AIP

HW 08 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202255535

이름 : 김진우

1. 서론

이번 과제에서는 stochastic Hill Climbing, random-restart, simulated annealing 알고리즘을 추가한 프로그램을 설계해야 한다. 우선 Stochastic Hill climbing은 evaluate값에 따른 확률을 배정한 후 이 확률로 neighbor를 뽑아 이전보다 좋아지면 새로운 값으로 solution을 대체하고 이 과정을 더 이상 좋아지지 않을 때까지 반복하는 것이다. Random-Restart는 말 그대로 알고리즘을 처음부터 다시 실행하는 것으로 시작점을 바꾸어 실행할 때 가장 좋은 것을 선택하는 알고리즘이다. Simulated-Annealing은 기존의 hill-climbing 방법과 유사하지만 차이점이 존재하는 알고리즘인데 바로 “안 좋아지는 방향”도 허용한다는 점이다. 기존의 알고리즘은 좋아지는 방향만 허용하기 때문에 local minimum(max)에 갇혀 빠져나오지 못할 가능성이 있는데 안 좋아지는 방향도 허용을 하면 더 좋은 값을 찾을 수 있게 된다. 또한 이번 과제에서는 exp.txt파일 내의 값만 수정하여 main을 실행시키기 때문에 이에 맞춰 main.py를 수정해야 하고 plot.py를 작성하여 그래프를 통해 성능 비교까지 해봐야 한다.

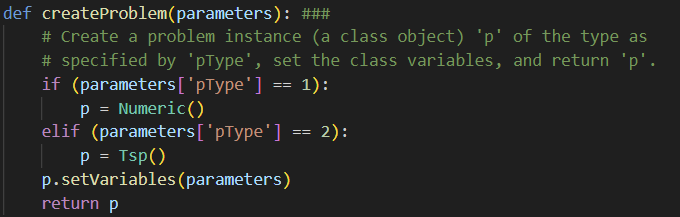
2. 본론

**2.1 main.py**

Main.py에서는 exp.txt에서 정보를 읽어와 가장 좋은 결과를 출력해야한다. Main.py는 main-skeleton.py에서 비어있는 createProblem/createOptimizer를 구현하여 완성하면된다.

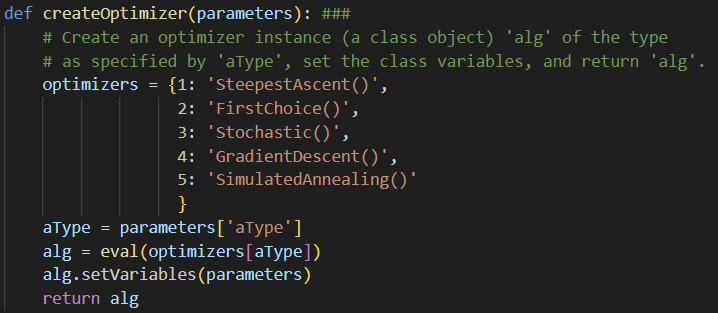
2.1.1 createProblems(parameters)

이 함수는 parametes에 저정되어있는 pType을 읽어와서 pType이 1이면 Numeric()을 2이면 Tsp() 문제가 되게하고 setVariables를 통해 설정을 한 뒤 p를 return한다.



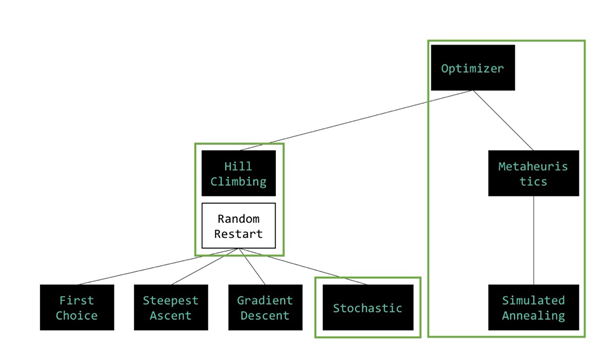
2.1.2 createOptimizer(parameters)

이 함수는 aType을 읽어와 aType에 맞는 알고리즘을 선택하고 setVariables를 통해 필요한 변수들을 생성하고 이를 return한다.



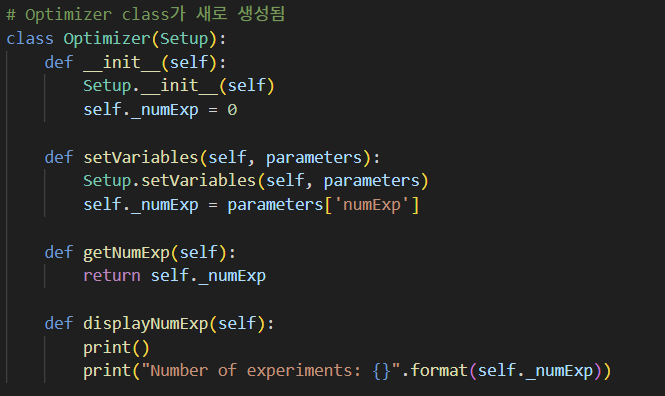
**2.2 optimizer.py**

여기서는 새롭게 Optimizer class를 생성하여 HillClimibing과 Metaheuristic이 이를 상속하게 한다. 그 다음 Hillclimbing을 First-choice, Steepest-Ascent, Gradient Descent, Stochastic이 상속하도록 하고 Metaheuristic은 Simulated Annealing이 상속하도록 class화 한다. (아래의 그림 참고)



**2.2.1. Optimizer(Setup)**

Setup class를 super class로 가지며 numExp 변수를 저장한다. setVariables 메소드에서는 Setup의 setVariables로 변수를 설정하고 numExp도 추가로 설정해준다. numExp를 위한 accessor 메소드도 구현을 하고 numExp를 출력 양식에 맞춰 터미널에 출력해주는 displayNumExp 메소드도 같이 구현을 해준다.

