

알고리즘 프로젝트 HW #4

Knapsack DP / GA

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2018년 6월 24일

학부 정보통신공학과

학년 3학년

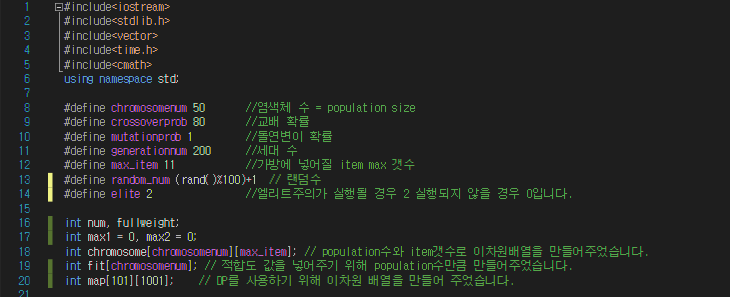
성명 김권중

학번 12141668

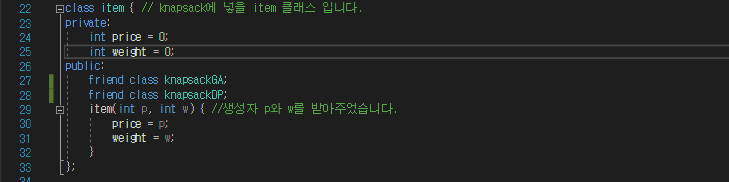
1. 개요

이번 과제는 NP-complete문제 중 하나인 knapsack문제를 DP와 GA를 이용해 비교 분석하고, GA내에서 값을 변경하며 최적의 값을 도출해 내는 프로젝트였습니다. Knapsack 문제를 두가지 방식으로 풀기 위해 DP와 GA 두 가지 방식으로 코드를 짰으며, 공통된 값의 아이템을 넣어 실행시간, 결과값이 맞게 나오는지 확인해 보았습니다.

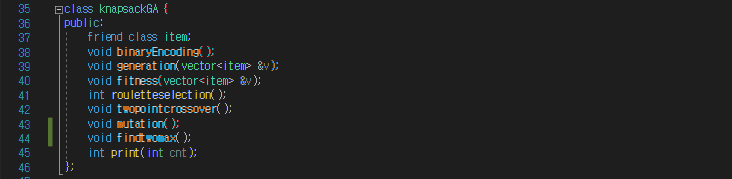
2. 상세 설계내용



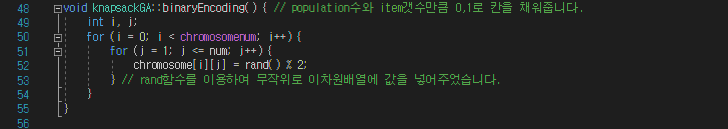
전역변수와 define입니다. Define을 변경하여, 데이터 셋을 조절할 수 있게 하였습니다.



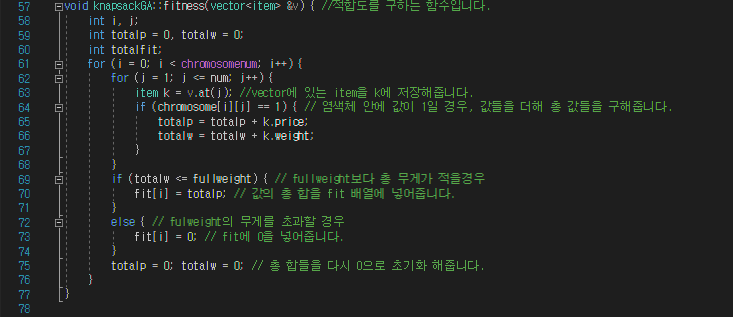
Knapsack에 넣을 item 클래스입니다. Price와 weight을 입력 받을 수 있도록 하였습니다. Friend class를 이용하여 item의 private값에 knapsackDP와 knapsackGA가 접근 할 수 있도록 해 주었습니다.



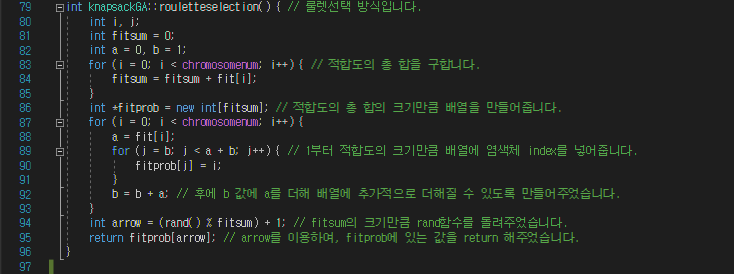
KnapsackGA 클래스입니다.



