유전 알고리즘 최종 보고서

- 유전알고리즘을 통한 지수귀문도 풀이 -

인공지능연구실(이상근 교수님)
석사과정
손수현

목 차

1. 문제 정의

지수귀문도(Hexagonal tortoise problem)는 조선시대의 최석정이 만든 마방진이다. 지수귀문도는 기본적으로 육각형이 맞물려있고, 일렬로 정렬된 가운데 육각형을 기준으로 위아래로 개수를 하나씩 줄여나가 전체적으로 마름모 형태로 표현할 수 있다. 아래 그림은 위키백과에서 가져온 차원(dimension)이 3인 지수귀문도에 대한 예시이다.

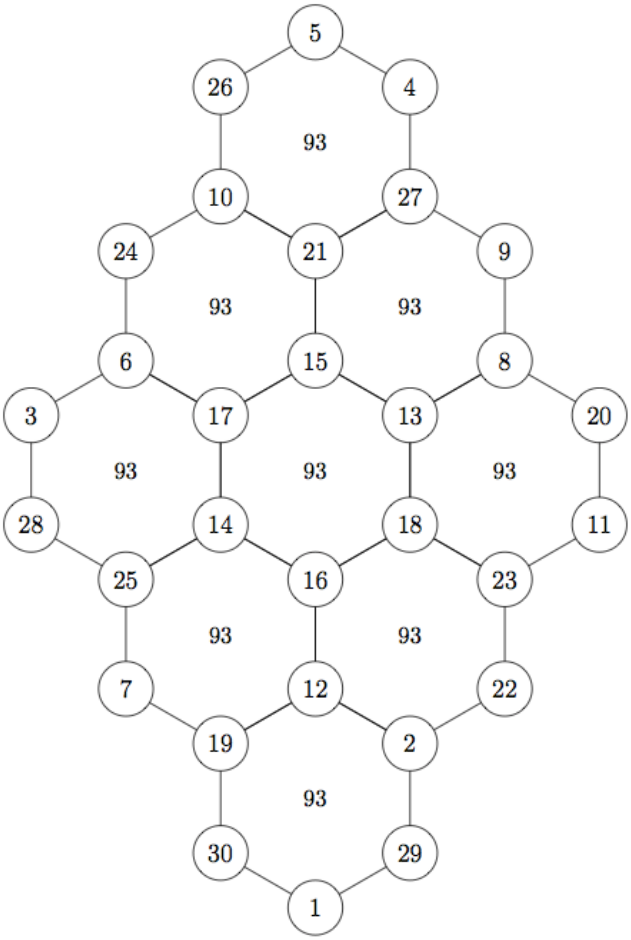


그림 1. 차원(dimension)이 3인 지수귀문도의 예시

지수귀문도는 육각형의 꼭짓점에 대응하는 숫자들의 합이 모든 육각형 꼭짓점에 배치된 수들의 자리를 바꾸어 93이 아닌 다른 총합이 나오게 할 수도 있다. 그 합은 77부터 109까지 가능하다.[1] 본 과제에서 해결할 문제의 지수귀문도 모양은 최석정의 지수귀문도 형태를 따른다.

1) https://ko.wikipedia.org/wiki/지수귀문도

2. 구현 연산자

1) 표현(Representation)

지수귀문도의 표현은 논문 [1]처럼 나타내었고, 아래 그림과 같다.

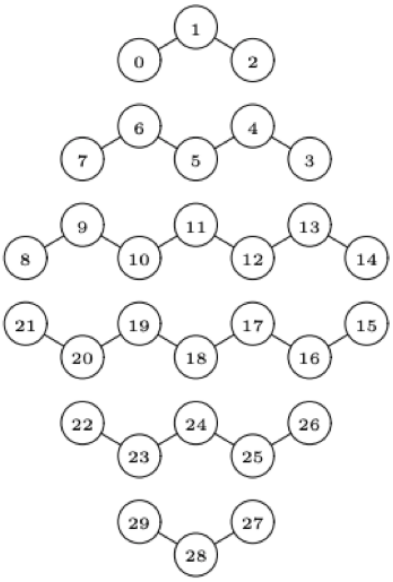


그림 2. 지수귀문도 표현 방법

2) 선택(Selection)

선택 방법은 모집단(population)에서 랜덤으로 2개의 부모를 선택했다.

3) 적합도(Fitness)

적합도를 계산하는 방법은 [1]의 방법과 동일하다. 2가지의 방법을 사용하는데, 기본적으로 해의 품질을 평가하는데 사용하는 적합도 식은 아래와 같다.

$$\text{Fitness} = -\sigma^2$$

$$\sigma = \sqrt{\sum_{i=1}^{m} \frac{(H_i - \mu)^2}{m}}$$

m은 육각형의 총 개수, H_i는 i번째 육각형의 합, μ는 전체 육각형의 평균을 의미하고, 해(Solution)를 찾게 되면, 적합도는 0이 된다.