****

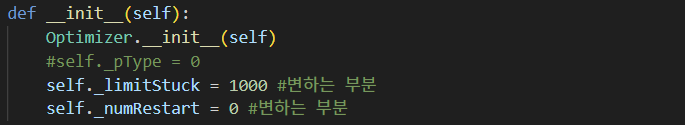
**2.2.2 HillClimbing(Optimizer)**

Setup이 아닌 새로운 Optimizer class를 상속받는다.

변한 부분만 구체적으로 하나씩 살펴보자면

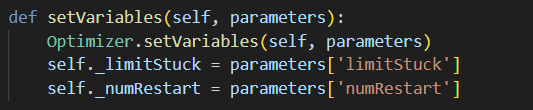
- \_\_init\_\_

이 메소드는 Setup이 아닌 새로운 Optimizer class를 상속받고, limitStuck와 numRestart라는 변수를 저장한다



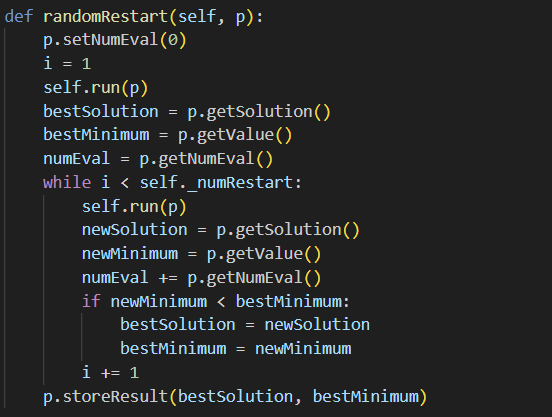
- setVariables(parameters)

각종 변수들을 설정하기 위한다. 이때 부모 class의 setVariables를 사용하고 limitStuck과 numRestart에 대해서도 설정을 해준다.



- randomRestart(p)

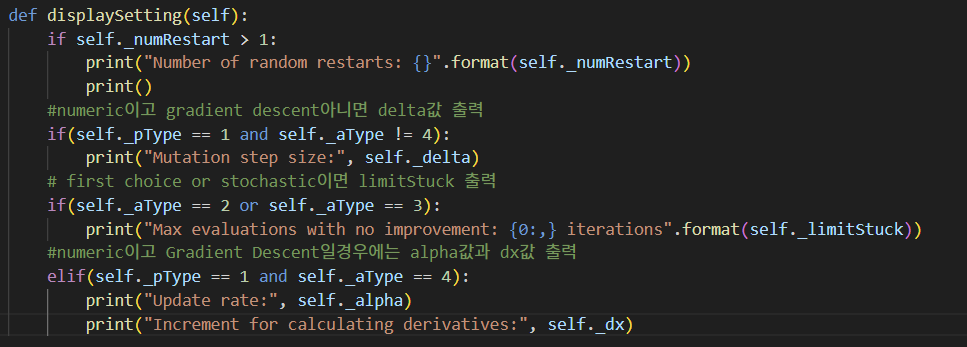
이번 과제에서는 Hill-climbing에 대해서 randomRestart를 적용시키기 때문에 이를 위한 메소드를 작성한다. 이 코드는 실습에서 진행하였기 때문에 설명은 생략하도록 하겠다.



- displaySetting

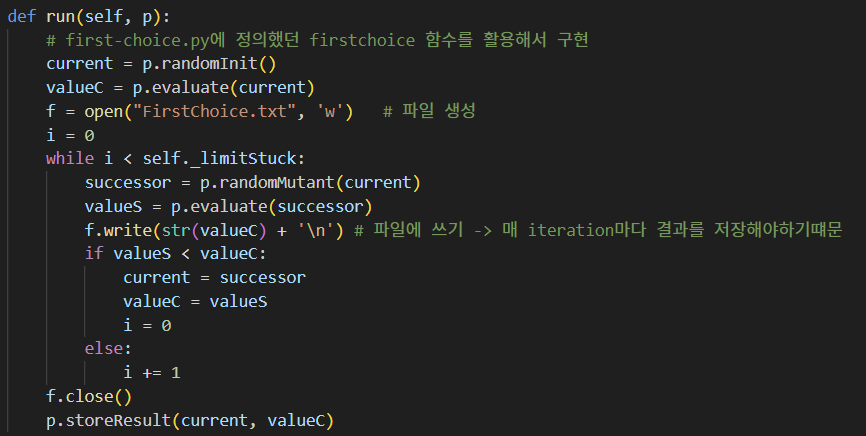
출력 양식에 맞게 출력되도록 displaySetting 메소드를 작성한다. 자세한 설명은 아래 사진에 첨부된 코드와 주석을 참고하길 바란다.

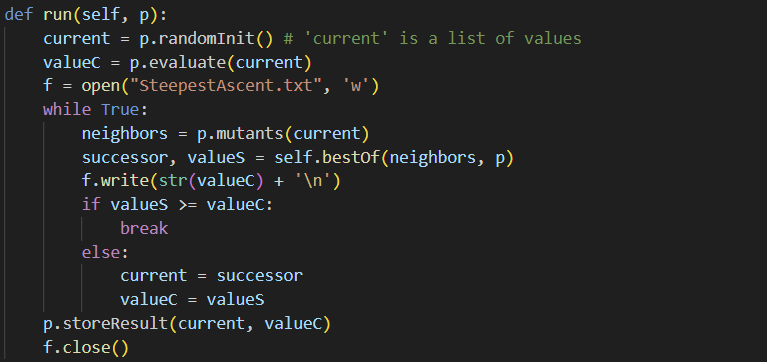
출력 형식은 모두 과제와 함께 첨부된 pdf와 실습 ppt를 참고하였다.