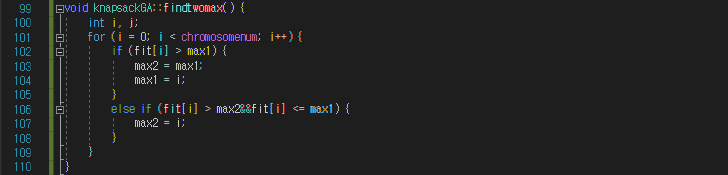
Knapsack의 경우 아이템이 들어가 있는지 확인하기 위해 binaryEncoding기법을 사용했습니다.



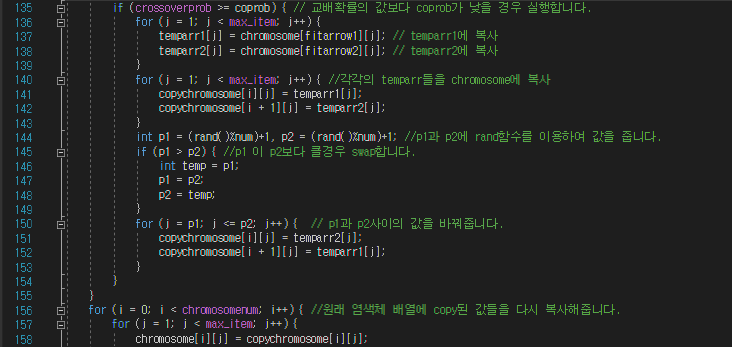
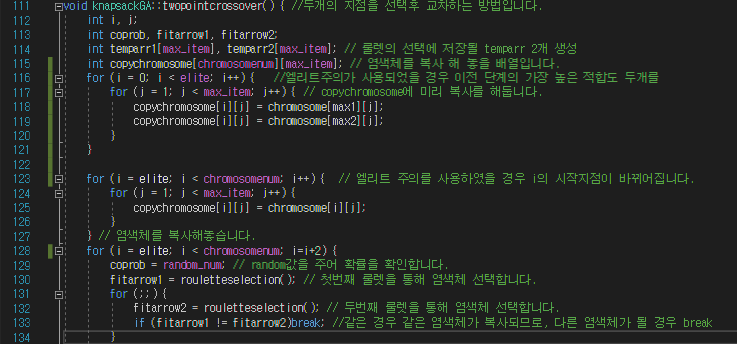
적합도를 부여해주는 부분입니다. 적합도는 fit배열에 저장을 해주었습니다. 적합도를 구하는 방법으로는 배낭에 들어있는 아이템의 총 무게와 총 가격을 구한 후, 총 무게가 주어진 fullweight을 초과 할 경우 적합도를 0으로 넘지 않았을 경우, 적합도를 총 가격으로 해 주었습니다. 루프가 돌 경우 다시 총 무게와 총 가격을 초기화 시켜주었습니다.



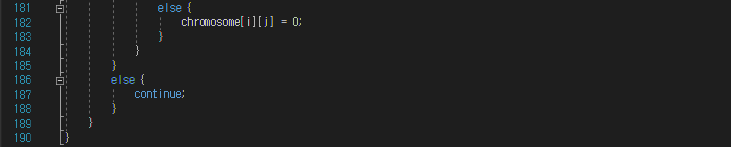
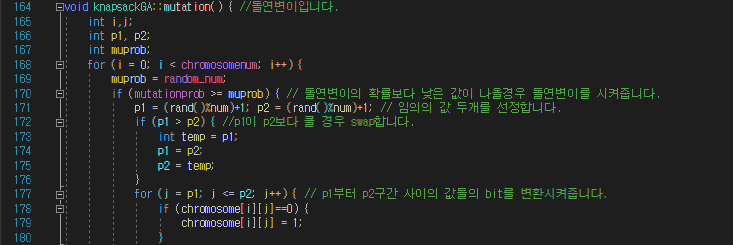
룰렛 선택입니다. 적합도의 총 합을 구합니다. 그 후, fitsum의 크기만큼 새로운 배열을 만들어 주었습니다. 각 fit에 있는 적합도의 크기만큼 fitprob에 인덱스 번호로 칸을 채워주었습니다. Fitsum의 크기만큼 rand 함수를 돌려 fitprob에 있는 수를 return해 주었습니다.