두 번째 적합도 평가 방법은 세대를 거쳐 오래 살아남은 유전자에게 패널티를 부과하기 위한 실제 적합도(Effective fitness)이고, 식은 아래와 같다.

$$\text{Effective Fitness} = \text{Fitness} - \frac{1}{10H} * \text{Age}$$

Age는 이전 세대의 유전자가 교체(Replacement) 연산으로 없어지지 않고, 다음 세대의 모

집단에 존재할 경우 증가하는 변수다. 구현할 때 파이썬의 사전(Dictionary) 자료구조를 사용해 구현했다. 실제 적합도를 사용하게 되면, 모집단이 지역 최적화에 갇혔을 때 지역 최적해의 적합도를 조금씩 떨어뜨림으로써 결국 교체 연산으로 유전자가 없어지게 된다. 여기서 Age 앞에 곱하는 수는 하이퍼 파라미터로 볼 수 있고, 이 값이 클수록 교체되지 않은 유전자에 더 많은 패널티를 부과할 수 있다. 다른 자료에서는 0.1을 곱해 강한 패널티를 부여하기도 했지만, 본 과제에서는 적은 패널티를 부과해 가장 좋은 유전자를 오래 살아남게 했다. 큰 패널티를 부여할 경우 가장 적합도가 높은 유전자는 세대가 지날수록 바뀌는 걸 확인할 수 있었지만, 차원이 높아질수록 해당 지역 최적해에 대한 탐색이 충분히 되어야한다고 생각해 본 과제에서는 H를 차원에 해당되는 육각형의 개수로 실험을 수행했다. 아래 그림은 나이가 120인 유전자에 대한 실제 적합도와 일반 적합도를 구해본 예시이다.

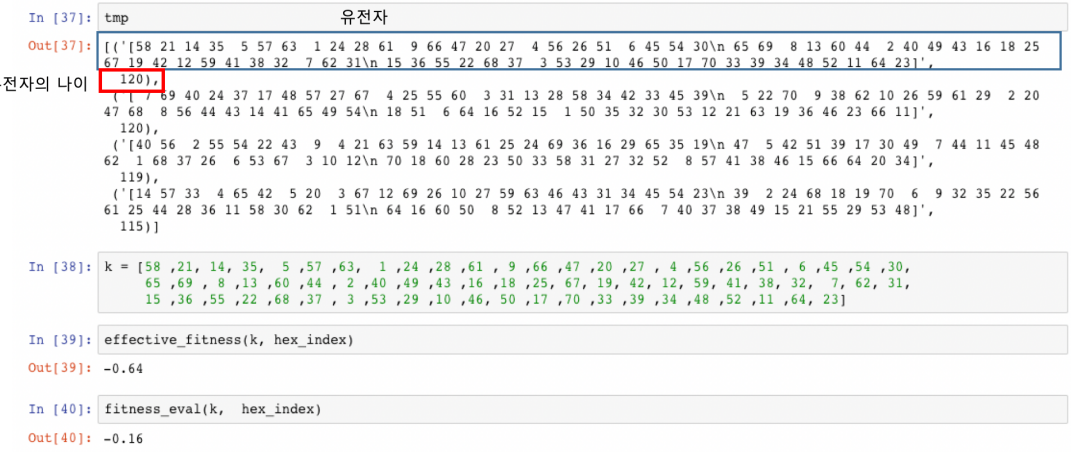


그림 6. 나이가 120인 유전자에 대한 실제 적합도는 -0.64로 나타났고, 일반 적합도는 -0.16으로 구해졌다. 따라서 세대를 지나면서 오래 살아남은 유전자에 대해 패널티를 부여하게 됨을 알 수 있다.

4) 교차(Crossover)

교차 연산은 2점 교차를 구현했다. 선택된 2개의 부모 유전자로부터 랜덤으로 2개의 위치를 정한 뒤, 연산을 수행한다. 이를 그림으로 표현하면 아래와 같다.

그림 7. 2점 교차의 예시

교차 연산 후, 생성된 자손 1, 2에 대해 각각 변이와 수선, 지역최적화 중 근접 교체를 사용했다. 마지막으로 각각에 대해 적합도 값을 구해 값이 큰 자손을 선택하는 토너먼트 방법을

적용해 하나의 자손을 모집단에 추가하도록 설계했다.

5) 변이(Mutation)

변이 연산은 [1]의 방법대로 구현했다. 유전자의 위치별로 1/3 확률로 1을 증가시키고, 1/3 확률로 1을 감소시킨다. 본 논문에서는 파이썬의 랜덤 함수를 사용해 0에서 99까지의 숫자를 생성하고, 생성된 숫자가 33보다 작을 경우 해당 위치에 1을 증가, 67보다 큰 경우 1을 감소하는 방법으로 구현했다.

6) 수선(Repairment)

교차 연산이나 변이 연산, 지역 최적화 중 이웃 탐색을 적용한 유전자는 값이 중복되는 등 적합한 해가 될 수 없게 된다. 따라서 이를 적합한 해의 형태로 만들어줘야 하는데 이때 수선 연산을 사용한다. 본 과제에서 수선 연산 구현은 다음과 같다. 1) 중복되어 나온 값들을 0으로 바꾼다. 2) 적합한 해가 되기 위해 나오지 않은 값들을 랜덤으로 유전자에서 값이 0인 부분에 덮어쓴다.

7) 교체(Replacement)

교체 연산은 모집단에 대해 실제 적합도(Effective fitness)를 구한 뒤, 가장 낮은 순대로 유전자를 교체했다. 본 논문에서는 모집단의 크기를 고정하고, 교차 연산으로 추가된 자손들의 개수만큼 유전자를 교체하는 방법을 사용했다.

