2015 GA Project 1: Traveling Salesman Problem

컴퓨터공학부 2008-11662 유재성

Abstraction

최적화 문제인 traveling salesman problem(TSP)를 순수 genetic algorithm(GA)를 통해 답을 구해본다. 일반적으로 local optimization 기법이 없는 순수 GA는 상대적으로 한계가 있다고 설명된다. 이 프로젝트에서는 순수 GA로 TSP 문제 해결을 위한 방법과 그 결과에 대해서 분석한다. 100개 이상의 도시에 대해 순수 GA만으로 알려진 global optimum을 찾는 것에는 한계가 있었지만 어느 정도 좋은 품질의 해를 찾을 수 있었다.

1. Introduction

1.1. 과제 개요

도시의 위치가 2차원으로 주어진 n개의 지점을 가장 짧게 순회하는 방법을 찾는다. 도시의 수는 최대 318개가 주어지며, 두 도시의 위치 사이의 거리는 두 도시의 위치(x1, y1), (x2, y2)의 좌표 차의 root mean square 값으로 계산한다. 각 도시의 위치는 모두 다르며, 한 도시에서 출발하여 모든 도시를 방문하고 다시 그 도시로 돌아오는 Hamiltonian cycle 중 가장 짧은 해답을 찾는다.

1.2. 구현 언어

프로그램은 C++을 사용했으며, 컴파일러 최적화 옵션은 -03까지만 사용한다.

1.3. GA의 구조 제한

순수 GA를 사용한다는 제약이 있기 때문에 local optimization 성격의 작업이 crossover 연산이나 mutation 연산에 사용될 수 없다. 단, selection 연산이나 replacement 연산에서는 해의 품질에 따라 선택할 해의 품질 비교나, 대체할 해의 품질 비교를 할 수 있다. 또, 초기 해를 만드는 과정에서도 local optimization 성격의 작업이 허용되지 않는다.

2. Genetic Algorithm Design

2.1. Overall Structure

과제 홈페이지의 뼈대코드와 구사하며, C로 구현된 부분을 C++로 변경했다. 기본적으로 매 generation마다 하나의 해를 replacement하는 steady state GA형태를 띈다. (기본적이라고 한 이유는 replacement 구현 중 하나는 경우에 따라 replacement를 수행하지 않기 때문이다.)

```
도시의 개수, 도시의 좌표, 테스트 시간을 입력 받는다.
Genetic algorithm에 사용될 selection, crossover, mutation, replacement 연산자를 지정한다.
Population을 초기화 한다.
While 테스트 시간 동안
Do
Selection(p1, p1Index)
Selection(p2, p2Index)
Crossover(p1, p2, c)
Mutation(c)
Replacement(p1, p2, c, p1Index, p2Index)
Done
```

2.2. Selection

해 집단(Population)에서 crossover 연산자에서 사용될 부모 해를 고른다. 이후 실험에서 다음과 같이 숫자로 selection 연산자를 구분한다. 0: Random, 1: Roulette wheel, 2: Tournament, 3: General tournament.

2.2.1. Random

해 집단에서 임의로 부모 해를 고른다.

2.2.2. Roulette wheel

해 집단의 모든 해들은 자기의 해로 순회했을 때의 거리를 값으로 계산하여 가지고 있다. Roulette wheel에서는 이 값을 해 집단에서 가장 긴 거리와 가장 작은 거리로 보정한 다음,

이 보정 값에 비례하여 부모 해를 선택한다.

2.2.3. Tournament

해 집단에서 임의로 두 개의 해를 선택한 후 특정확률(구현에서는 0.6)보다 높은 확률일 때는 거리가 더 짧은 해를, 낮은 확률일 때는 거리가 더 긴 해를 선택한다.

2.2.4. General tournament [1] [2]

초기 구현은 해 집단에서 임의로 특정 개수(구현에서는 5개)의 해를 선택한 후 특정확률(구현에서는 0.6, 0.6 * 0.4, 0.6 * 0.4^2, 0.6 * 0.4^3, 그 나머지)에 따라 거리가 더 짧은 해가 선택될 확률을 높이는 방식으로 구현했다.

