 안으로 옮겨서 사용

optimizer.py 실습

Simulated Annealing 수행 (5분)

Simulated Annealing 코드 수행해보기

1. main.py-selectAlgorithm에 Simulated Annealing 추가

2. main.py-main에 if 분기문 추가

3. 임시적으로 구현한 Simulated Annealing이 잘 수행될 수 있도록 setVriables method 추가

Tsp100.txt 문제 풀어보고 hill climbing 알고리즘과 결과 비교해보기

```
class SimulatedAnnealing(MetaHeuristic):
    def setVariables(self, _):
        pass
```

```
def selectAlgorithm(pType):
    ...
    print("  4. Stocahstic")
    print("  5. SimulatedAnnealing")
    while True:
        aType = int(input("Enter the number: "))
        if not invalid(pType, aType):
            break
    optimizers = { 1: 'SteepestAscent()',
                   2: 'FirstChoice()',
                   3: 'GradientDescent()',
                   4: 'Stochastic()',
                   5: 'SimulatedAnnealing()'}
    alg = eval(optimizers[aType])
    ...
```

```
def main():
    ...
    if issubclass(type(alg), HillClimbing):
        alg.randomRestart(p)
    else:
        alg.run(p)
```

main.py 실습

Text 파일(exp.txt)로부터 상수를 읽어와서 사용하도록 변경

알고리즘을 여러 번 실험할 수 있도록 Main 함수 변경

Main 함수에서 여러 번 실험을 통해 결과를 report할 수 있도록 여러가지 기능 확장 예정

```
# If you are solving a function optimization problem,
#   what should be the step size for axis-parallel mutation?
   Mutation step size (delta ) : 0.01
#
# If your algorithm choice is 2 or 3,
#   what should be the number of consecutive iterations without improvement?
   Give the number of iterations (limitStuck) : 1000
#
# If your algorithm choice is 4 (gradient descent),
# what should be the update step size and increment for derivative?
   Update rate for gradient descent (alpha) : 0.01
   Increment for calculating derivative (dx) : 10 ** (-4)
#
# If you want a random-restart hill climbing,
#   enter the number of restart.
# Enter 1 if you do not want a random-restart.
   Number of restarts (numRestart) : 10
#
# If you are running a metaheuristic algorithm,
#   what should be the total number of evaluations until temination?
  Enter the number (limitEval) : 100000
#
# Enter the total number of experiments
   Enter the number (numExp) : 10
```

main.py 실습

main – skeleton.py 완성하고 코드 이해하기

```
def createProblem(parameters):
    # debugger로 parameters 인자 확인하기
    pType = parameters['pType']
    if pType == 1:
        p = Numeric()
    elif pType == 2:
        p = Tsp()
    # p.setVariables 함수 수정하기
    p.setVariables(parameters)
    return p
```

```
def createOptimizer(parameters):
    optimizers = { 1: 'SteepestAscent()',
                   2: 'FirstChoice()',
                   3: 'Stochastic()',
                   4: 'GradientDescent()',
                   5: 'SimulatedAnnealing()' }
    aType = parameters['aType']
    alg = eval(optimizers[aType])
    # alg.setVariables 함수 수정하기
    alg.setVariables(parameters)
    return alg
```

main.py 실습

```python
class Problem(Setup):
    def __init__(self):
        Setup.__init__(self)
        self._solution = []
        self._value = 0
        self._numEval = 0

        self._pFileName = ''
        self._bestSolution = []
        self._bestMinimum = 0
        self._avgMinimum = 0
        self._avgNumEval = 0
        self._sumOfNumEval = 0
        self._avgWhen = 0

    def setVariables(self, parameters):
        Setup.setVariables(self, parameters)
        self._pFileName = parameters['pFileName']
```

```python
class Optimizer(Setup):
    def __init__(self):
        Setup.__init__(self)
        self._numExp = 0

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