8) 지역최적화(Local optimization) 1 : 근접 교체(Consecutive exchange)

첫 번째 지역 최적화 방법은 논문 [1]에서 사용한 근접 교체(Consecutive exchange)를 구현했다. 알고리즘은 아래와 같다.

```
do {
    changed ← False;
    for (v ← 1; v < n; v++) {
        l0 ← locationOf(v);
        l1 ← locationOf(v + 1);
        if (gain(l0, l1) > 0) {
            exchange the gene at l0 with the gene at l1;
            changed ← True;
        }
    }
} while (changed);
```

그림 8. 근접 교체 알고리즘

근접 교체의 아이디어는 유전자 값을 바꿨을 때, 적합도가 증가하면 계속해서 위치를 바꿔주고, 적합도 증가가 없을 경우 반복문을 빠져나오게 된다. 논문 [1]에서는 유전자 값의 차이가 1일 때를 사용했지만, 차이를 크게 설정할 수도 있다. 따라서 어느 방법이 좋은지 실험해봤고, 차원이 5로 설정하고 모집단 500일 때 적합도, 차이가 1일 때의 근접 교체, 차이가 2일 때의 근접 교체, 차이가 3일 때의 근접 교체 후의 유전자에 대한 적합도를 비교해 보았다. 아래 그래프는 각각의 방법에 대한 결과를 나타낸다.

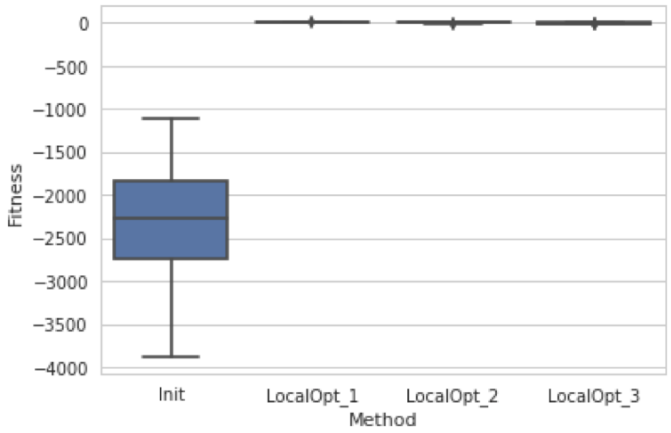


그림 9. 처음 적합도, 차이가 1, 2, 3으로 설정한 근접 교체 후의 적합도를 비교한 그래프.

확실히 지역 최적화를 했을 경우 적합도가 큰 폭으로 상승하는 걸 알 수 있다. 하지만 위의 그래프로 차이가 1, 2, 3으로 설정한 근접 교체의 비교가 어려워, 3가지만 다시 그래프를 그렸고 결과는 아래와 같다.

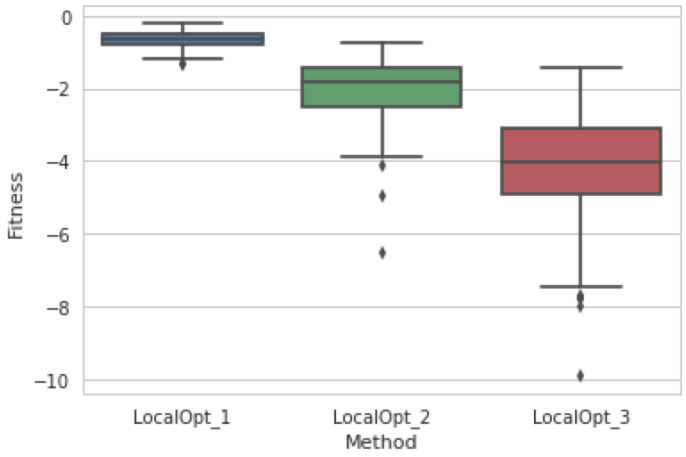


그림 10. 차이를 1, 2, 3으로 설정한 근접 교체 후의 유전자에 대한 적합도 비교.

근접 교체의 유전자 값의 차이를 1로 설정 하는게 가장 좋은 적합도를 보였다. 따라서 본 과제에서도 유전자 값의 차이를 1로 설정하고 실험을 진행했다.

두 번째 지역 최적화 방법은 타부 검색(Tabu search)을 적용했다. 핵심 아이디어는 해당 위치의 유전자를 바꾸었을 때, 비록 적합도가 증가하지 않았지만 탐색하지 않은 유전자일 경우에 위치를 바꿔준다는 점이다. 지역 최적화에 시간은 더 걸리지만, 더 많은 해를 탐색하고 그 결과로 유전자의 품질이 향상된다. 아래는 근접 교체에 타부 검색을 적용하는 알고리즘이다.

```
T = { original solution };
do {
    changed ← False;
    for each pair l0, l1 by the exchanging policy
        if (gain(l0, l1) > 0) {
            exchange the gene at l0 with the gene at l1;
            T ← { current solution };
            changed ← True;
        } else if (gain(l0, l1) = 0 ∧ current solution ∉ T) {
            exchange the gene at l0 with the gene at l1;
            T ← T ∪ { current solution };
            changed ← True;
        }
} while (changed);
```

그림 11. 근접 교체에 타부 검색을 적용한 알고리즘

본 과제에서 실험을 수행했을 때 차원이 4, 5일 때는 타부 검색이 없는 근접 교체만으로 해를 찾을 수 있었다. 차원을 6으로 했을 때 근접 교체만으로 해를 찾을 수 없어 타부 검색을 적용한 근접교체로 바꾸어 시도해 보았지만, 700세대 안에서 해를 찾을 수 없었다. 아래는 타부 검색이 없는 근접 교체의 적합도와 타부 검색을 적용한 근접 교체의 적합도를 비교한 그래프다.