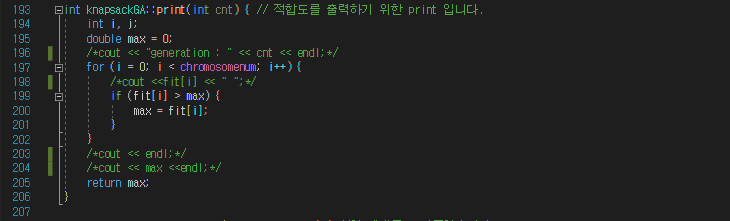
elite주의를 사용할 때, 가장 큰 값 두개를 뽑아 내기 위해 만들었습니다.



Twopointcrossover입니다. 엘리트 주의를 사용할 경우 copy된 염색체 위에서 두번째 칸까지 적합도가 가장 높은 두개의 염색체가 채워집니다. 룰렛 선택을 이용하여, 두가지 염색체를 선택합니다. 이 때 같은 염색체가 선택되지 않도록 fitarrow1, fitarrow2를 설정해주었습니다. 랜덤한 두 위치를 선택하여, 그 사이 값을 교환 해 주었습니다.

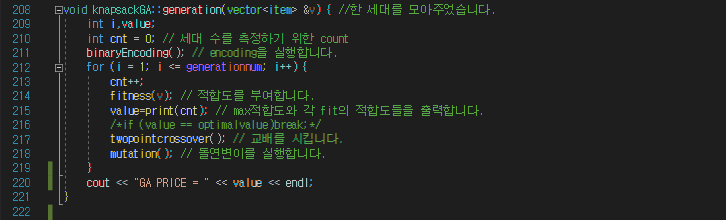


돌연변이입니다. 랜덤한 값 두개를 선택한 후, 그 사이의 값에 0일 경우 1로, 1일 경우 0으로 바꾸어주었습니다.

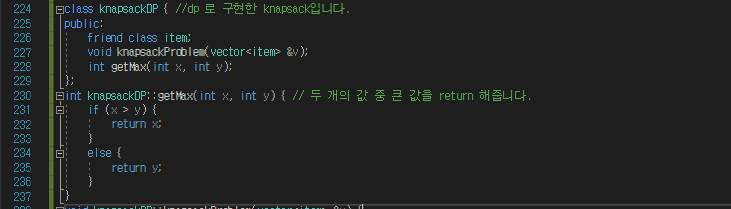


ga알고리즘의 한 세대를 세주며, 각 세대의 적합도와 최대 적합도를 출력 해 주기 위한 함수입니다.

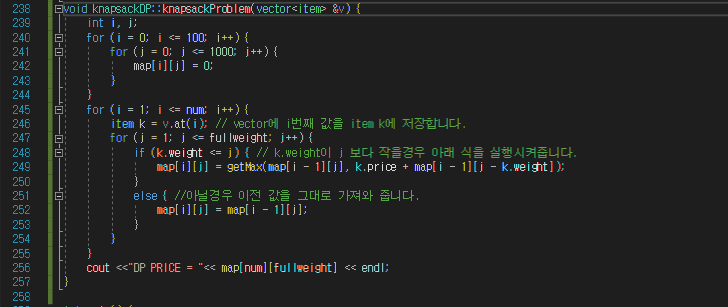
또한 최대 max값을 return하도록 해주었습니다.



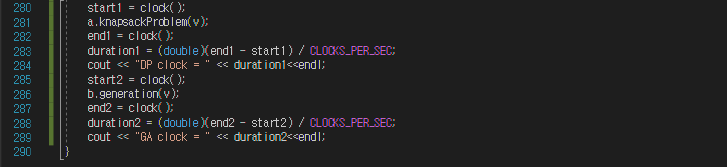
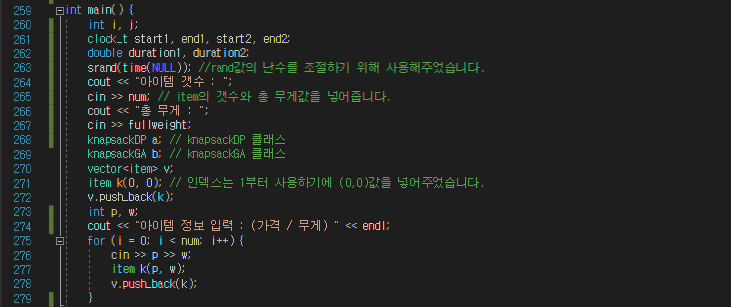
한 세대를 구별 해주기 위해 만들어 주었습니다. 인코딩 -> 적합도 -> 교배 -> 돌연변이 순으로 한 세대가 진행됩니다.



