AIP

HW 09 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202255535

이름 : 김진우

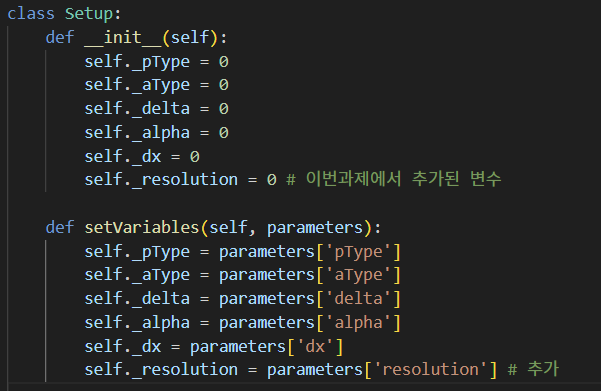
1. 서론

이번 과제에서는 GA알고리즘을 추가해야 한다. GA알고리즘은 생물체가 진화해가는 모습을 모티브로 하여 최적의 방법을 찾아내는 알고리즘으로 하나의 모집단에서 individual에 crossover과 mutation을 통해 최적의 solution을 찾아내는 알고리즘이다. 또한 이제까지 설계한 모든 알고리즘을 numeric/tsp 모든 문제에 적용시켜보고 파라미터들을 바꿔가며 값을 표로 정리한 후 분석까지 진행해야 한다.

2. 본론

**2.1 setup.py**

Resolution 값을 setup 클래스에 추가하고 이를 setVariables에도 값 할당을 해준다.

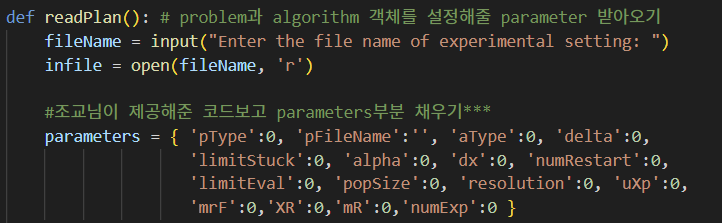


**2.2. main.py**

GA알고리즘을 추가하였기 때문에 이에 맞게 main도 수정을 해야한다.

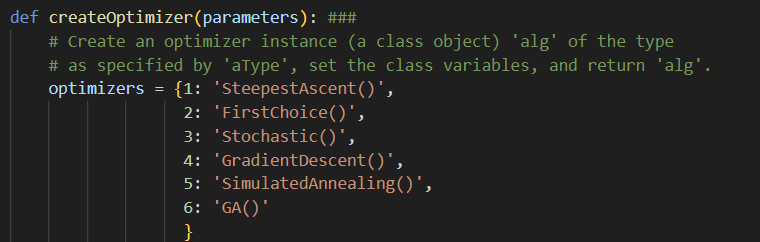
2.2.1. readPlan()

GA에 사용되어 새롭게 추가된 파라미터들을 읽어와야 하기 때문에 파라미터를 추가해준다.



2.2.2createOptimizer()

GA알고리즘도 선택할 수 있게 추가한다

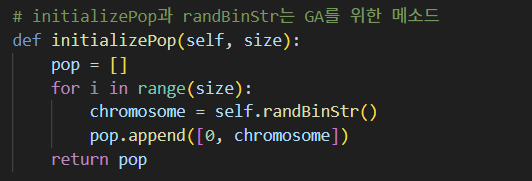


**2.3. problem.py**

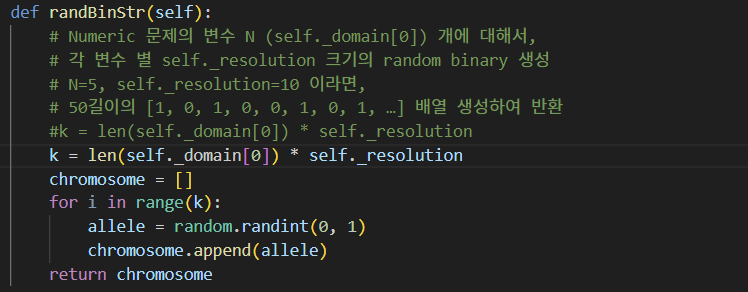
2.3.1 numeric class

GA에서 사용하기 위한 initializePop, randBinStr, evalInd, decode, binaryToDecimal, crossover, uXover, mutation, indToSol 메소드를 새롭게 정의한다.