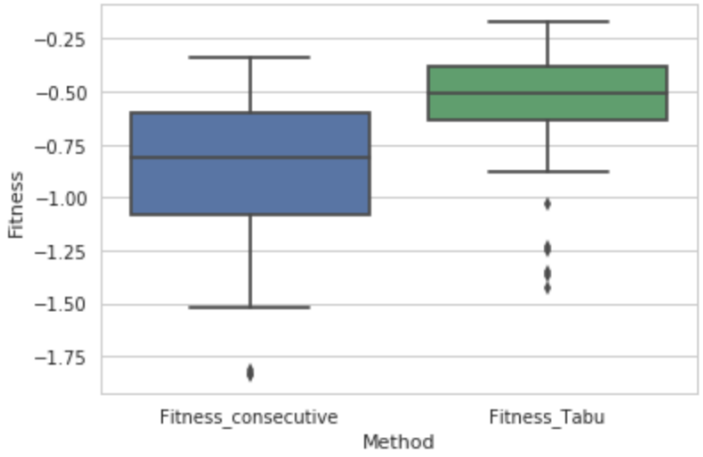


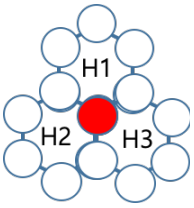
그림 12. 근접 교체를 적용한 해의 적합도와 근접 교체에 타부 검색을 적용한 해의 적합도를 비교한 그림. 타부를 적용한 근접 교체의 해의 품질이 더 높음을 알 수 있다.

9) 지역최적화(Local optimization) 2 : 이웃 탐색 (Nearby search)

지역 최적화 두 번째 방법은 논문 [1]에 있는 이웃 탐색이다. 이웃 탐색은 모든 노드에 대해 아래 식의 값을 구한 뒤, 이 값이 가장 작은 노드에 +1을, 가장 큰 노드에 -1을 해준다.

$$E(n) = \sum_{h \in H_n} [s(h) - \mu]$$

노드 n이 포함된 육각형마다, 각 육각형의 합에서 평균을 뺀 값을 더한 값이 위의 식이 된다. 이 값이 작다는 것은 노드에 해당되는 값이 작아 아래 그림과 같이 노드가 걸쳐 있는 육각형의 합이 평균보다 낮다는 의미다. 따라서 해당 노드 값을 증가시켜줌으로써 전체 분산을 낮추는데 목적이 있다. 반대로 위의 식의 값이 크다면, 평균보다 노드를 포함하는 육각형의 합이 크기 때문에 전체 분산이 큰 경우를 의미한다. 그러므로 해당 노드 값을 낮춰 육각형의 합을 적게 해 분산을 낮추는 효과를 기대할 수 있다.



논문 [1]에서는 값이 가장 작은 노드, 가장 큰 노드만 바꾸었지만, 위의 식으로 나온 값에 대해 분포를 만들어 백분위를 기준으로(위의 식의 값이 전체 노드에서 구한 값 중 20%보다 작으면 +1, 80%보다 크면 -1을 적용함) 값을 바꾸는 방법도 간단히 실험해 보았다. 차원 5인 지수귀문도에 근접 교체를 거친 모분포 500개를 기준으로, 2가지 방법을 적용한 결과는 아래 그래프와 같다.

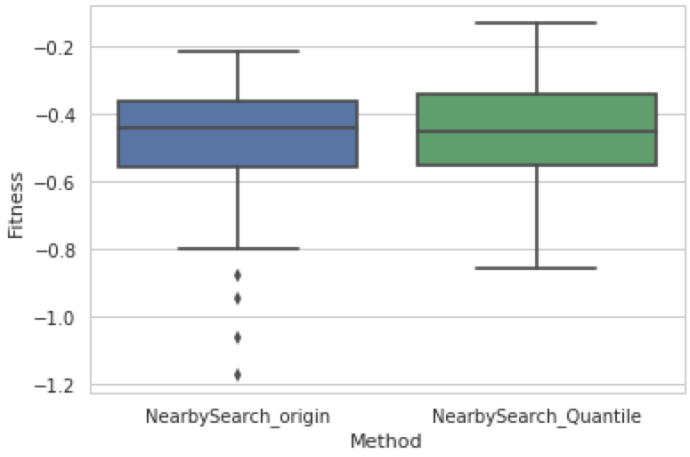


그림 15. 논문 [1]에서 제시한 방법과, 백분위를 기준으로 하는 방법을 적용한 후의 적합도를 비교한 그래프.

위의 그래프를 보면 논문 [1]에서 제시한 방법과 백분위를 기준으로 한 방법에 대한 적합도 비교 결과는 적합도 최댓값만 보면 두 번째 방법이 더 좋다. 하지만 해의 다양성 관점에서 보면 논문을 따른 방법이 더 나음을 알 수 있다. 그리고 백분위를 사용할 경우, 노드가 더 많이 바뀌게 되고 이는 기존 유전자가 지닌 스키마를 더 파괴하기 때문에 최종 해로 수렴이 잘 되지 않을 것이라 판단했다. 본 과제에서는 논문 [1]과 같이, 가장 값이 작은 노드, 가장 값이 큰 노드를 각각 1씩 증가, 1씩 감소하는 방법으로 실험을 진행했다.

10) 코사인 선택(Cosine selection)

코사인 선택 연산은 모집단이 지역 최적화에 갇혀 전역 최적해를 찾지 못할 때 사용하기 위해 고안해낸 방법이다. 코사인 유사도의 개념에 대한 그림은 아래와 같다.

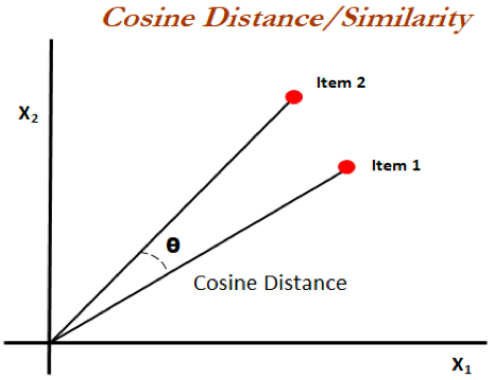


그림 16. 코사인 유사도를 나타낸 그림. 코사인 유사도가 클수록 두 벡터 사이의 Θ값이 작고, 코사인 유사도가 작을수록 Θ값이 크다.