DP로 구현한 knapsack 클래스입니다. getMax의 경우 두 개중 큰 값을 return하도록 만들어 주었습니다.



Map 배열을 0으로 초기화 해 주었습니다. J가 아이템의 무게보다 적을 경우 점화식을 적용해주었습니다.



main함수 입니다.

GA알고리즘에 적용한 기법

Encoding : binary Encoding 기법을 사용했습니다. 아이템이 담겨져 있는지 여부를 확인하기 위해 “1일 경우 넣어있다, 0일 경우 넣어있지 않다” 로 해 주었습니다.

Selection : Roulette wheel selection을 사용했습니다. 전체 적합도의 크기 만큼의 배열 선언 후, 적합도 만큼 배열에 칸을 부여해주었습니다. Rand를 이용하여 랜덤으로 룰렛에 값을 뽑아냈습니다.

Crossover : twopointcrossover를 사용했습니다. 여기서 엘리트주의의 사용여부에 따라 나눠집니다. Elite주의를 사용할 경우, 적합도가 가장 높은 두개의 염색체를 교배시키지 않고 그대로 넘겨주었습니다. Elite주의를 사용하지 않을 경우, 전체 염색체를 가지고 교배를 실시하였습니다. 이때 교배확률을 조정 할 수 있도록 하였습니다. 기본적인 비교 땐 80%의 확률로 사용했습니다.

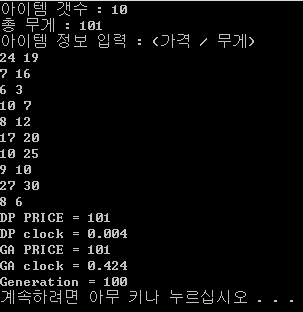
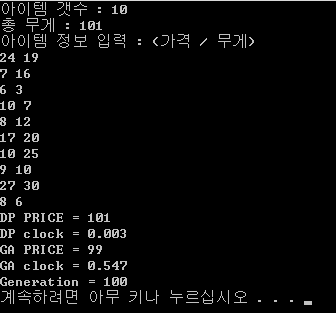
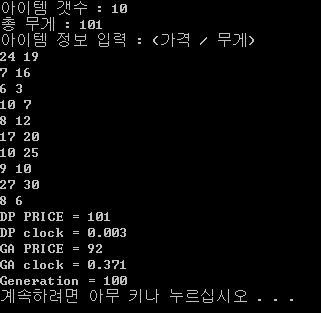
Mutation : 두개의 랜덤지점을 선택 한 후, 그 사이의 값의 비트를 바꾸어주었습니다. 이때 돌연변이 확률을 조정 할 수 있도록 하였습니다. 기본적인 비교 땐 1%의 확률로 사용했습니다.

3.실행화면

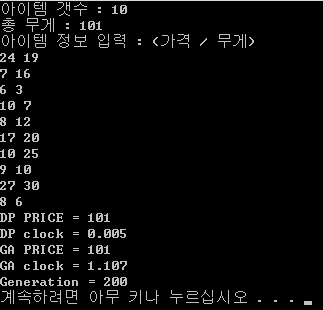
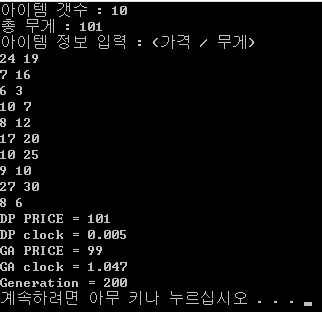
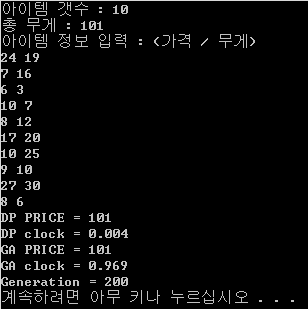
DP와 GA를 비교하며, GA의 정확도를 확인하기 위해 generation수를 증가시키며 비교해 보았습니다.