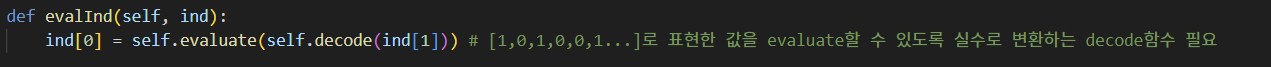
- initializePop(size) : Population size 만큼 individual을 만든다. 이때 randBinStr메소드를 사용하여 chromosome을 만든다.



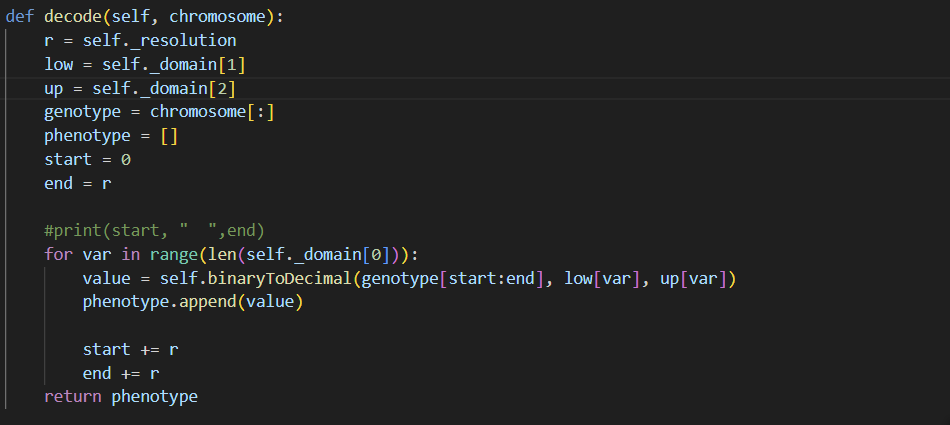
- randBinStr() : “resolution\*변수의 수” 길이만큼의 0/1로 구성된 리스트를 만들고 이를 리턴한다.



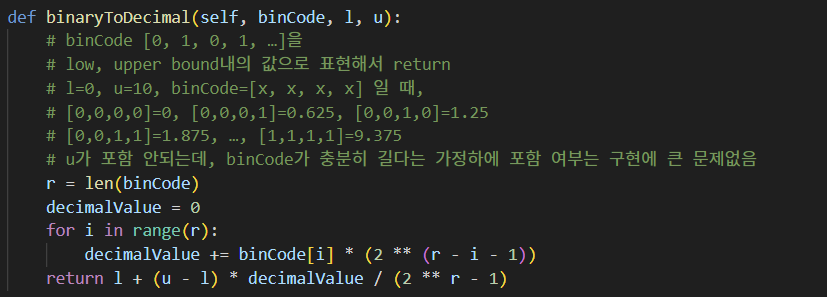
- evalInd(ind) : Individual의 값을 evaluate 메소드를 사용해 계산한다. 이때 [1,0,1,0,0,1...]로 표현한 값을 evaluate할 수 있도록 실수로 변환하는 decode메소드가 필요하다.



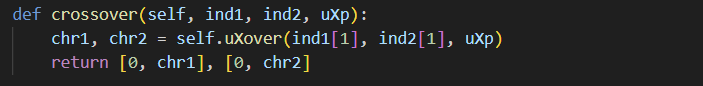
- decode(chromosome) : chromosome은 0과 1로 구성된 리스트인데, 이를 evaluate하기 위해서는 실수로 변환해야한다. 이 과정을 수행하는 것이 바로 decode 메소드이다.



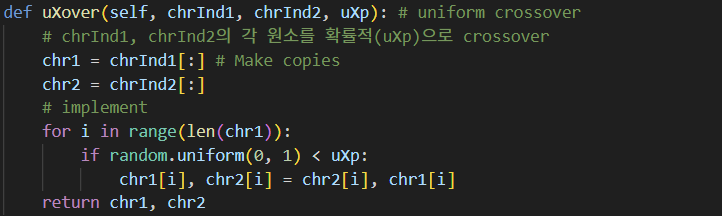
- binaryToDecimal(binCode, l, u) : binCode를 최댓값과 최솟값 사이의 실수로 변환하여 주는 메소드이다.