코사인 유사도를 통해 좌표평면 상의 두 개의 벡터 사이의 각을 알 수 있다. 코사인 선택은 모집단에서 가장 좋은 적합도를 가진 유전자 K개를 선택한 후, 각 유전자별로 코사인 유사도가 가장 낮은, 좌표평면 상에서 반대에 있는 유전자를 찾아내 자손을 만드는데 목적이 있다.

코사인 유사도를 구하는 공식은 아래와 같다.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

코사인 선택을 통해 지역 최적화에 갇힌 모집단을 벗어나게 해, 더 많은 해를 탐험(Exploration)하는데 목적이 있다. 자세한 코사인 선택 연산 방법은 아래와 같다.

- 모집단에서 적합도가 가장 높은 K개의 유전자를 선택한다.
- 선택한 유전자 K에 대해 각각의 코사인 유사도가 가장 낮은 유전자를 선택해 교차, 변이, 수선, 토너먼트를 거쳐 최종 자손을 K개 생성한다.
- 원래의 모집단에서 K개의 유전자를 랜덤으로 선택해 코사인 선택으로 만들어진 유전자 K개로 대체한다.

11) 병렬프로그래밍

병렬 프로그래밍의 핵심은 지수귀문도를 풀면서, 병목이 되는 부분과 독립적인 연산을 찾아내는 것이다. 첫 번째로 지역 최적화 중 근접 교체(consecutive exchange)하는 부분이 병목이고 독립적으로 연산 가능한 부분이다. 유전자를 입력으로 받으면, 근접 교체를 적용한 유전자를 결과로 내도록 프로그래밍을 했다.

두 번째로 교차(crossover), 변이(mutation), 수선(repairment), 근접 교체(consecutive

exchange), 토너먼트(tournament) 연산을 하나로 묶을 수 있다. 언급된 연산은 2개의 부모만 필요하기 때문에, 2개의 부모의 인덱스를 저장한 2차원 배열을 만든 뒤, 원소별로 병렬 연산을 수행하도록 프로그래밍 했다.

세 번째로 지역 최적화 중 이웃 탐색 역시 독립적인 연산이다. 유전자 하나만 입력으로 받으면, 이웃 탐색을 수행한 유전자를 결과로 내보낼 수 있기 때문에, 병렬 프로그래밍을 적용하면 시간을 크게 단축시킬 수 있다.

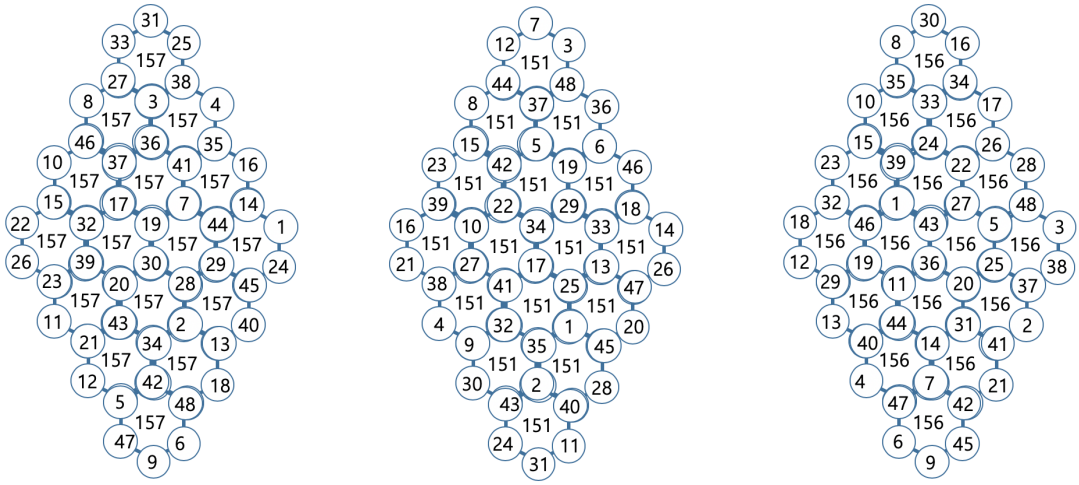
3. 실험 설계 및 결과

1) 차원 4 (노드;꼭짓점 48개, 육각형 16개)

모집단은 500개, 종료 조건은 해를 찾거나 최대 세대를 500으로 설정했다. 교차 연산으로 250개의 자손을 생성하고 모집단에 추가되며, 이에 따라 매 세대마다 랜덤으로 250개의 유전자가 교체된다. 지역 최적화는 근접 교체(consecutive exchange)와 이웃 탐색을 수행했으며 실제 적합도(effective fitness)는 적용하지 않고, 기본 적합도를 구하는 방법만 적용했다. 요약하면 아래 표와 같다.

	모집단 크기/ 최대 세대	Aging 적용	Cosine selection 적용	탐색한 해의 개수	해 탐지
실험1	500/500	X	X	-	O

실험 결과 시도할 때마다 해를 찾았고, 찾은 해는 아래와 같다.



9세대 72세대 64세대

그림 19

차원 4는 해 공간이 넓지 않고, 근접 교체가 강력한 지역 최적화임을 볼 수 있었다.