하지만, 초기 구현의 수행 시간이 길다고 판단하여, 특정확률(구현에서는 0.6, 0.6 * 0.4)에 따라 거리가 가장 짧은 해와 그 다음으로 짧은 해를 선택하고 하였고 그 외에는 임의의 해중에 선택하도록 구현했다. 그 외의 확률은 가장 짧은 두 해의 확률에 비해 충분히 작다고 판단했기 때문이다.

2.2.5. Hybrid

Random, roulette wheel, tournament, general tournament를 각각 31:23:31:15의 비율로 적용하여 부모 해를 선택한다.

2.3. Crossover

이후 실험에서 다음과 같이 숫자로 crossover 연산자를 구분한다. 0: Simple, 1: Modified order, 2: Order, 3: Cycle, 4: PMX, 5: Edge recombination.

2.3.1. Simple

임의로 첫 번째 부모 해 또는 두 번째 부모 해를 crossover 연산자의 자식해로 만든다.

2.3.2. Modified order

One-point crossover에서 영감을 받아 각 도시를 한번씩 방문하는 TSP 해의 조건에 맞게 알고리즘을 수정하였다. 수정하고 보니, order crossover와 동일한 알고리즘인데 첫 번째 자름 선을 0으로 하는 것과 같은 구현이었다.

2.3.3. Order [3]

2.3.4. Cycle [3]

2.3.5. PMX (Partially Matched Crossover) [3]

2.3.6. Edge recombination [3]

2.3.7. Hybrid

Simple, modified order, order, cycle, PMX, edge recombination을 각각 24:13:35:11:17의 비율로 적용하여 새로운 해를 만들어낸다.

2.4. Mutation

이후 실험에서 다음과 같이 숫자로 mutation 연산자를 구분한다. 0: Random, 1: Typical, 2: Range shuffle, 3: Inversion, 4: Double-bridge kick move, 5: Or change.

2.4.1. Random

해의 두 도시를 임의로 선택하여 서로 바꾼다.

2.4.2. Typical

해의 각 도시에 대해 특정 확률(구현에서는 0.125)로 임의의 도시와 바꾸도록 한다.

2.4.3. Range shuffle

해의 두 지점을 잡아 두 지점 사이의 도시 순회 순서를 임의로 섞는다.

2.4.4. Inversion (2-change)

Inversion mutation 연산자는 임의로 연속된 구간을 잡아 그 구간에 있는 도시의 순서를 뒤집는다. 그런데 이것은 2개의 간선을 임의로 선택하여 제거한 후, 그 간선을 사용하지 않고 다른 간선 두 개를 사용하여 유효한 경로를 만드는 2-change mutation 연산자와 같은 구현이다.

2.4.5. Double-bridge kick move

네 간선을 임의로 제거한 후 그림과 같은 새로운 네 개의 간선으로 경로를 수선하여 새로운 경로를 만들어내는 방법이다. 이 구현을 생각해보면 해에 임의의 세 지점을 정하여 1, 2, 3, 4로 나눈 다음, 4, 3, 2, 1로 재조합 하는 것과 같은 구현이기 때문에 이와 같이 구현하였다.

2.4.6. Or change

한 도시를 임의로 다른 두 도시 사이에 넣는 방법이다.

2.4.7. Hybrid

Random, range shuffle, inversion, double-

bridge kick move, or change를 각각 8:9:56:1:25의 비율로 적용하여 해를 변형한다.

2.5. Replacement

이후 실험에서 다음과 같이 숫자로 replacement 연산자를 구분한다. 0: Random, 1: Elitism, 2: Preselection, 3: Hybrid 1, 4: Hybrid 2.

2.5.1. Random

해 집단의 임의의 해를 대체한다.

2.5.2. Elitism

해 집단의 가장 나쁜 해(TSP 문제에서는 가장 길이가 긴 해)를 대체한다.

