AI Programming
[Week 9] Practice

2024. 10. 31.

목차

- 실습 준비
- 실습 목표
- Search Algorithm - Steepest Ascent, First Choice
- Numeric optimization, Travelling salesman Problem (TSP)
- TSP 실습
- Numeric 실습
- 과제 안내

Steepest Ascent, First Choice를 통해 **Numeric Optimization, TSP** 해결

*'HW05' 폴더의 skeleton code 이용

1. Steepest Ascent를 이용해 TSP를 수행하는 코드 작성
2. First Choice를 이용해 TSP를 수행하는 코드 작성
3. Steepest Ascent를 이용해 Numeric Optimization을 수행하는 코드 작성
4. First Choice를 이용해 Numeric Optimization을 수행하는 코드 작성

공통적으로 사용되는 코드 **리팩토링(모듈화)**

*'HW05 (using modules)' 폴더의 skeleton code 이용

1. Numeric Optimization을 해결할 때, 공통적으로 사용되는 코드를 Numeric.py로 모듈화
2. TSP를 해결할 때, 공통적으로 사용되는 코드를 TSP.py로 모듈화

Steepest Ascent

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

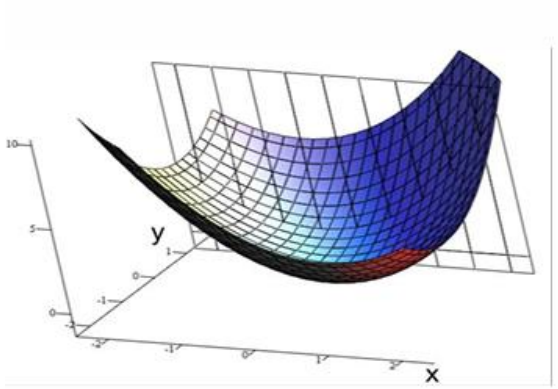
모든 neighbors 중, 가장 좋은 값(Best Value)을 골라 현재 값(Current Value)과 비교
Best Value > Current Value 이면, State를 Update

First Choice

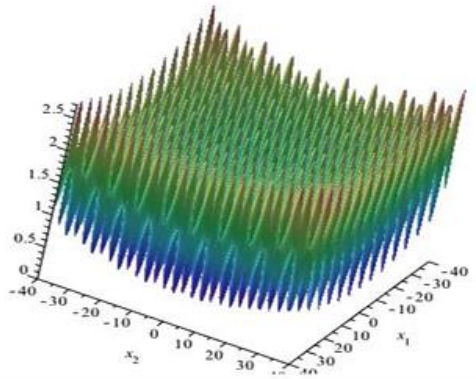
Generates Successors randomly until one is found that is better than the current state

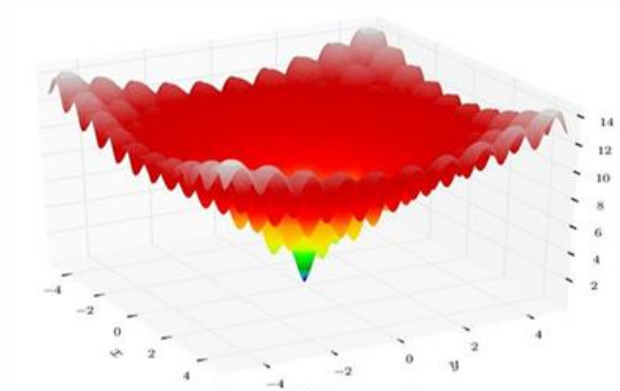
Numeric Optimization

<Convex.txt>
y = (x1 - 2) ** 2 +5 * (x2 - 5) ** 2 + 8 * (x3 + 8) ** 2 + 3 * (x4 + 1) ** 2 + 6 * (x5 - 7) ** 2
x1,-30,30
x2,-30,30
x3,-30,30
x4,-30,30
x5,-30,30 주어진 함수의 최소 값을 찾는 문제