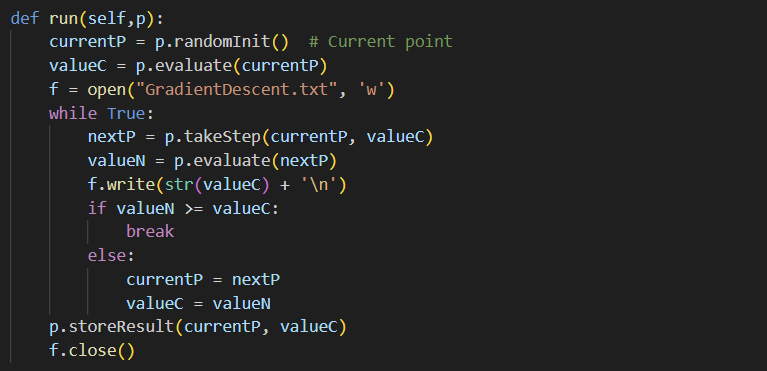


**2.2.3. FirstChoice, SteepestAscent, GradientDescent(Hillcliming)**

알고리즘 수행 과정에서 File I/O를 사용해 매 iteration마다 결과를 저장해야하기 위한 코드를 run 메소드에 추가한다. 3개의 class모두 같은 흐름이기 때문에 묶어서 설명하였다.

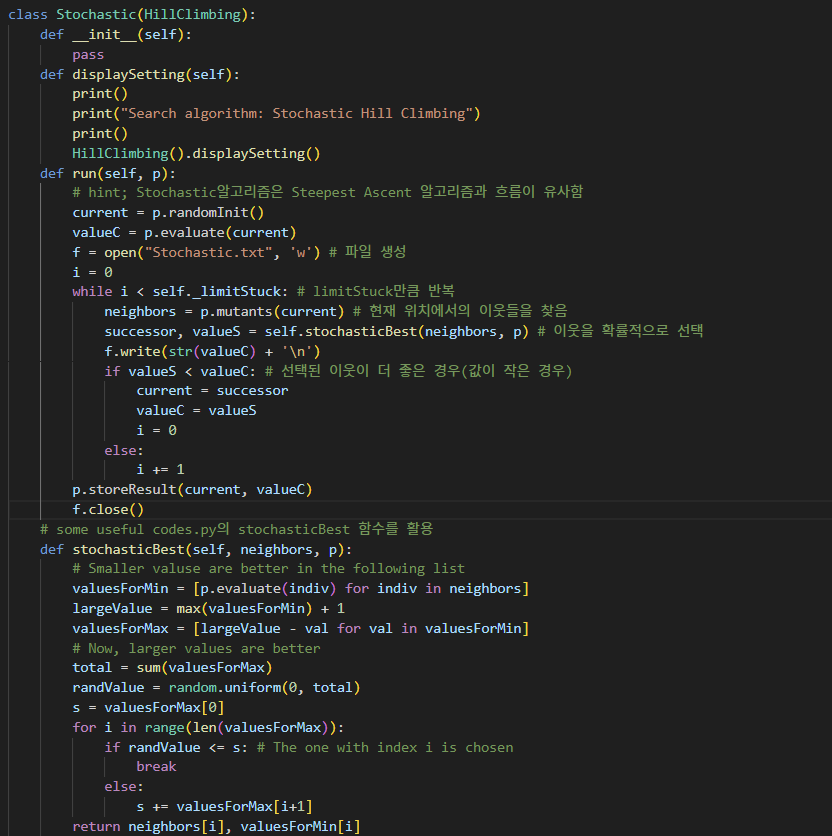






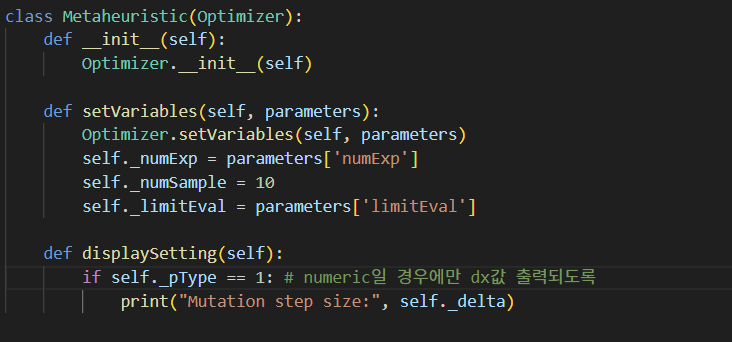
**2.2.4 Stochastic(Hillclimbing)**

Hillclimbing class를 상속하며 displaySetting, run, stochasticBest method를 정의한다. Stochastic알고리즘은 Steepest Ascent 알고리즘과 흐름이 유사하다는 점을 활용해서 run 메소드를 구현하고 이곳에서도 앞에서와 마찬가지로 매 iteration마다 결과를 저장해야하기 위한 코드를 함께 구현한다. stochasticBest 메소드는 함께 제공된 some-useful- codes.py를 참고하였다.