```python
class Setup:
    def __init__(self):
        self._pType = 0
        self._aType = 0
        self._delta = 0
        self._alpha = 0
        self._dx = 0
    def setVariables(self, parameters):
        self._pType = parameters['pType']
        self._aType = parameters['aType']
        self._delta = parameters['delta']
        self._alpha = parameters['alpha']
        self._dx = parameters['dx']
    def getAType(self):
        return self._aType
```

main.py 실습

exp.txt 설정: tsp – tsp50.txt – first choice – limit stuck = 1000, numExp = 10
위 설정으로 구현했던 main.py debugging하며 필요한 위치에 method 추가 및 변경하기
(과제 시에는 다른 설정에서도 main.py가 실행될 수 있도록 구현)

*출력 결과도 예시와 같이 나올 수 있도록 변경

main.py 실습

```
Number of cities: 50
City locations:
    (96, 22)    (56, 12)    (19, 24)    (83, 58)     (62, 5)
    (79, 31)     (1, 0)     (29, 71)    (17, 89)    (43, 66)
    (82, 74)    (52, 35)    (84, 92)    (93, 45)    (41, 24)
    (36, 83)    (82, 35)    (89, 71)    (93, 89)    (67, 10)
    (71, 82)    (68, 50)    (84, 81)    (74, 94)    (53, 13)
    (81, 31)    (17, 92)    (99, 82)    (25, 63)     (0, 2)
    (21, 83)    (70, 64)     (79, 6)    (31, 53)    (90, 50)
    (48, 14)    (41, 26)    (80, 56)    (49, 51)    (19, 38)
     (2, 0)     (29, 63)    (18, 59)    (10, 44)     (49, 7)
    (37, 9)     (19, 14)    (90, 85)    (100, 5)    (34, 55)

Number of experiments: 10

Search Algorithm: First-Choice Hill Climbing
Max evaluations with no improvement: 1,000 iterations

Best order of visits:
  37   34   13   16    5   25    0   48   32   19
   4   44    1   24   35   11   36   14   45   40
   6   29   46    2   39   43   33   49   42   28
  41    7   30    8   26   15    9   38   21   31
  20   23   12   47   18   27   17   22   10    3
Minimum tour cost: 588

Total number of evaluations: 57,792
```

```
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)

Number of experiments: 10

Search Algorithm: First-Choice Hill Climbing

Max evaluations with no improvement: 1,000 iterations

Average tour cost: 649
Average number of evaluations: 5,455

Best tour found:
   1   28   41   10   37   26   49    2   44    8
  27   47   12   22   43    7   25   14   39   13
  24   34   40   18   29   30   46   15   36    9
  23   48   42   31   11   19    0   16   17   35
  38    5   20    4   33   32   45    3   21    6
Best tour cost: 624