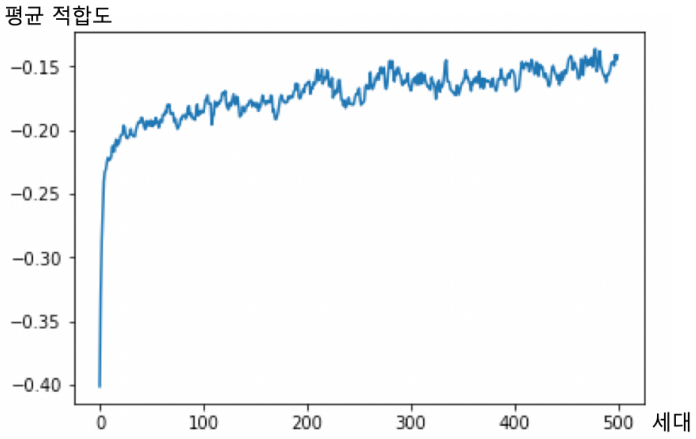
2) 차원 5 (노드 70개, 육각형 25개)

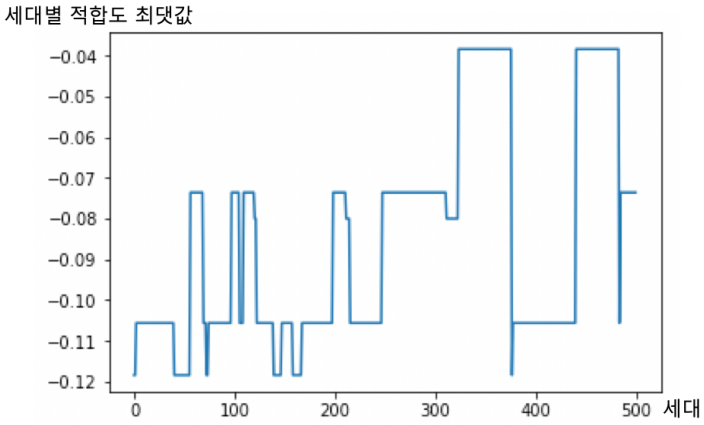
총 4가지의 실험을 수행했다. 실험1은 차원 4에서 적용한 그대로를 최대 세대 수만 늘렸다. 하지만 해를 찾을 수 없었다. 실험2부터는 실제 적합도(effective fitness)를 적용하기 위해 유전자의 빈도수를 세어 저장했고(aging), 실제 적합도를 기준으로 품질이 낮은 유전자를 교체하는 방법으로 바꾸었다. 모집단도 1000으로 늘렸고, 교차 연산으로 500개의 자손을 모집단에 추가했고, 500개의 유전자를 언급한 방법대로 교차했다. 하지만 실험2에서도 해를 탐지하지 못했다. 실험3에서는 코사인 선택(cosine selection)을 20세대마다 적용해 보았다. 적용한 결과 탐색한 해의 개수는 훨씬 늘어났지만, 역시 해를 탐지하지 못했다. 마지막 실험 4에서는 실험3과 조건은 같지만 코사인 선택을 매 100세대마다 적용하는 것으로 수정했다. 그 결과 해를 탐지할 수 있었고 위에 언급한 실험을 정리하면 아래 표와 같다.

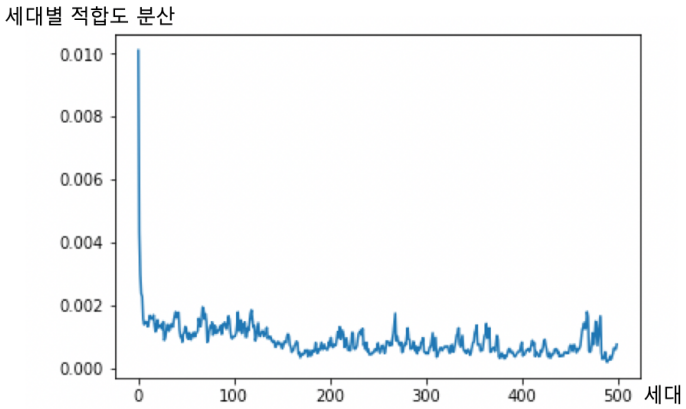
	모집단 크기/ 최대 세대	Aging 적용	Cosine selection 적용	탐색한 해의 개수	해 탐지
실험1	500/2000	X	X	-	X
실험2	1000/500	O	X	4,846	X
실험3	1000/500	O	O/20	21,828	X
실험4	1000/500	O	O/100	12,274	O

가장 처음에 했던 실험 1에 대해서는 해를 구할 수 없었다. 실험 1을 수행할 때 적합도를 미리 저장하지 않아 데이터가 없지만, 실험 2부터 글로벌 변수에 적합도를 저장했고, 이를 보면서 본 과제에서 설계한 유전알고리즘이 어떻게 동작하는지 짐작할 수 있었다.

실험 2의 결과는 아래 그래프와 같다.

