**Assignment3**

20221153 컴퓨터공학과 김현수

**<Assignment02와 비교하여 달라진 점과 그 이유>**

Assignment02와 달라진 점은 해의 표현 방식, 돌연변이 연산 및 교차 연산과 선택 연산, 엘리티즘 비율로 성능이 더 좋은 알고리즘을 만들고자 하였다. 개선 전의 해의 표현은 그래프의 정점의 부분집합으로 표현했으나 현재는 그래프의 각 정점에 대한 이진 값을 가지는 배열로 표현 방식을 변경하였다. 변경한 해의 표현 방식은 각 정점이 두 개의 파티션 중 어느 파티션에 속하는지를 나타내는 일에 있어 더 직관적이고 효율적인 방식이라 생각 되었기에 현재의 방식으로 변경하였다. 변경된 해의 표현 방식에 맞추어 돌연변이 연산도 수정하였다. 이전의 알고리즘에서는 랜덤하게 정점을 추가하거나 제거하는 방식으로 돌연변이를 적용하였고 이는 현재의 해의 표현 방식에 알맞지 않아 돌연변이 연산을 개체의 유전자 하나를 랜덤하게 토글하여 이진 표현 방식에 더 적합하도록 수정하였다. 선택 연산과 교차 연산 또한 수정하였는데 선택연산은 이전의 룰렛휠 방식을 사용하였으나 현재는 토너먼트 방식을 선택했다. 토너먼트 선택은 토너먼트 사이즈를 조정할 수 있기에 다양한 비교가 가능하고 다양성과 수렴속도 사이의 균형을 맞출 수 있다는 이유로 토너먼트 방식으로 변경하였다. 또한 룰렛휠은 지역 최적해에 갇히는 경향이 컸는데 토너먼트 방식은 무작위 방식으로 개체를 선택하여 지역 최적해에 갇히는 경향이 적었다. 뿐만 아니라 교차 연산 또한 수정하였는데 개선 전의 알고리즘에서는 두 부모에서 임의의 점을 선택하여 그 점을 기준으로 두 부모의 유전자를 교환하는 2 point crossover 방식을 사용하였으나 현재는 uniform crossover 방식을 사용하도록 수정하였다. 이는 각 위치에서 부모의 유전자를 무작위로 선택하므로 전체 chromosome에 걸쳐 더 균일한 교환을 가능하게 한다. 이는 2 point crossover 방식보다 유전자의 다양성을 더 효과적으로 유지하며 해의 탐색 공간에서 더 넓은 영역을 커버할 수 있도록 해준다. 마지막으로 이전의 알고리즘보다 엘리티즘 비율을 증가시켰다. 이는 각 세대에서 우수한 개체들을 더 많이 다음 세대로 직접 전달하여 알고리즘의 수렴을 빠르고 효과적으로 만든다.

**<가장 좋은 GA를 선택하게 된 과정과 사용한 연산자에 대한 설명>**

선택 연산자를 룰렛휠에서 토너먼트로 변경도 해보고 교차 연산도 k-point crossover 방식으로도 해보았는데 해의 품질이 음수에서 벗어나질 못했다. 세대를 거듭할수록 해의 품질 또한 아주 미세하게 진화하였다. 계속 선택 연산과 교차 연산을 다양한 조합으로 변경해보며 수정을 거듭하다 토너먼트 선택 방식과 uniform crossover방식으로 수정하고 엘리티즘의 비율을 50%로 높여 보았더니 여전히 음수에 머물러 있긴 하나 세대를 거듭할수록 그 값이 점점 더 좋은 품질로, 아래와 같이 눈에 띄게 진화하는 양상을 보였다.

<generation> <best fitness>

|  |  |
| --- | --- |
| 100 | -94512 |
| 200 | -56880 |
| 300 | -40028 |
| 400 | -29296 |
| 500 | -19744 |
| 600 | -13256 |
| 700 | -11252 |
| 800 | -4368 |

(input = chimera\_946.txt)