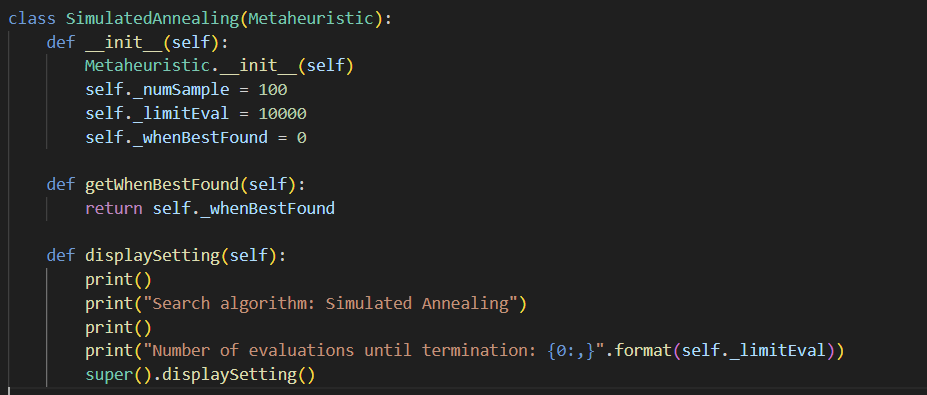
**2.2.5. MetaHeuristics(Optimizer)**

Optimzer를 상속받는 클래스로 앞선 클래스들과 마찬가지로 setVariables를 통해 변수를 저장하고 출력 형식에 맞게 출력되도록 displaySetting를 설정해준다.

****

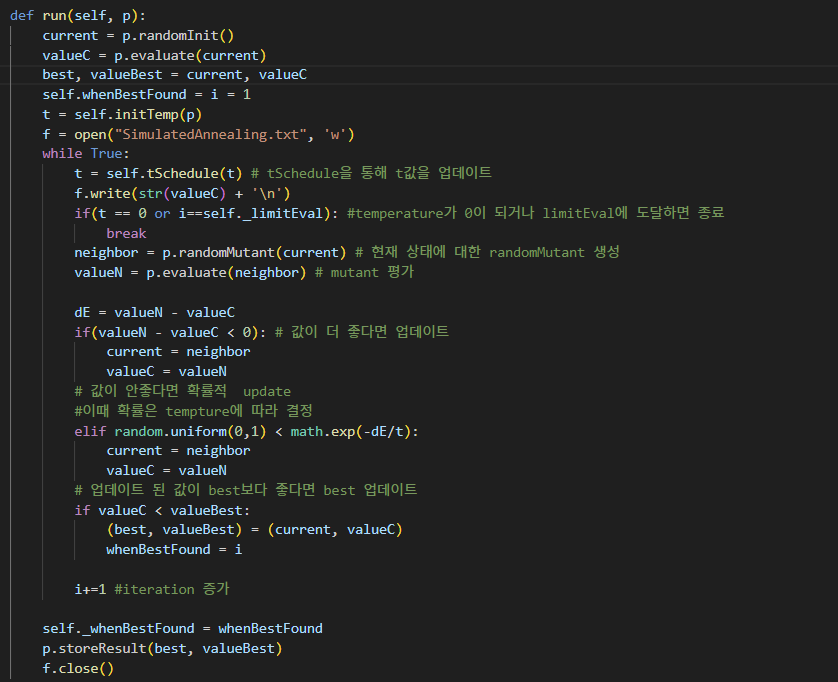
**2.2.6. SimulatedAnnealing(MetaHeuristics)**

이 클래스는 MetaHeuristics class를 상속받으며 numSample 변수를 가지며, displaySetting, run, initTemp, tSchedule method를 가진다. 이때 initTemp, tSchedule method는 some-useful-codes.py에서 주어졌기 때문에 설명을 하지 않고 넘어가도록 하겠다.



Run 메소드에 대해서 자세히 설명하자면 tSchedule을 통해 t를 업데이트하고 현재 상태에 대한 randomMutant를 생성한다. 그 다음 p.evaluate를 통해 mutant값을 평가하고 mutant의 값이 좋다면 무조건 업데이트를 하고 그렇지 않은 경우에는 확률에 따라 업데이트하게 된다. 즉 값이 더 나쁘더라도 확률적으로 변경될 수 있다는 것을 의미한다.

그 다음 update된 값이 best값보다 좋으면 best update를 진행하도록 코드를 구현해야한다.