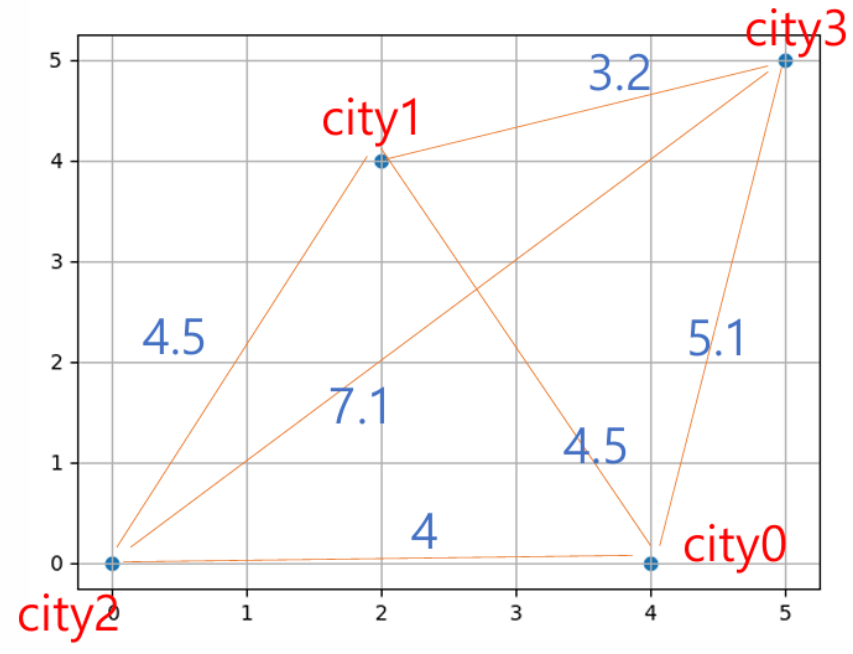
Bivariate convex


Second order Griewank


2 variable Ackley

TSP

<TSP30.txt>
30
(8, 31)
(54, 97)
(50, 50)
(65, 16)
(70, 47)
(25, 100)
(55, 74)
(77, 87)
(6, 46)
(70, 78)
(13, 38)
…



출발 / 도착	0	1	2	3
0	0	4.5	4	5.1
1	4.5	0	4.5	3.2
2	4	4.5	0	7.1
3	5.1	3.2	7.1	0

주어진 도시들의 최단 travel 경로를 찾는 문제

TSP 실습 – First Choice (8mins)

HW05 - first choice (tsp).py

```
def calcDistanceTable(numCities, locations): ###
    return table # A symmetric matrix of pairwise distances
```

numCities = 30
locations = [(8, 31), (54, 97), (50, 50), (65, 16), ...]

table =

출발 / 도착	0	1	2	3
0	0	4.5	4	5.1
1	4.5	0	4.5	3.2
2	4	4.5	0	7.1
3	5.1	3.2	7.1	0

City 0 <-> City 1 사이의 거리 = 4.5
City 1 <-> City 0 사이의 거리 = 4.5

TSP 실습 – First Choice (8mins)

HW05 - first choice (tsp).py

```
def evaluate(current, p): ###
    ## Calculate the tour cost of 'current'
    ## 'p' is a Problem instance
    ## 'current' is a list of city ids
    return cost
```

current = [24, 31, 25, 8, 7, 15, ...], 현재 travel할 경로 list
p = [num of cities, location of cities, symmetric distance table]

Current 경로를 따라 이동거리(cost)가 얼마나 되는지 symmetric distance table을 이용해서 계산

Current가 [1, 2, 3] 일 경우,
1->2
2->3
3->1 처럼 마지막 돌아오는 경로에 주의하여 구현

TSP 실습 – Steepest Ascent

HW05-steepest ascent (tsp).py

First Choice에서 구현했던 **evaluate, calcDistanceTable** 함수 재사용

TSP 실습 – Steepest Ascent (8 mins)