1. population 50, crossover 80, mutation 1

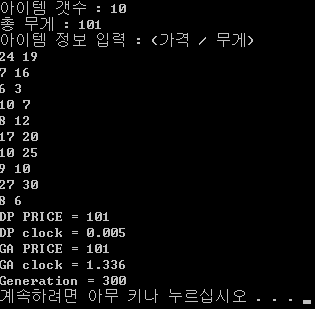
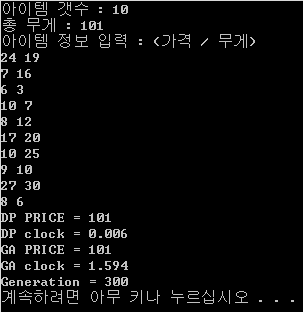
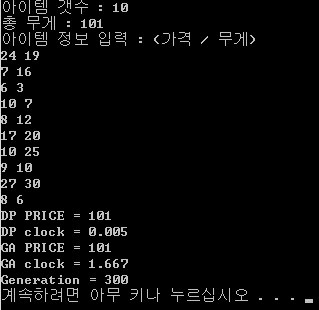
Generation 100



Generation 200



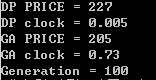
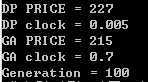
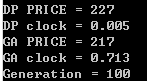
Generation 300



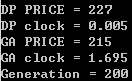
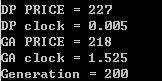
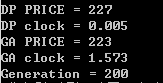
DATA 1 그래프.

2. population 50, crossover 80, mutation 1

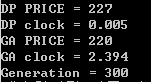
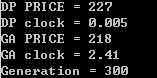
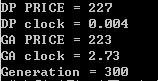
1)generation 100

2)generation 200

3)generation 300

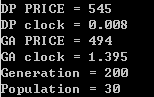
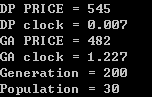
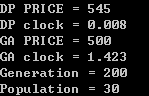
  

DATA 2 그래프

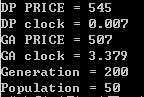
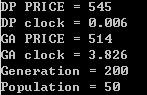
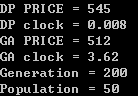
이번엔 generation은 고정으로 하고, population의 수를 변동해보았습니다.

3. generation 200, crossover 80, mutation 5

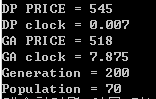
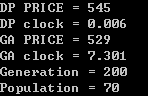
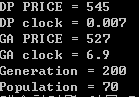
1) population 30

2) population 50

3)population 70

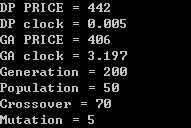
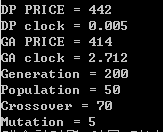
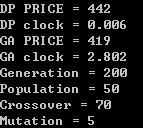
  

DATA 3 그래프

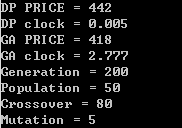
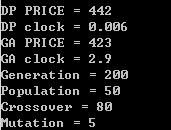
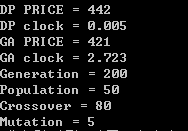
이번엔 교차율을 조정해가며 데이터 셋을 실행시켜보았습니다.

4.generation 200, population 50, mutation 5

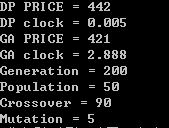
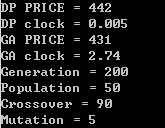
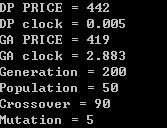
1) crossover 70

2) crossover 80

3) crossover 90

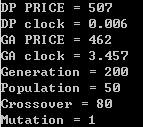
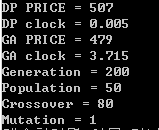
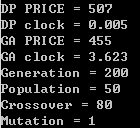
  

DATA4 그래프

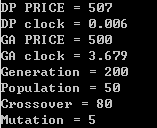
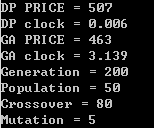
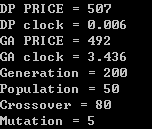
이번엔 교차율을 조정하여 데이터 셋을 실행시켜보았습니다.