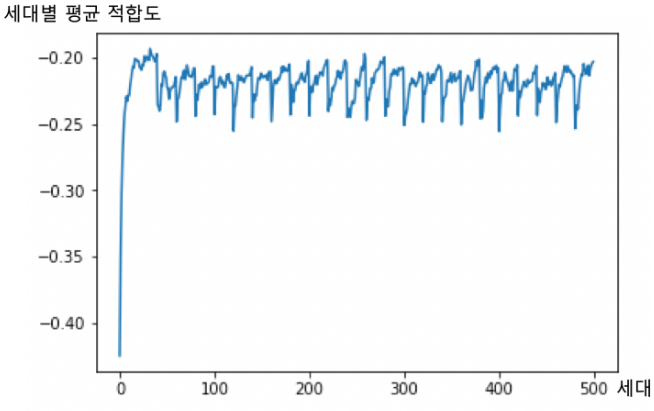


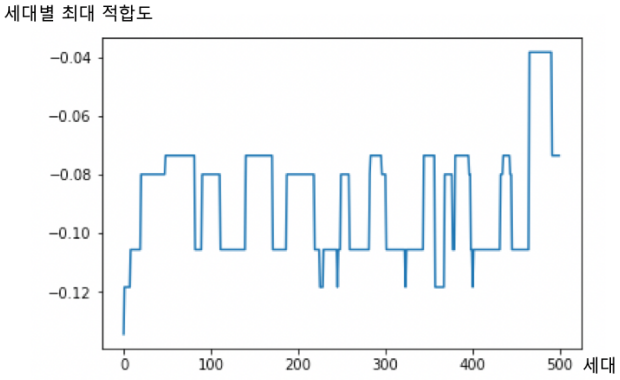


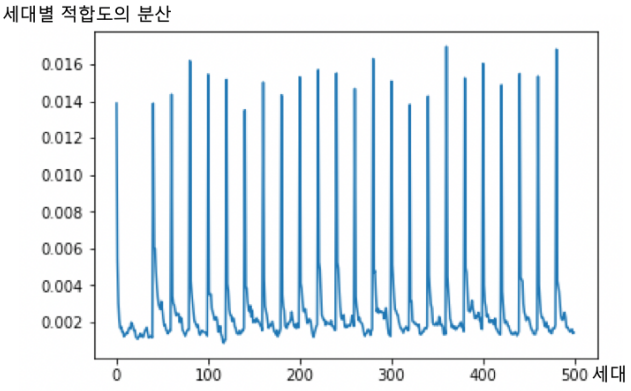


첫 번째 그래프를 보면, 세대가 증가할수록 유전자의 평균 적합도도 같이 상승함을 알 수 있다. 이는 구현한 알고리즘이 글로벌 최적 해로 나아가는 것 같지만, 탐색한 해의 개수를 실험 3, 실험 4와 비교했을 때 해의 범위가 좁아 지역 최적해일 가능성이 높다. 어느 정도 세대가 지나면, 모집단에 대한 충분한 해의 평가(Exploitation)가 되었다 생각하고, 모집단을 크게 바꿔주어 지역 최적해를 벗어나 더 많은 해를 탐색(Exploration)하는 방향으로 실험 3을 설계했다.

실험 3의 결과는 아래 그래프와 같다.

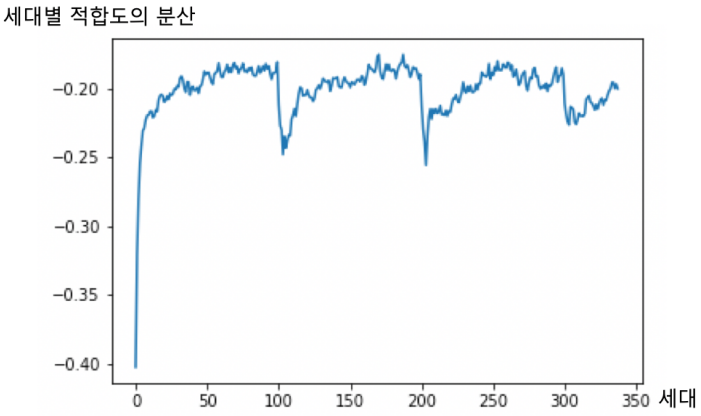




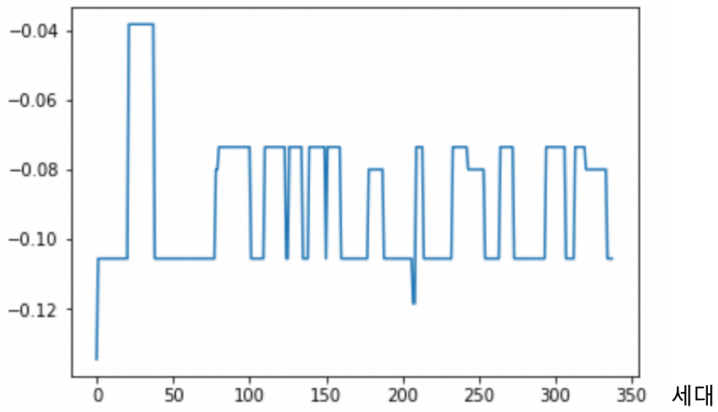


실험 3에서는 모집단에 큰 교란을 가하기 위해 코사인 선택을 20세대마다 적용했다. 확실히 20세대마다 분산이 크게 증가했음을 알 수 있고 이전 실험 2보다 훨씬 많은 해를 탐색했다. 하지만 실험 3의 경우 코사인 선택으로 새로운 해를 찾아 나가는데 성공했지만 모집단에 자주 큰 교란이 가해져서 모집단에서 충분한 해의 평가가 이루어지지 않았다. 이는 최적 해를 찾기 위한 모집단의 수렴을 방해했다고 생각되어 실험 4는 코사인 선택이 일어나는 세대 간격을 크기 조정했다.