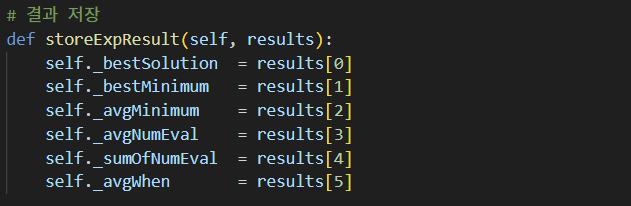


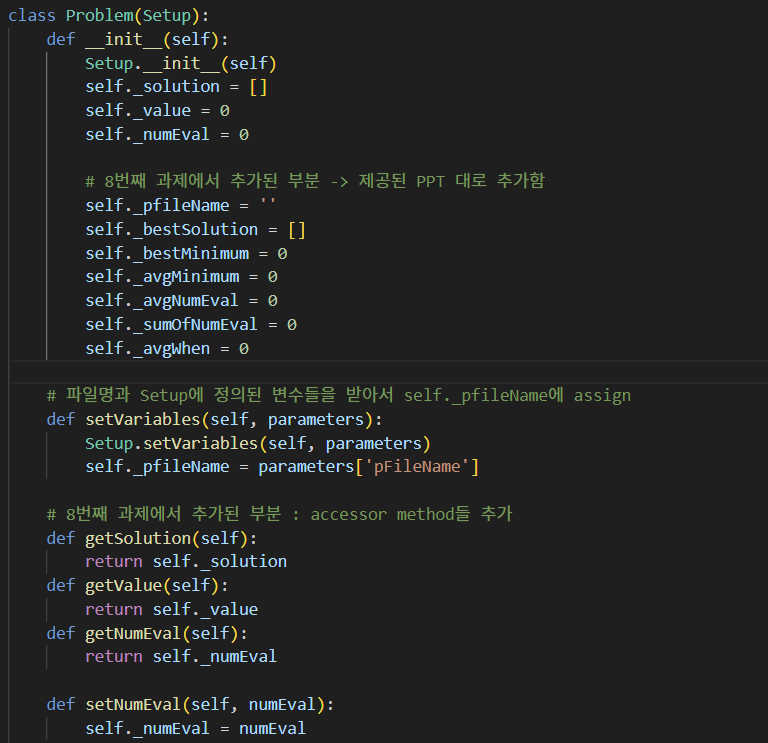
**2.3. problem.py**

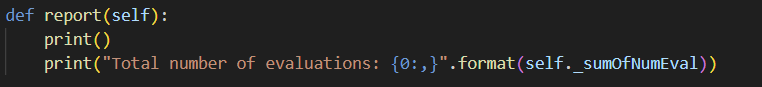
**2.3.1. Problem class**

Problem class는 setup class를 상속한다. 이 클래스는 파일의 이름, bestSolution 등과 같은 변수를 저장한다. 그리고 setVariable 메소드를 통해 파일명과 Setup에서 정의하는 변수들을 설정한다. Solution과 value, numEval에 접근하기 위한 accessor 메소드로 구현하며 numEval의 값을 초기화 시켜주기 위한 setNumEval이라는 메소드도 구현한다. 마지막으로 실험의 결과를 저장할 storeExpResult라는 메소드까지 구현하면 된다.  
또한 공통적으로 마지막에 출력되는 evaluation의 개수를 출력하는 것을 sumOfNumEval을 사용해 출력 형식에 맞게 출력되도록 코드를 작성한다.

이때 변수를 저장하는 이유는 random-restart방식을 사용하기 때문이다.





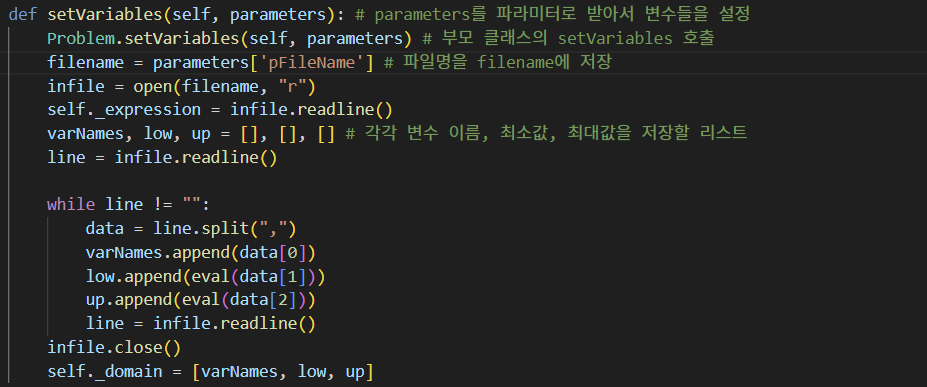


**2.3.2. Numeric class**

변경된 부분에 대해서만 설명해보면

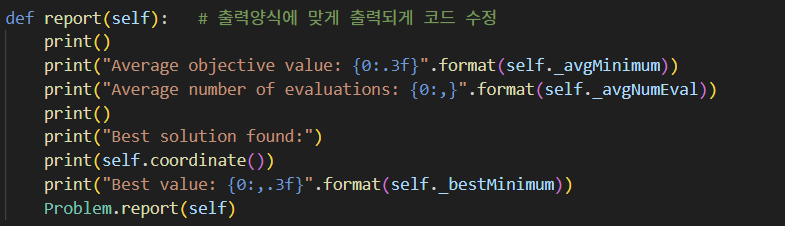
- setVariables(parameters)

이전과는 다르게 parameters라는 것을 파라미터로 추가로 받게 구현한다. 그리고 Problem class의 setVariables를 호출하여 필요한 변수들을 정의한다.



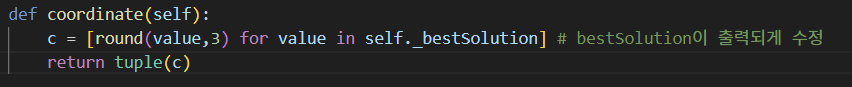
- report

출력 양식에 맞게 출력되도록 코드를 설정한다. 이때 Numeric problem을 풀 때 출력되는 형식은 첨부된 pdf의 출력 형식을 참고하였다.



- coordinate

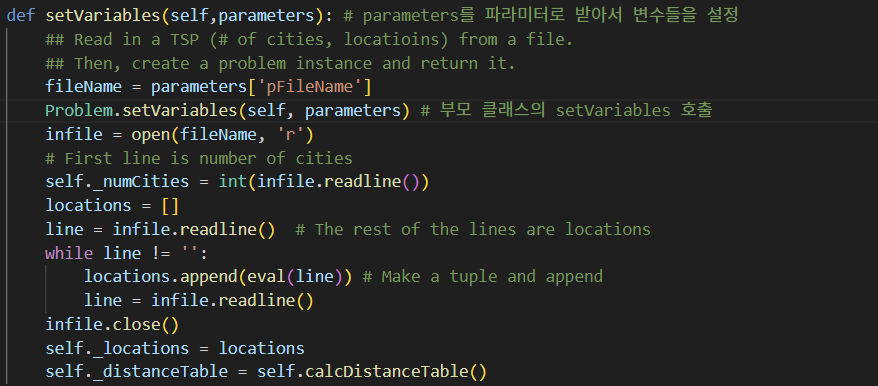
이 메소드에서는 이전에는 그냥 solution을 return했다면 이번 과제에서는 bestSolution을 return하도록 코드를 수정하였다.