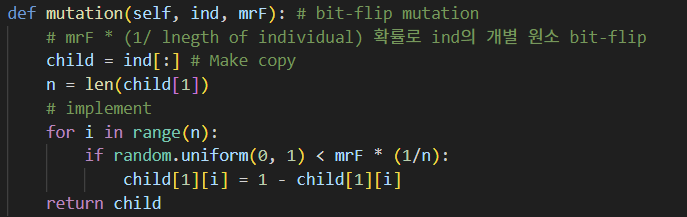
- crossover(ind1, ind2, uXp) : Crossover를 통해 새로운 2개의 individual을 반환한다. 이때 uXover이라는 메소드를 사용한다,



- uXover(chrInd1, chrInd2, uXp) : uniform crossover를 구현한 메소드로 uXp을 기준으로 crossover 진행 여부를 결정한다.



- mutation(ind, mrF) : mrF \* (1/ lnegth of individual) 확률로 mutation 유무를 결정한다.



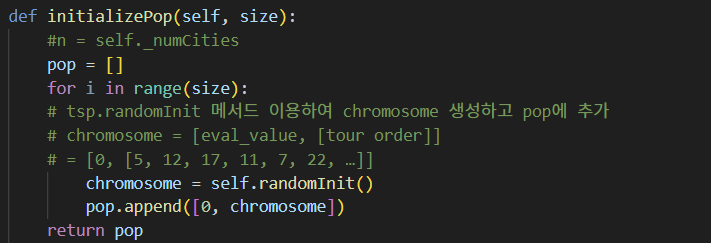
- indToSol(ind) : individual을 solution 형태로 변환한다.



2.3.2 tsp class

Numeric class에서와 마찬가지로 GA에서 사용하기 위한 initializePop, evalInd, crossover, oXover, indToSol, mutation 메소드를 정의한다.

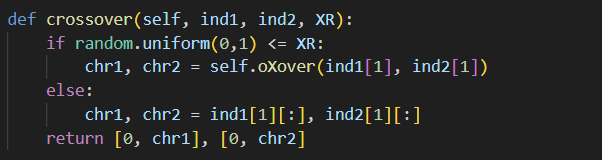
- initializePop(size) : 랜덤한 방문 순서를 pop 크기만큼 만들고, randomInit을 통해 chromosome을 만든 다음 이를 통해 population을 만들어 return을 한다.



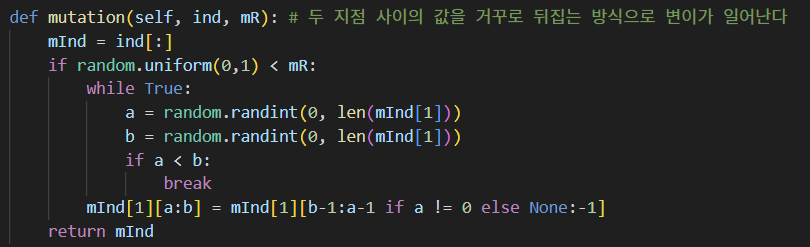
- evalInd(ind)



- crossover(ind1, ind2, XR) : Crossover를 통해 새로운 2개의 individual을 반환한다. 이때 oXover이라는 메소드를 사용한다.



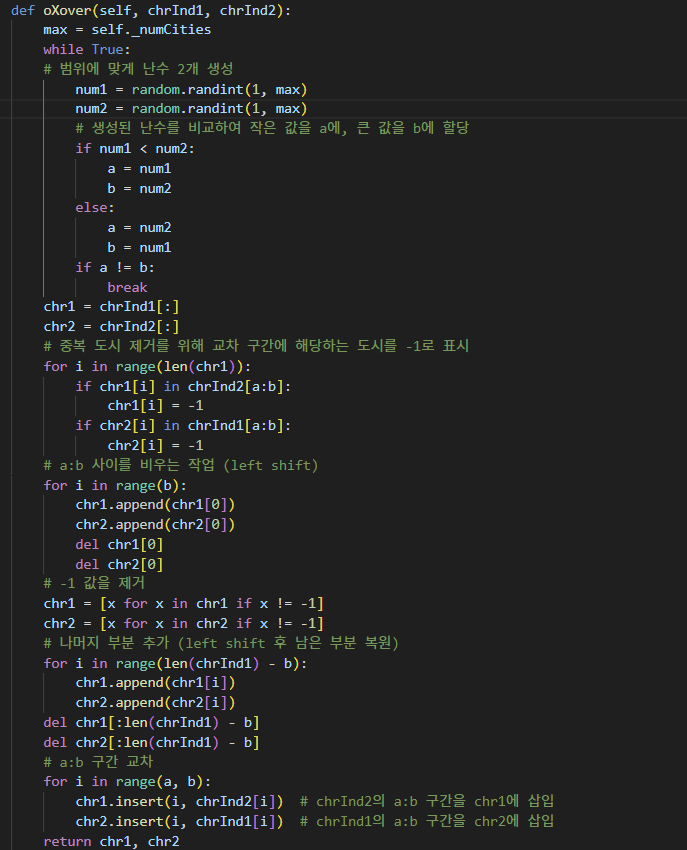
- mutation(ind,mR) : mR값에 따라 mutation을 진행한다.