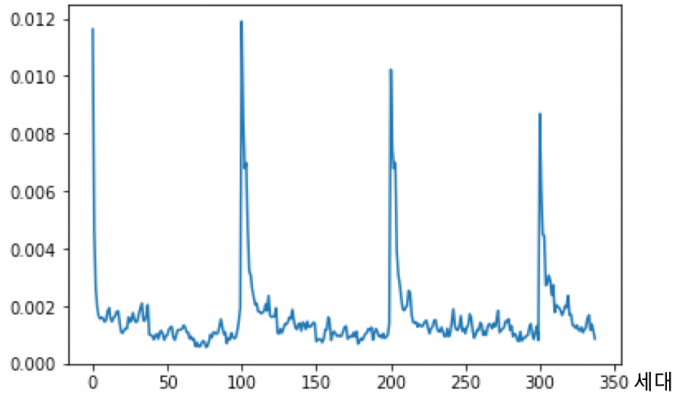
실험 4의 결과는 아래 그래프와 같다.



세대별 최대 적합도



세대별 적합도의 분산



실험 4에서는 코사인 선택을 100세대마다 수행하도록 설정했다. 실험 3에서보다 안정적으로 해를 탐색(Exploration)하고, 모집단의 평균 유전자의 적합도가 어느 정도 지역 최적해에 수렴했을 때 코사인 선택이 이루어진 것 같다. 388세대에 찾은 해는 아래와 같다.

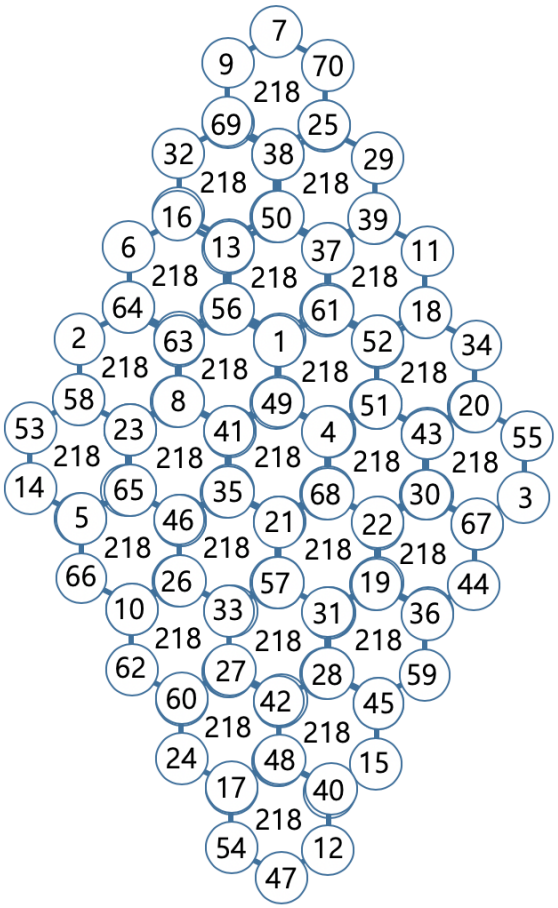


그림 30. 388세대에 찾은 차원 5의 지수귀문도

3) 차원 6일

근접 교차를 타부 검색이 들어간 근접교차로 적용해 해를 찾으려 시도했지만, 세대 수 800 이내에 해를 찾지 못했다.

4. 고찰

많은 실험을 통해 적절한 하이퍼 파라미터의 값을 찾아보는게 중요했다. 특히 지수귀문도의 경우 차원이 높아질수록 문제 해의 공간이 매우 복잡해지기 때문에, 이전 차원에서 사용한 파라미터를 그대로 사용하면 비효율적인 것 같다. 그리고 본 과제에서는 랜덤으로 부모를 선택해 교차 연산을 진행했지만, 실제 적합도(effective fitness)를 구현했기 때문에 이를 적극 활용하기 위해 선택하는 과정에서도 룰렛 휠 등을 사용하면 더 효율적인 알고리즘이 되었을 것 같다.

특히 본 과제를 통해 지역 최적화의 중요성을 알게 되었다. 처음에는 지역 최적화를 사용하면 해의 다양성이 떨어진다 생각해 차원 4의 지수귀문도를 풀 때, 랜덤으로 위치한 모집단으로 연산을 수행했었다. 하지만 적합도가 너무 적었고, 세대가 수렴하지 않고 아주 조금씩 적합도 향상이 있었다. 그러다가 논문 [1]에 있는 근접 교체를 적용해봤는데 한 세대를 지나는데 시간은 오래 걸렸지만 해의 적합도가 큰 폭으로 향상함을 보였다. 그리고 차원 4의 경우는 문제 공간이 낮다는 걸 알 수 있었는데, 근접 교체만 제대로 구현해 적용하면 적은 세대만으로도 최적해를 찾을 수 있었다. 하지만 차원이 높아질수록 최적해가 아닌, 지역 해에 갇혀 모집단을 크게 흔들어줄 필요성을 느꼈다.

차원 5의 해는 코사인 선택을 통해 답을 찾았는데, 6에 대한 해는 찾지 못해 아쉬웠다. 실험을 계속할 시간이 부족해 하이퍼 파라미터 튜닝이 제대로 안 된 채 유전알고리즘을 돌렸다고 생각한다. 차원 6의 해를 찾기 위해 코사인 선택을 150번마다 수행했었는데, 처음에는 150번이면 충분하다고 생각했지만 나중에는 150세대만으로는 최적해가 있을지도 모르는 모집단의 해공간에 대해 충분히 탐색이 이루어지지 않았다고 생각한다. 실험을 몇 번 해보지 못해 차원 6의 공간이 얼마나 복잡한지 잘 모르겠지만, 어느 시점에서 모집단을 크게 교란시켜줄지는 문제마다 다르고, 이를 잘 찾아내는 게 적은 세대로 해를 찾을 방법이라 생각한다.

참고 자료

[1] Hybrid Genetic Algorithm for the HExagonal Tortoise Problem