2.5.3. Preselection

부모해중 더 나쁜 해(TSP 문제에서는 더 길이가 긴 해)를 대체한다.

2.5.4. Hybrid 1

자식 해가 부모 해보다 성능이 좋으면(TSP 문제에서는 길이가 더 짧으면) preselection를 수행하고, 나쁘면 elitism를 수행한다.

2.5.5. Hybrid 2

자식 해가 부모 해보다 성능이 좋으면(TSP 문제에서는 길이가 더 짧으면) preselection를 수행한다.

2.5.6. Hybrid 3

Preselection, hybrid 1, hybrid 2를 각각 9:47:44의 비율로 적용하여 해를 대체한다.

2.6. 문제의 표현

과제 뼈대코드에서 사용한 것처럼 order based encoding을 사용하여 문제를 표현하였다. 이 문제에서는 order based encoding을 사용하면 crossover 연산자 등의 구현이 간단해지기 때문에 다른 encoding으로 변경하지 않고 그대로 사용했다. 다만, order based encoding은 같은 해를 표현하는 다양한 유전자 표현이 등장할 수 있는데 이 문제를 해결하기 위해 normalization을 수행했다. 다만 이 normalization의 효과에 대해서는 이후의 실험에서 다시 다루었다.

3. Experiments

3.1. 실행환경

3.1.1. Machine 1

Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
MemTotal: 2051280 kB

3.1.2. Machine 2

Intel(R) Core(TM) i7-4550U CPU @ 1.50GHz
MemTotal: 7860648 kB

3.2. 전수 조사

모든 selection, crossover, mutation, replacement 조합에 대해 실험을 진행하였다. Shell script를 통해 다수의 GA를 동시에 학습시켰다.

3.2.1. 도시 101개, 실험 횟수: 2~3회.

Selection	Crossover	Mutation	Replacement	Average	표준편차	Min
2	0	3	2	779.8015	1.710491	778.592
0	0	3	2	793.315	7.394923	788.086
2	1	3	3	802.1235	14.90369	791.585
0	1	3	4	804.3255	5.240368	800.62
0	4	3	4	804.568	2.804386	802.585

3.2.2. 도시 51개, 실험 횟수: 4회

Selection	Crossover	Mutation	Replacement	Average	표준편차	Min
2	0	3	2	601.2498	3.097876	598.271
0	0	3	2	603.3565	11.41423	589.348
2	2	3	3	605.0038	9.219514	598.271

0	2	3	3	605.0155	7.55924	596.133
3	1	3	3	605.96	15.56448	592.396

3.2.3. 도시 21개, 실험 횟수: 10회

(Selection, Crossover, Mutation, Replacement)가 각각 (0,0,3,2), (0,1,3,2)……(3,2,5,4)외 다수의 조합 모두 Average: 399.711, 표준편차 0, Min: 399.711

3.2.4. 도시 11개, 실험 횟수: 15회

(Selection, Crossover, Mutation, Replacement)가 각각 (0,0,0,2), (0,0,0,3)……(3,4,5,4)외 다수의 조합 모두 Average: 345.875, 표준편차: 0, Min: 345.875

3.3. 전수조사에서 가장 뛰어난 성능을 보인 연산자 조합 성능 측정

	Min	표준편차	Average	Global optimum	# of min = global
도시 11개	345.875	0	345.875	345.875	30
도시 21개	399.711	0	399.711	399.711	30
도시 51개	589.348	3.899017	601.1904	589.348	1
도시 101개	774.58	5.335449	781.9198	772.119	0
도시 318개	504.155	7.022401	519.5951		

문제 공간이 커질수록(도시의 개수가 증가할수록) global optimum 해를 찾는 빈도가 급격하게 준다는 것을 확인할 수 있었다. 그럼에도 불구하고, 꽤 좋은 성능의 해를 찾을 수 있다는 것을 확인하였다.