HW05-steepest ascent (tsp).py

```
def bestOf(neighbors, p): ###
    return best, bestValue
```

neighbors =
[[24, 31, 25, 8, 7, 15, …],
[31, 24, 25, 8, 7, 15, …],
[24, 31, 8, 25, 7, 15, …],
…] #mutants 함수에 의해 생성 된 current의 이웃들

p = [num of cities, location of cities, symmetric distance table]

evaluate 함수를 이용해서 이웃들 중 가장 거리가 짧은 이웃(best), 그 거리(bestValue)를 반환

TSP 실습 – modularization (8mins)

HW05 (using modules) – tsp.py, steepest ascent (tsp).py, first choice (tsp).py

1. Steepest ascent, First Choice에서 공통적으로 사용한 함수를 tsp.py에 구현
2. Steepest ascent, First Choice에서 tsp.py import 하기
3. 모듈화 된 Steepest ascent (tsp), first choice (tsp) 실행해보기

Numeric 실습 – first choice (8 mins)

HW05 – first choice (n).py

```
def createProblem(): ###
    ## Read in an expression and its domain from a file.
    ## Then, return a problem 'p'.
    ## 'p' is a tuple of 'expression' and 'domain'.
    ## 'expression' is a string.
    ## 'domain' is a list of 'varNames', 'low', and 'up'.
    ## 'varNames' is a list of variable names.
    ## 'low' is a list of lower bounds of the varaibles.
    ## 'up' is a list of upper bounds of the varaibles.
    return expression, domain
```

```
Enter the file name of a function: problem/Convex.txt # Example Input
```

Problem File 경로를 입력받아 File I/O를 이용해서 problem 읽어온 후 반환하는 함수

expression = '(x1 - 2) ** 2 +5 * (x2 - 5) ** 2 + 8
* (x3 + 8) ** 2 + 3 * (x4 + 1) ** 2 + 6 * (x5 – 7) ** 2'

domain = [['x1', 'x2', 'x3', 'x4', 'x5'],
[-30, -30, -30, -30, -30],
[30, 30, 30, 30, 30]]

과제 안내 – Numeric – first Choice


```
def randomInit(p): ###
    return init    # Return a random initial point
                   # as a list of values
```

p = (expression, domain)
init = [value1, value2, value3, …]

Init의 길이는 p에 정의된 variable 수와 같아야 함.
value를 random하게 초기화할 때, value는 각 variable의 lower bound, upper bound 내에 포함되어야 함

```
def randomMutant(current, p): ###
    return mutate(current, i, d, p) # Return a random successor
```

mutate(current, i, d, p): current의 i 번째 variable을 d 만큼 바꾸는 함수

i를 랜덤하게 선택하고, d는 미리 정의된 global variable DELTA를 사용
D는 0.5의 확률로 증가할지, 감소할지 정해져야 함

과제 안내 – Numeric – Steepest Ascent

```
def mutants(current, p): ###
    return neighbors     # Return a set of successors
```

가능한 모든 이웃 경우의 수(neighbors)를 반환

P에 정의된 변수가 5개이면 각 변수마다 DELTA만큼 증가, 감소하여 총 10개의 neighbors를 반환함

이 때, variable이 domain 값에 포함되면 값을 변화시키고, 그렇지 않으면 그대로 반환

```
def bestOf(neighbors, p):
    return best, bestValue
```

evaluate 함수를 이용해서 이웃들 값이 가장 작은 이웃(best)과 그 값(bestValue)을 반환

과제 안내 – Numeric – Modularization

1. [HW05] – First Choice, Steepest Ascent를 완성
2. [HW05 (using modules)] – Numeric.py에 두 알고리즘의 공통된 부분을 작성
3. [HW05 (using modules)] – 공통된 부분을 제외한 나머지 내용 작성

과제 안내 – 제출물

6개의 .py파일을 하나의 폴더로 묶어서 압축하여 제출 (HW05_NAME.zip)
과제 리포트 제출

HW05_NAME/
- tsp.py
- first-choice (tsp).py
- steepest ascent (tsp).py
- numeric.py
- first-choice (n).py
- steepest ascent (n).py

HW05_NAME.pdf – 첨부한 template 참조