- indToSol(ind) : individual을 solution형태로 저장한다.



- oXover(chrInd1, chrInd2) : crossover를 진행하는 부분으로 ppt에서 제공된 과정에 맞게 코드를 구현하였다. 자세한 설명은 주석을 참고하길 바란다. 핵심 내용은 중복된 숫자를 제외하기 위한 과정을 거쳐야 하고, 지정된 구간만큼이 서로 교차되어 새로운 chronosome이 만들어지는 것이다.

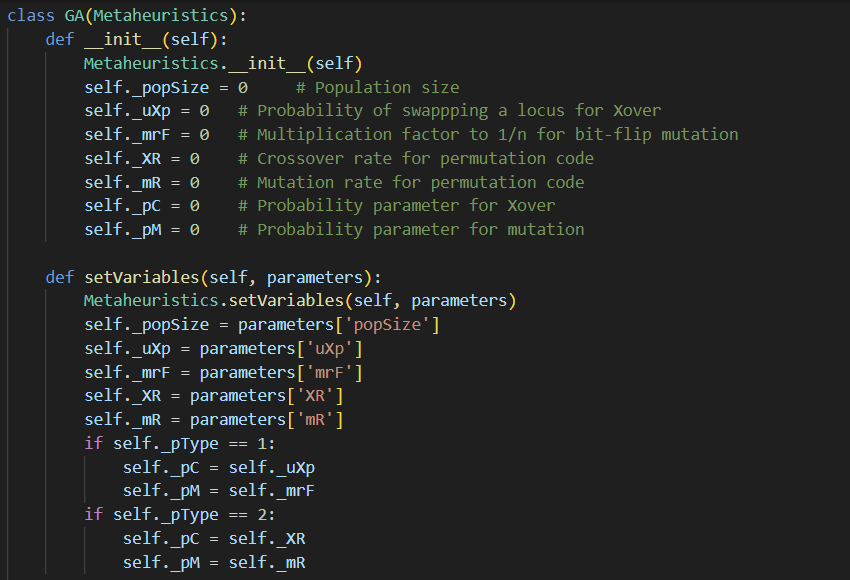


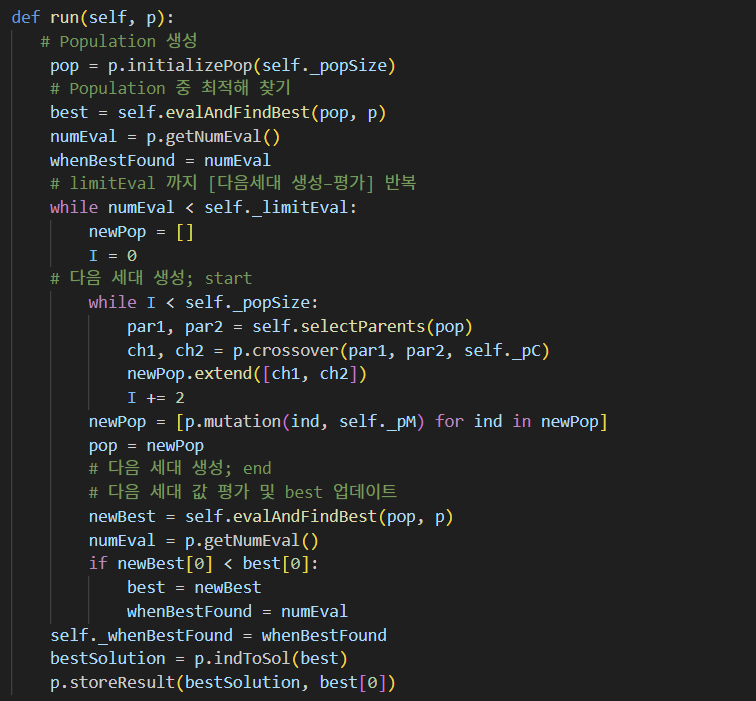
2.4. optimizer.py

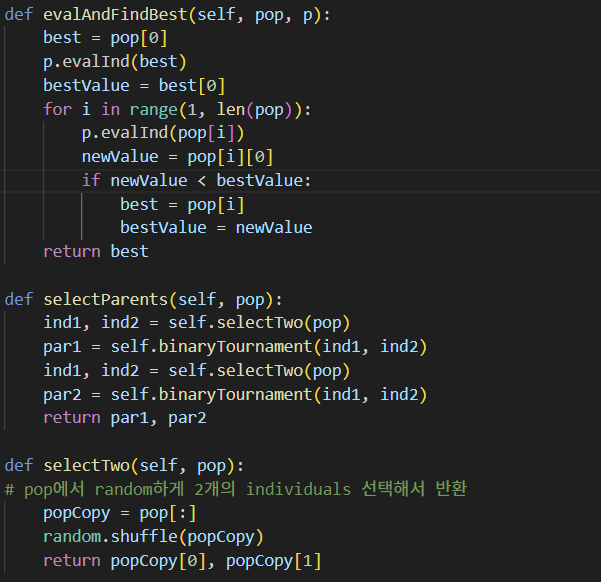
GA 알고리즘에 필요한method들과 생성자, setVariables, run, displaySetting method를 정의한다.

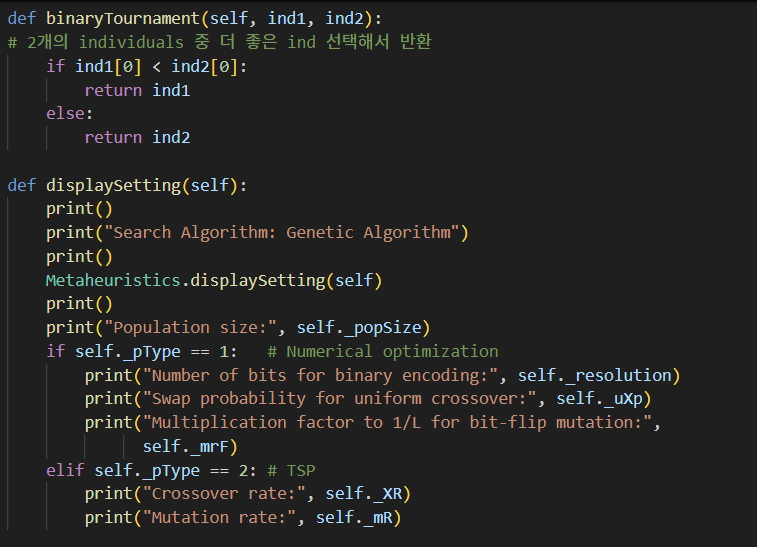
이때 제공된 skeleton code와 실습때 진행한 코드 대부분을 그대로 사용하였다. 그렇기 때문에 코드만 첨부를 하고 설명은 하지 않았다.

1) \_\_init\_\_









3. 결론

numRestart = 10, numExp = 10

**<Numeric>**

Avg object value/ avg num of evaluations/best objective value/avg iteration of find best solution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Convex | Griewank | Ackley |
| Steepest Ascent | Delta = 0.01 | 0.0  794,804  0.0 | 0.227  70.855  0.069 | 17.948  12.261  14.258 |
| First Choice | Delta = 0.01  Limistuck = 1000 | 0.0  281.880  0.0 | 0.182  37,746  0.084 | 16.936  14,335  13.401 |
| Stochastic | Delta = 0.01  Limistuck = 1000 | 0.0  1,963,752  0.0 | 0.160  377,200  0.054 | 18.175  143,267  15.346 |
| Gradient Descent | alpha = 0.01  dx = 10^-4 | 0.0  71,366  0.0 | 0.135  455,454  0.049 | 17.585  5,616  16.245 |
| Simulated Annealing | limitEval = 50000 | 0.002  275,110  0.000  48,346 | 0.409  275,110  0.197  16,259 | 18.952  275,110  17.086  4,867 |
| GA | resolution = 10  limitEval = 50000  popSize = 100  uXp = 0.2  mrF = 1 | 744.964  50,450  4.247  49,490 | 0.530  50,450  0.082  46,620 | 9.435  50,450  0.163  49,230 |