**2.3.3. Tsp class**

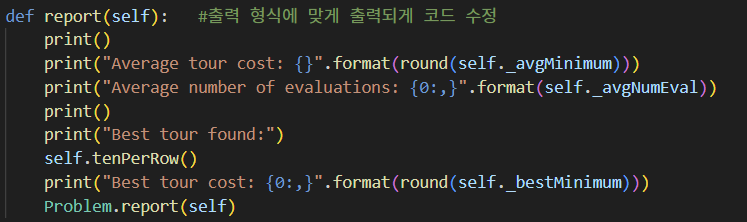
Numeric class와 흐름이 유사하게 이전 코드로부터 약간의 추가 및 수정을 해주면 된다.

- setVariables(parameters)



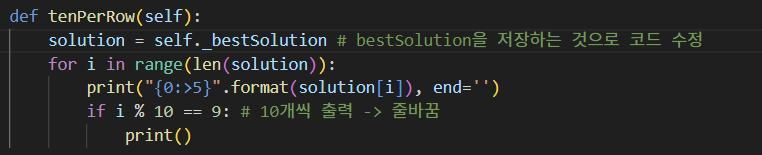
- report

출력 형식에 맞게 출력되도록 하였는데 TSP의 경우는 조교님과 함께 진행한 실습 ppt의 출력 형식을 참고하였다.



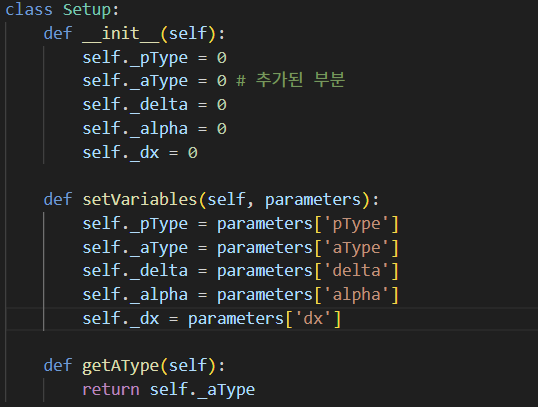
-tenPerRow

Numeric의 coordinate와 마찬가지로 이전까지 리턴했던 solution값 대신에 bestSolution을 리턴하도록 코드를 수정한다.



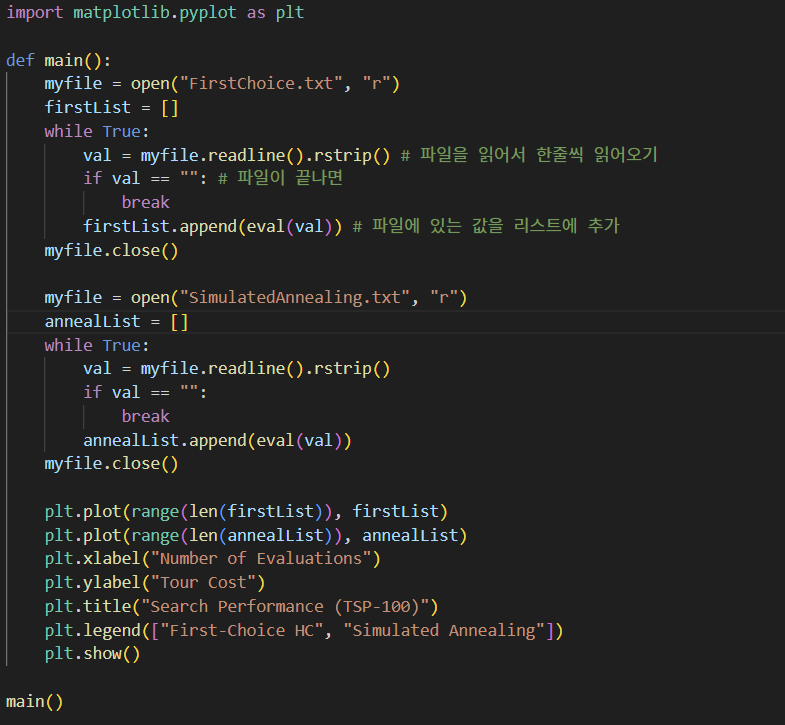
**2.4 setup.py**

Setup class에서는 aType이라는 것이 추가되고 aType을 get하기 위해 getAType도 추가되게 된다. 또한 변수를 Parameters에 저장된 값으로 저장하기 위해 setVariables도 추가해준다.