5. generation 200, population 50, crossover 80

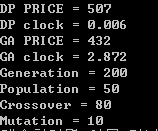
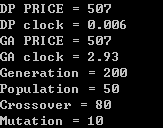
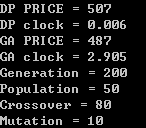
1)mutation 1

2)mutation 5

3)mutation 10

DATA5 그래프

4. 분석 및 결과

DP알고리즘과 GA알고리즘의 코드를 실행시켜 각각의 데이터 셋을 넣고 실행을 해 보았습니다. DP알고리즘의 경우 주어진 아이템들에 대해서 최대값의 가격이 나오는 반면, GA알고리즘의 경우 무게를 넘지 않는 선에서 반복적인 교배와 돌연변이를 통해 최적의 값을 찾아나가는 것이었습니다. 따라서 GA알고리즘의 경우 정확한 값이 나오지 않을 수도, 나올 수도 있었습니다.

총 5가지의 경우를 가지고 데이터 비교를 해보았습니다. 첫번째로 변화를 준 것은 generation 횟수를 증가 시키는 것 이었습니다. 100, 200, 300의 세대수를 가지고 비교를 해보았습니다. 예상한 결과는 세대수가 증가 할수록, 여러 번의 교차와 돌연변이가 일어나기에 수렴값이 DP알고리즘의 측정치와 비슷해 질 것이라고 생각했습니다. 예상한 것과 같이 세대수가 늘어날수록 GA알고리즘의 값이 측정치와 비슷하게 수렴해갔습니다.

2번째로, 세대수를 갖게 증가시키지만 아이템의 개수를 늘려보았습니다. 아이템의 개수가 늘어 날 경우 1, 0으로 구성되어야 할 경우의 수가 증가하기에 이전 단계에 비해 수렴값을 찾는데 어려움이 있을거라 생각했습니다. 실행결과, 세대수가 증가 할수록 수렴값과는 가까워지지만, DP알고리즘의 정확한 측정값에 도달하지는 못하였습니다.

3번째로, 세대를 고정하고 population 즉 염색체 수를 증가 시키며 실행해보았습니다. 염색체 수가 늘어날 경우 교배되고, 돌연변이가 생성되는 것이 많기 때문에, 원하는 수렴치에 가까워지는 것을 방지하고 측정치와 가까워 질 것이라고 생각했습니다. 30, 50, 70의 population을 주어 실행을 해보았습니다. 예상했던 것과 같이 population값이 늘어 날수록 수렴값에 가까워지는 것을 확인할 수 있었습니다.

4번째로, 교배율을 조정해보았습니다. 70,80,90의 교배율을 가지고 측정을 해보았습니다. 이 경우는 어떠한 값에 원하는 측정치와 가까워지는지 예상 할 수가 없었습니다. 실행결과 70의 경우 가장 적은 수렴값을 갖게 되었으며, 80과 90은 거의 비슷한 수치로 수렴하는 것을 확인할 수 있었습니다.

5번째로, 돌연변이 확률을 조정해보았습니다. 돌연변이의 확률이 너무 높지 않은 선에서 가장 높은값이 측정치와 가장 가깝게 나올 거라 예상했습니다. 1, 5 ,10의 확률을 가지고 실행을 해보았습니다. 실제 돌려본 결과 5의 경우가 가장 크게 나왔지만, 수렴값과 측정값이 같은 경우는 10이었습니다. 하지만 10의 경우 측정치의 범위가 다른 것에 비해 넓은 것을 알 수 있었습니다. 1의 경우 돌연변이확률이 낮기 때문에, 원하지 않는 설익은 수렴이 나타나는 것을 알 수 있었습니다.

위의 실행결과로 GA알고리즘의 경우 세대수를 많이 할수록, population을 증가 할수록, 교차율은 80~90 % 돌연변이의 경우 5%가 적당한 것을 알 수 있었습니다. 하지만 generation을 많이 할수록, population을 증가시킬수록 계산 구간이 늘어나기에, GA알고리즘의 시간이 크게 늘어나는 것을 알 수 있었습니다.

GA알고리즘의 경우 OPTIMAL VALUE를 통해 DP알고리즘으로 증명된 답을 주고 적합도중에 MAX값이 OPTIMAL VALUE랑 같을 경우 멈추는 방식을 통해 값을 알 수 있었지만, 이 경우 DP알고리즘을 통해 이미 답을 알고 있어야 했기에, 하지 않았습니다. 그래서 각 세대의 크기 제한을 주고, 세대가 끝났을 때 적합도를 통해 결과를 도출하였습니다. 따라서 시간을 비교하기에 무리가 있었습니다.

DP알고리즘의 경우 대부분의 경우에서 0.01을 넘는 시간이 나오지 않았습니다. 성능이 좋다 보니, GA알고리즘과 비교할 수가 없었습니다.