3.4. 전수조사 결과를 통해 hybrid 연산자 구현 및 성능 측정

Selection, crossover, mutation, replacement 연산자 각각에 대해서 좋은 성능을 보였던 함수들을 특정한 비율로 결합하여 hybrid 연산자를 만든다면 좋은 성능을 보일 것이라는 추측을 했다. 따라서 도시 11개일 때 Average가 Min인 연산자의 조합에서 각각의 연산자의 점유 비율을 조사하였다. 도시 21개일 때와 51개일 때는 Min가 가장 작은 상위 73개의 조합에서 각각의 연산자의 점유 비율을 조사하였다. 마지막으로 도시 101개일 때는 Min가 가장 작은 상위 75개의 조합에서 각각의 연산자의 점유 비율을 조사하였다.

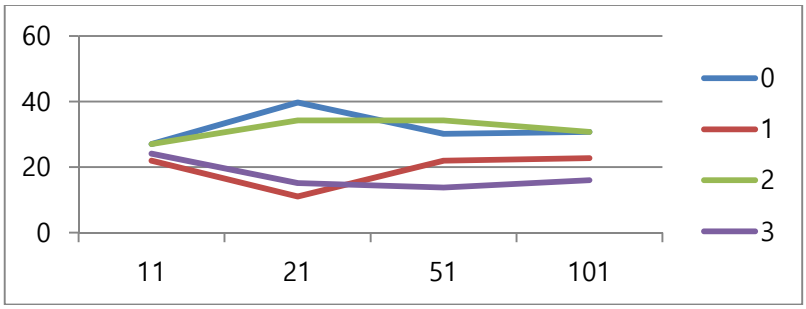


그림 1 도시의 개수에 따라 각 selection 연산자가 점유하는 비율(%)

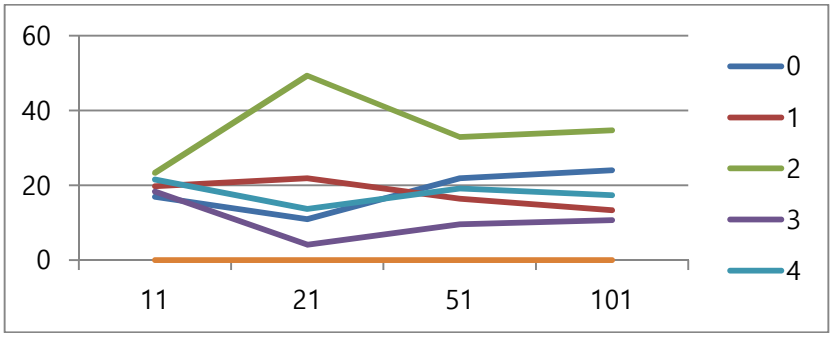


그림 2 도시의 개수에 따라 각 crossover 연산자가 점유하는 비율(%)

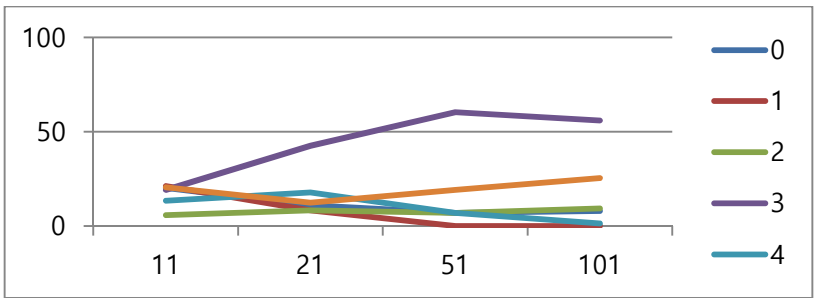


그림 3 도시의 개수에 따라 각 mutation 연산자가 점유하는 비율(%)