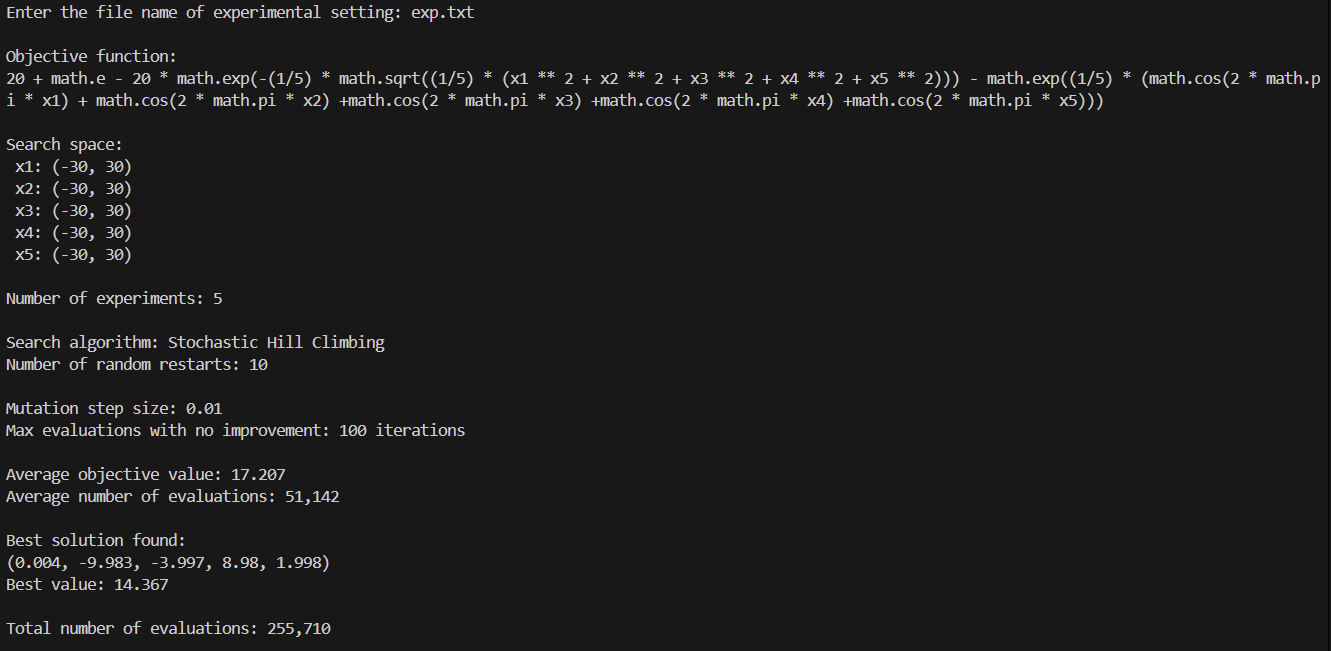
**2.5 plot.py**

Run method를 통해 생성된 txt파일을 이용해 변화를 시각화하기위한 코드이다. Firstchoice와 SimulatedAnnealing에서 numEval에 따른 Current Value를 시각화하는 것이 과제였기 때문에 이 2개이 파일로 한정하여 코드를 작성하였다.