1. Steepest Ascent : 계산양이 적고 빠르게 수렴하나, 복잡한 문제에서는 최적의 값을 잘 찾아내지 못한다.

2. First Choice : 계산 속도는 Steepest Ascent와 유사하지만 steepest ascent에 비해 Griewank와 Ackley에서 약간의 성능 향상이 있다.

3. Stochastic : 실행해본 결과 가장 시간이 오래 걸리고 연산양이 많으며, Convex에서만 최적해를 보장하는 것을 확인할 수 있다

4. Gradient Descent : hill-climbing 알고리즘들 중에서는 계산 효율이 가장 좋고 Convex 및 Griewank에서 우수한 성능을 보인다.

5. Simulated Annealing : 높은 계산량에도 불구하고 상대적으로 다른 알고리즘들에 비해 최적의 solution을 잘 찾아낸 것을 확인할 수 있다.

6. Genetic Algorithm (GA) : 높은 계산량을 필요로하지만 Ackley에서 다른 알고리즘에 비해 압도적 성능을 가짐을 확인할 수 있다.

즉, Numeric 문제를 모든 알고리즘에 대해 수행했을 때, 간단한 문제(Convex)는 Gradient Descent가 가장 효율적이고 복잡한 문제 (Griewank, Ackley): Genetic Algorithm이 최적의 품질을 제공하며, Simulated Annealing도 높은 성능을 발휘한다고 분석할 수 있다. 그러나 시간이 너무 오래 걸리기 때문에 계산 효율이 더 중요하면 Steepest Ascent나 Gradient Descent를, 최적해 찾는 것이 더 중요하다면 Simulated Annealing이나 GA를 선택하면 된다고 결론지을 수 있다.

**<TSP>**

Avg object value/ avg num of evaluations/best objective value/avg iteration of find best solution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Tsp30 | Tsp50 | Tsp100 |
| Steepest Ascent |  | 571  6,650  545 | 807  19,890  757 | 1236  87,001  1,104 |
| First Choice | Limistuck = 1000 | 461  27,674  454 | 614  54,413  596 | 914  127,779  878 |
| Stochastic | Limistuck = 1000 | 459  690,082  454 | 608  1,972,874  597 | 914  127,779  878 |
| Gradient Descent | alpha = 0.01  dx = 10 | 불가능 | 불가능 | 불가능 |
| Simulated Annealing | limitEval = 50000 | 457  275,110  454 33,012 | 610  275,110  598 43,881 | 880  275,110  851 48,216 |
| GA | limitEval = 50000  popSize = 100  XR = 0.1  mR = 0.1 | 1195  50,450  759  46,700 | 2119  50,450  1,318 47,200 | 4420  50,450  2,668 50,270 |

1. Steepest Ascent : numeric에서와 마찬가지로 계산량은 적지만, 복잡한 문제에서는 성능이 크게 저하되는 것을 확인할 수 있다.

2. First Choice : 계산량이 적지만, tsp5과 tsp100에서 steepest ascent에 비해 성능이 향상됨을 확인할 수 있다.

3. Stochastic : 계산량이 매우 크고 실제로도 실행시간이 가장 오래 걸리는 비효율적인 알고리즘이다.

4. Gradient Descent : TSP 문제에서는 미분이 불가능하기 때문에 사용 할 수 없다.

5. Simulated Annealing : 계산량이 적고 품질도 뛰어나다.

6. Genetic Algorithm (GA) : 계산량이 크지만, 성능은 다른 알고리즘들에 비해 좋지 못하다.

즉, tsp문제를 모든 알고리즘에 대해 실행시켰을 때, 대부분의 문제가 optimal solution이거나 이와 비슷한 값을 찾아주는 것을 확인할 수 있었다. 그래도 좀 더 분석하자면 소규모 문제 (TSP30)는 First Choice 또는 Simulated Annealing 방법이 효율적이고 중규모 문제 (TSP50)는 Simulated Annealing이 성능과 계산 효율성에서 균형을 이루고 대규모 문제 (TSP100)는 Simulated Annealing이 적합하다는 결론을 내릴 수 있다.