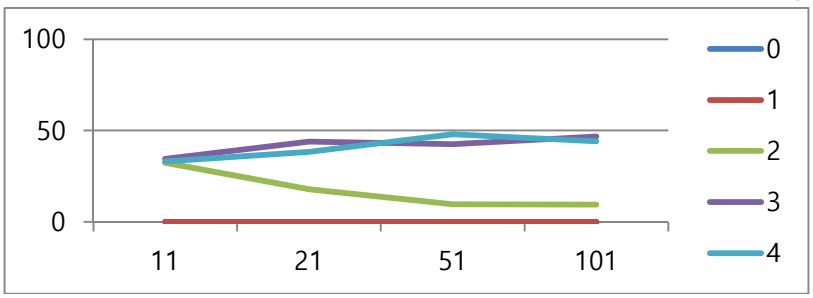


그림 4 도시의 개수에 따라 각 replacement 연산자가 점유하는 비율(%)

Selection, crossover 연산자들은 상대적으로 점유하는 비율의 차이가 크지 않았고, 이해 비해 mutation, replacement 연산자들은 점유하는 비율의 차이가 극명했다.

3.4.1. Hybrid 연산자 성능 측정. (30회 측정)

	Min	표준편차	Average	Global optimum	# of min = global
도시 11개	345.875	0	345.875	345.875	30
도시 21개	399.711	0	399.711	399.711	30
도시 51개	589.348	8.222715	602.1031	589.348	2
도시 101개	784.279	10.58082	801.4974	772.119	0
도시 318개	469.824	40.67756	551.0095		

전수조사에서 구한 최고의 조합과 유사하게 문제 공간이 커질수록(도시의 개수가 많아질수록) 알려진 global min distance를 얻는 횟수는 급격하게 줄어들었지만, min distance를 참고하면, global min distance와 유사한 품질 좋은 local optimum 해를 구한다는 것을 확인할 수 있었다. 하지만 전수조사에서 구한 최고의 조합과 비교했을 때 아무 미세하지만 조금의 성능 개선이 이뤄짐을 확인할 수 있었다. Crossover, mutation 연산자에서 조금의 비율이지만 다양성이 가미되었기 때문에 이런 결과가 얻어졌다고 판단할 수 있다.

3.5. 다른 연산자를 hybrid 연산자로 고정한 후 selection 연산자들에 대해 각각 성능 측정(30회)

Selection	0	1	2	3
Min	778.918	818.272	779.525	789.541
Average	802.112	874.546	797.746	805.278
표준편차	12.3628	34.6427	11.6141	10.185

실험 결과에 따르면 selection 연산자의 종류에 따른 성능의 차이는 거의 없다고 봐도 무방할 것 같다.

3.6. 다른 연산자를 hybrid 연산자로 고정한 후 cross 연산자들에 대해 각각 성능 측정(30회)

Crossover	0	1	2	3	4	5
Min	776.791	781.256	1112.91	958.035	781.326	3600.58
Average	794.115	801.417	1192.76	1130.2	798.428	3717.68
표준편차	9.46481	11.08	34.24	71.026	11.0914	42.9699

Crossover 연산자 중 edge recombination crossover 연산자의 성능이 다른 연산자에 비해 매우 나쁨을 확인할 수 있었다. 심지어, 부모해중 임의의 해를 선택하여 자식해로 쓰는 random crossover 연산자의 성능이 상당히 좋게 나온다는 것은, crossover의 구현을 복잡하게 하여 다양성을 높이는 것이 생각보다 큰 의미를 가지지 않을 수도 있다는 것을 의미한다.

3.7. 다른 연산자를 hybrid 연산자로 고정한 후 mutation 연산자들에 대해 각각 성능 측정(30회)

Mutation	0	1	2	3	4	5
Min	897.329	3015.59	875.009	781.708	1122.83	829.434
Average	954.1923	3096.48	975.09	798.762	1250.65	905.163
표준편차	37.33598	42.7171	36.5161	11.2072	45.1553	30.925

Mutation 연산자 중 typical mutation 연산자의 성능이 다른 연산자에 비해 매우 나쁨을 확인할 수 있었다.