Total number of evaluations: 54,546
```

과제 안내

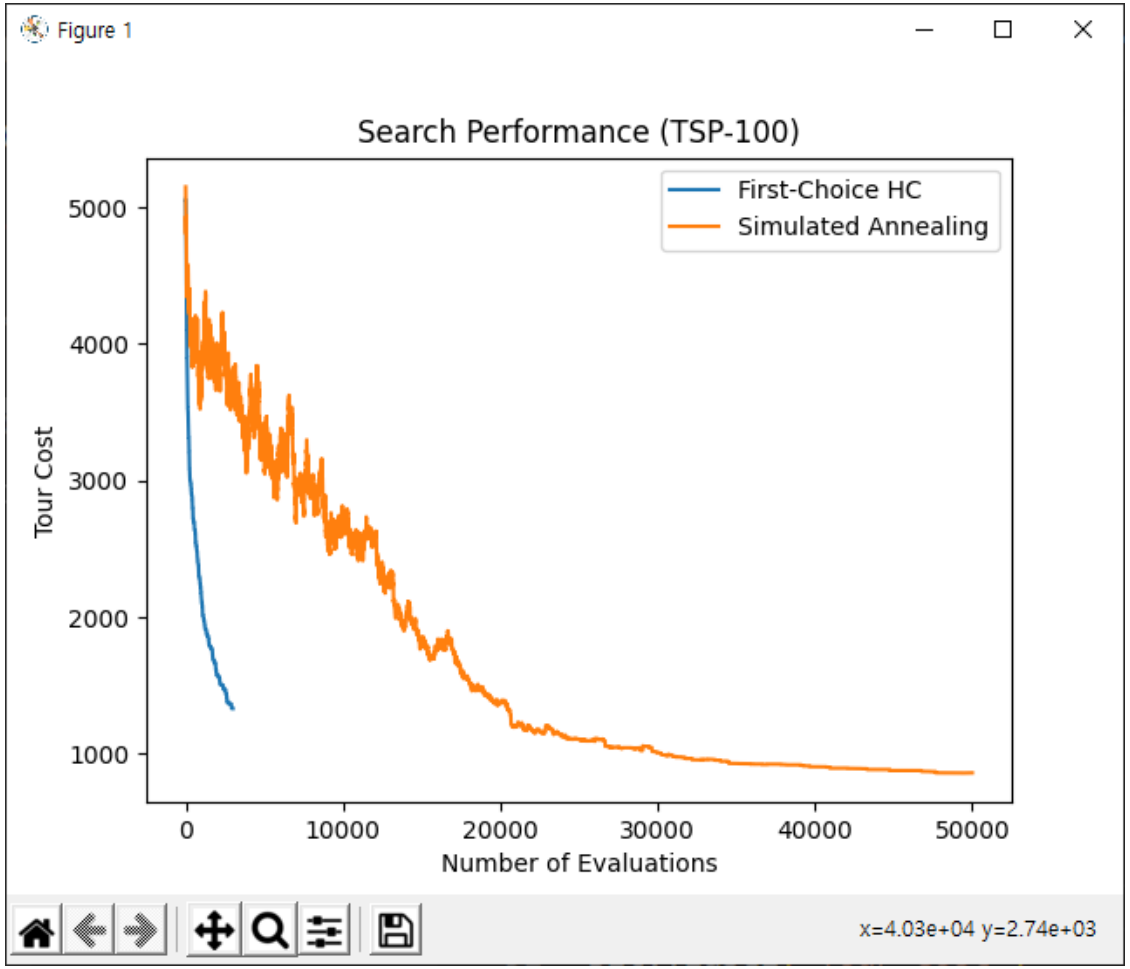
제공된 main – skeleton 코드를 사용하여 아래 알고리즘들 모두 정상적으로 수행될 수 있도록 구현
(output으로 실험 결과가 정상적으로 출력될 수 있어야 함)

* 반드시 구현했던 Class들을 활용하여 실행되어야 함

Hill climbing (random start)
- **Stochastic**
- **first-choice**
- **steepest ascent**
- **gradient descent**

Meta Heuristics
- **Simulated Annealing**

+ First Choice, Simulated Annealing에서 NumEval에 따른 current Value를 시각화해서 제출 (tsp100.txt) 사용



과제 안내

제출물:

파이썬 파일 총 5개를 **HW08_NAME 폴더**로 묶어서 **압축**하여 제출 (.zip)

- main.py
- optimizer.py
- problem.py
- setup.py
- plot.py

리포트 제출 (**.pdf**)

- Stochastic, simulated Anneaing으로 Numeric, TSP 문제 각각 실행한 terminal 스크린샷 총 4개
- Plot.py로 만든 그래프 스크린샷

과제 안내

알고리즘 수행 과정에서
File I/O를 이용해서
매 iteration마다 결과를 저장

Plot.py에서는 저장된 결과를
불러와서 그래프로 표시

```
class Stochastic(HillClimbing):
    def displaySetting(self):
        print()
        print("Search Algorithm: Stochastic Hill Climbing")

    def run(self, p):
        current = p.randomInit()
        valueC = p.evaluate(current)
        f = open('stochastic.txt', 'w')
        i = 0
        while i < self._limitStuck:
            neighbors = p.mutants(current)
            successor, valueS = self.stochasticBest(neighbors, p)
            f.write(str(valueC)+'\n')
            if valueS < valueC:
                current = successor
                valueC = valueS
                i = 0
            else:
                i += 1
        p.storeResult(current, valueC)
        f.close()
```