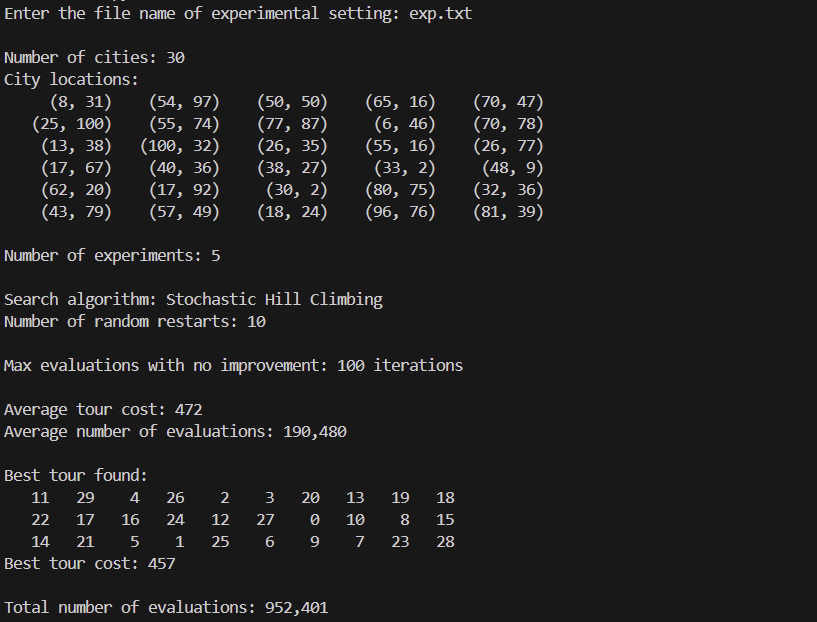


**2.6 실행 결과**

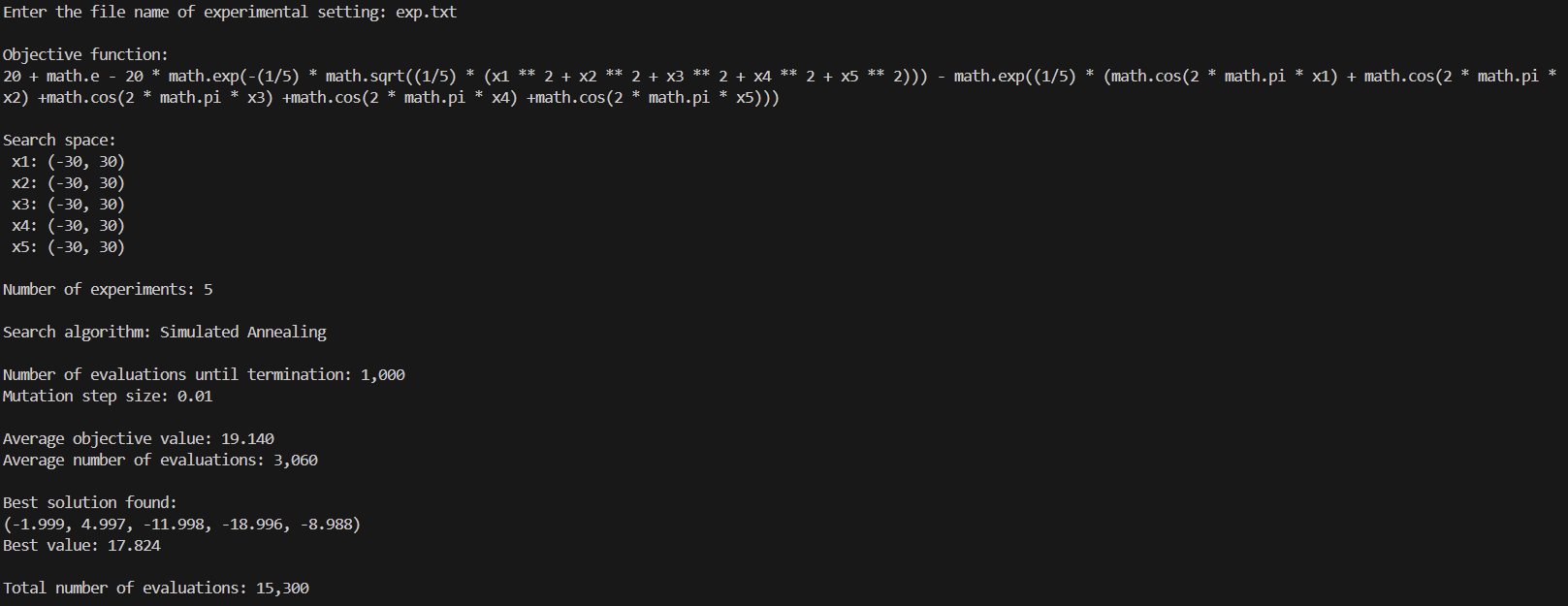
**- Stochastic – Numeric : Ackley.txt**



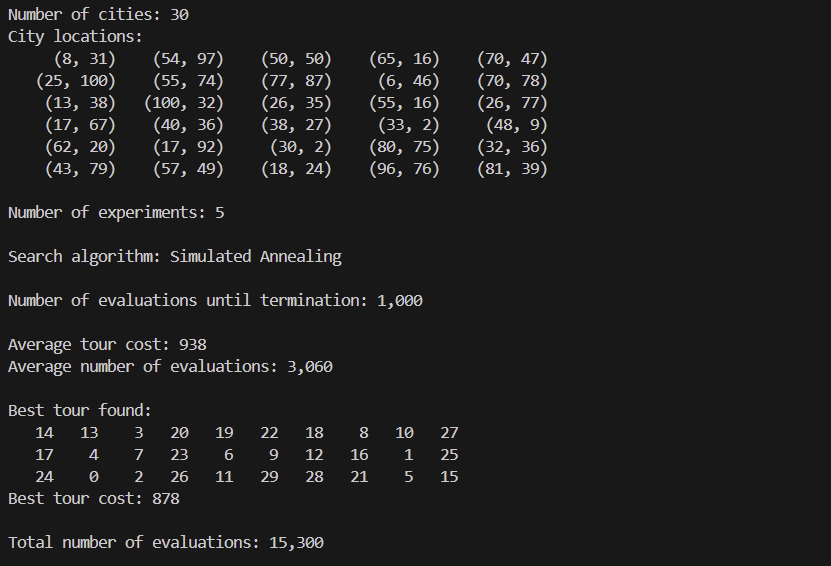
**- Stochastic – TSP : tsp30.txt**



**- Simulated Annealing – Numeric : Ackley.txt**

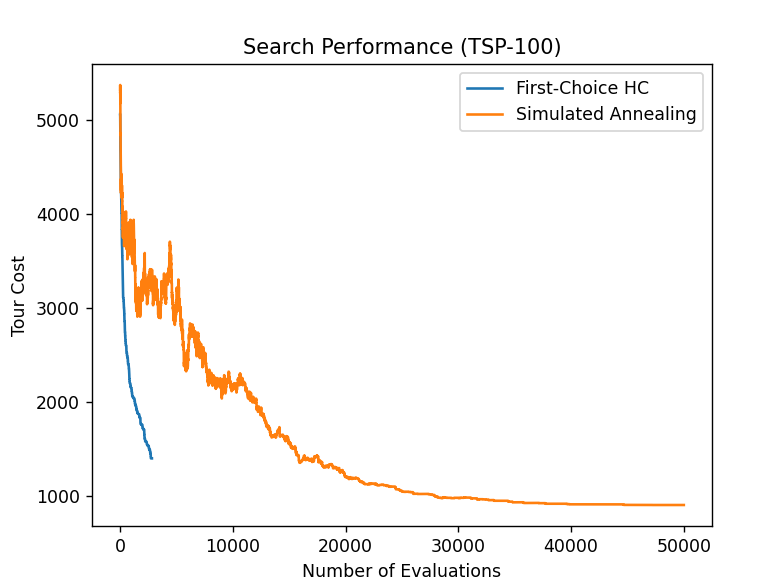


**- Simulated Annealing – TSP : tsp30.txt**



**- Plot.py**

Tsp100.txt사용(limitStuck = 100, numeval = 50000 -> 이 값이 exp.txt에 적힌 초기값임)



3. 결론

**First-Choice HC**의 경우 초기 단계에서 비용이 급격히 감소하는 것을 볼 수 있다. 그러나 특정 시점 이후 더 이상의 개선이 이루어지지 않는다는 것을 확인할 수 있다. 이는 Hill Climbing 알고리즘의 특성 상 더 나은 해를 찾지 못하고 local minimum에 빠진 상태라고 해석할 수 있다.

**Simulated Annealing**의 경우 초기에는 불안정한 경로 비용 감소를 보이지만, 시간이 지남에 따라 비용이 점진적으로 감소하는 것을 확인할 수 있다. 마지막에는 First-Choice HC보다 더 낮은 비용으로 도달하게 된다. 즉 Simulated Annealing은 탐색이 진행될수록 점진적으로 최적해에 가까워지는 특성이 있다고 할 수 있다.

정리하자면 First-Choice HC는 빠르게 해를 찾지만, 탐색이 제한적이고 local minimum에 빠질 확률이 낮은 cost의 경로를 찾지 못한다. 반면 Simulated Annealing은 초기에는 불안정하고 오히려 cost가 높아지는 방향으로 이동하기도 하지만, 더 많은 평가 횟수를 통해 꾸준히 비용을 낮추며 최적화된다. 결과적으로 First-choice HC보다 더 낮은 cost에 최종적으로 도달하게 됨을 확인할 수 있다.