3.8. 다른 연산자를 hybrid 연산자로 고정한 후 replacement 연산자들에 대해 각각 성능 측정 (30회)

Replacement	0	1	2	3	4
Min	3504.42	3669.07	1028.2	784.01	781.513
Average	3728.37	3794.53	1084.42	807.411	804.613
표준편차	66.1837	49.8241	26.1945	13.6033	15.2341

Replacement는 연산자에 따라 성능차이가 가장 컸다. 특히 random과 elitism의 성능이 매우 좋지 않았다.

3.9. Normalization을 수행하지 않고 성능 측정. (30회, 전수조사 때의 연산자 조합으로 수행)

	Min	표준편차	Average	Global optimum	# of min = global
도시 11개	345.875	0	345.875	345.875	30
도시 21개	399.711	0	399.711	399.711	30
도시 51개	589.348	6.00185	599.581	589.348	3
도시 101개	773.042	8.26943	783.448	772.119	0

도시 318개	511.206	5.72773	520.379		

Normalization을 하지 않았을 때 해의 평균적인 품질이 높아졌다(distance가 줄어들었다.) 이는 normalization의 효과가 그리 크지 않으며, 오히려 normalization 때문에 generation 횟수가 줄어들어 생긴 결과라고 추측된다.

3.10. Population의 크기에 따른 해의 품질 측정.

3.10.1. 30회, 도시 51개, 전수조사 때의 연산자 조합으로 수행

Population size	25	50	100	200
Min	592.626	589.348	589.348	589.348
Average	596.3259	592.7712	601.1904	597.4645
표준편차	3.200636	4.527767	3.899017	6.192359

3.10.2. 30회, 도시 101개, 전수조사 때의 연산자 조합으로 수행

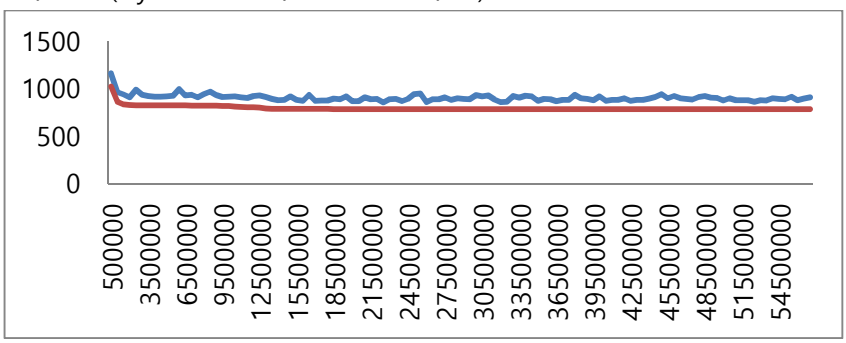
Population size	25	50	100	200
Min	911.204	788.235	774.58	775.376
Average	945.0449	805.767	781.9198	790.556
표준편차	13.75504	6.431547	5.335449	8.439138

3.10.3. 30회, 도시 318개, 전수조사 때의 연산자 조합으로 수행

Population size	25	50	100	200
Min	973.918	676.181	504.155	442.166
Average	999.1535	692.8174	519.5951	454.9565
표준편차	9.769824	9.03707	7.022401	6.944931

문제의 크기가 작을 때는(도시의 개수가 51~101개일 때) Population size가 25일 때는 다른 경우에 비해 나쁜 결과를 보였고, 그 외의 경우는 비슷한 성능을 보였다. 이는 Population size가 충분히 크다면 결과에 큰 영향을 미치지 않는다는 것을 알 수 있었다. 오히려 도시 101개일 때 population size가 200일 때 100일 때보다 성능이 좋지 않았는데, 이는 population size가 커지면 population들이 local optimum으로 수렴하는 시간이 부족해지기 때문이라는 추측을 해볼 수 있다. 하지만 문제의 크기가 커지면(도시의 개수가 318개일 때)는 population size가 200일 때 확실한 효과가 있었다.

3.11. 도시가 101개인 데이터에 대해 세대가 진행됨에 따라 population 내의 해들의 평균 품질과 최고 품질의 모양(Hybrid 연산자 조합으로 수행)



4. Discussion

4.1. 예상한 대로 동작한 점

문제 공간이 작으면(도시의 개수가 작으면) 어떤 조합의 연산자를 사용해도 global optimum

을 잘 찾는다.
문제 공간이 커져도(도시의 개수가 커져도) 좋은 성능의 연산자들의 조합이면 global optimum 까지는 아니어도 적당한 성능의 local optimum을 잘 찾는다.

4.2. 실험을 통해 알게 된 점

연산자들마다 성능 차이가 거의 없거나 상당히 크다. 특히, selection 연산자들의 경우 연산자들 사이의 성능 차이가 거의 없음을 확인할 수 있었고, crossover 5(edge recombination), mutation 1(typical), replacement 0(random), 1(elitism)의 연산자들은 다른 연산자들에 비해 성능이 확연하게 떨어지는 것을 확인할 수 있었다.
Hybrid 연산자들을 만들기 위해 전수조사에서 상위 해들에서 각 연산자들의 빈도를 조사했는데, 이 빈도가 높을수록 연산자 각각의 성능 또한 좋은 편임을 확인할 수 있었다. 반대로 상위 해들에서 거의 발견되지 않는 연산자들은 각각의 성능을 측정했을 때도 형편없는 것을 확인할 수 있었다. E.g. crossover 5: edge recombination, mutation 1: typical, replacement 0: random, 1: elitism. 하지만 이 둘 사이에는 완벽히 비례하지는 않는데, crossover 2(order) 의 경우 상위 해들에서 가장 다수 발견된 crossover 연산자인데, crossover 연산자들끼리의 성능 측정에서는 생각보다 좋지 않았기 때문이다.
Hybrid 연산자들의 성능이 더 좋았다는 점 또한 알게 되었다. 이는 앞에서 이야기했듯이, 확률에 따라 다른 연산자를 사용함으로써 다양한 문제 공간에 대응할 수 있기 때문이 아닐까 추측한다.

4.3. 기대했던 것과 다른 결과를 보인 부분

Normalization이 성능에 거의 영향을 미치지 않았다. 물론 실험에서는 특정 연산자들에 대해서만 normalization 성능을 측정했기 때문에 normalization이 큰 영향을 미치는 연산자를 썼다면 결과가 달라졌을지도 모른다는 추측을 해본다.
Population size와 성능이 비례하지 않는 다는 것도 놀라웠다. 실제로는 문제의 복잡도에 따라 적당한 population 크기를 사용해야 한다는 실험결과를 얻었기 때문에 데이터에 대한 이해와 실험의 중요성에 대해 알 수 있는 부분이었다.
Selection 연산자가 거의 무의미하고, crossover 0(부모 해 random)의 성능이 좋다는 것 또한 시사하는 바가 크다. Crossover, mutation 연산자가 해 집단에 다양성을 부여하기 때문에 아주 중요하다고 생각했는데 순수 GA에서는 mutation 연산자와 replacement 연산자를 어떻게 디자인하는지가 더 중요하다는 것을 알 수 있었다. 아마 crossover 연산자에 의해 생성된 자식 해가 다양성을 증가시키는 mutation 때문에 crossover 연산자의 영향이 줄어들었다고 추측한다.

4.4. 아쉬운 점

측정할 때 두 개의 환경에서 측정한 점이 아쉽다. 또, 동시에 많은 실험을 하느라 환경의 성능이 GA 성능에 영향을 꽤 끼쳤을 것이라고 판단된다.
구현을 하였는데 시간적인 측면에서 너무 많은 단점이 있는 구현을 없애거나 원래의 구현과 다르게 했는데, rank selection은 구현을 했는데 제거하였고, typical mutation은 vector create을 하지 않는 방식으로 변경하였다. General tournament selection 또한 성능 때문에 1, 2위와 그 나머지로 구현을 변경하였다.

5. Reference

[1] http://en.wikipedia.org/wiki/Tournament_selection
[2] http://www.complex-systems.com/pdf/09-3-2.pdf
[3] http://www.aistudy.com/biology/genetic/operator_moon.htm#_bookmark_1a5dc40