이후 Merge Sort 정렬 방식으로 토너먼트 연산자에서 토너먼트 크기를 다양하게 선택하며 여러 실험을 해본 결과 토너먼트 크기가 커지면 시간이 오래 걸리며 200세대만에 음수에서 양수의 해를 도출하지만 양수의 해에서 더딘 증가폭을 보이며 최종(1000세대) 품질은 토너먼트 사이즈가 적당할 때보다 Best Fitness가 낮았다. (지역 최적해에 갇힘.) 토너먼트 크기를 작게 설정할 경우 음수에서 양수의 해를 찾기까지 여러 세대를 반복해야하나 음수에서 시작하여 세대를 거듭할수록 더 좋은 품질을 찾아 진화하며 최종적으로 양수의 품질 값을 도출했다. 그러나 음수에서 양수의 해를 찾기까지 여러 세대를 반복해야 한다는 점에서 최종(1000세대)적인 품질은 토너먼트 사이즈가 적당할 때보다 좋지 않았다. 따라서 토너먼트 사이즈를 너무 크지도 않고 너무 작지도 않게끔 조정하여 최적의 사이즈를 찾는 것에 집중했다. 다양하게 토너먼트 사이즈를 비교해본 결과 사이즈 5 정도가 가장 좋은 결과를 도출했다.

|  |  |
| --- | --- |
| 100 | -29685.6 |
| 200 | 11285 |
| 300 | 23480.2 |
| 400 | 30470.1 |
| 500 | 34069.9 |
| 600 | 37562.4 |
| 700 | 39371.1 |
| 800 | 40240.4 |
| 900 | 41928 |
| 1000 | 42742.3 |

<Generation> <Average Fitness> <Generation> <Average Fitness> <Generation> <Average Fitness>

|  |  |
| --- | --- |
| 100 | -71940 |
| 200 | -19012.4 |
| 300 | 4000.86 |
| 400 | 16127.1 |
| 500 | 23647.4 |
| 600 | 28654.4 |
| 700 | 33239.4 |
| 800 | 36101.5 |
| 900 | 37700.7 |
| 1000 | 39966.2 |

|  |  |
| --- | --- |
| 100 | -7908.32 |
| 200 | 27791.4 |
| 300 | 35882.6 |
| 400 | 39477.3 |
| 500 | 40465.2 |
| 600 | 41349.5 |
| 700 | 42550.6 |
| 800 | 42581.3 |
| 900 | 42993.2 |
| 1000 | 43327.9 |

Tournamentsize=5 Tournamentsize=10 Tournamentsize=2

(input = chimera\_946.txt) (input = chimera\_946.txt) (input = chimera\_946.txt)

위의 과정들을 토대로 나의 GA는 selection연산은 tournament, crossover연산은 uniform crossover, random mutation, generational GA replace로 가중치의 정렬은 intelligent quick sort를 사용한다.

현재 GA에서 사용한 tournament Selection연산자는 무작위로 선택된 여러 개체 중에서 가장 적합도가 높은 개체를 선택하는 연산이다. 동일한 프로세스가 두 번째 부모를 선택하기 위헤 반복된다. Crossover연산에는 uniform crossover를 사용하였는데 uniform crossover연산은 두 부모 개체로부터 자식을 생성하는데, 각 유전자의 위치에서 무작위로 부모의 유전자를 선택한다. 이는 개체들의 다양성을 유지하는 데 효과적이다. Mutation연산은 이전 assignment02와 동일한 연산을 사용하는데, 이는 개체의 각 유전자에 대해 설정된 확률 (0.01)로 돌연변이를 발생시킨다. 돌연변이는 유전자의 값을 반전시키는 방식으로 이뤄진다. 가중치의 정렬에서는 다양한 정렬알고리즘 중 intelligent quick sort 방식을 선택했다. 해당 정렬 알고리즘은 기본 quick sort알고리즘의 변형으로 중앙값을 피벗으로 선택하는 quick sort알고리즘이다. 이는 개체들을 빠르고 효율적으로 정렬하고 기본 quick sort보다 더 안정적인 알고리즘이다. 마지막으로 엘리티즘 방식을 사용하여 가장 적합한 개체를 일부 보존함으로써 세대 간의 우수한 유전 정보를 유지한다.

**텍스트, 스크린샷, 폰트, 직사각형이(가) 표시된 사진

자동 생성된 설명<5가지 정렬 알고리즘을 현재 GA를 이용해 비교 분석한 과정과 결과>**

Input 데이터는 chimera\_946.txt이고 정렬 알고리즘은 Merge sort를 기준으로 다양한 GA를 수정하며 최적의 GA를 찾아낸 후 해당 GA를 토대로 5가지의 정렬 알고리즘을 비교했다. Chimera\_946 데이터로 동일한 조건을 두고 5가지의 정렬 알고리즘을 비교하였다. 5개의 정렬 알고리즘 중 Counting Sort를 제외한 나머지 4개의 Sorting알고리즘은 좋은 품질을 도출했으나 Counting Sort는 세대를 거듭할수록 좋은 품질로 향상되지 않는 것은 물론이며 음수 가중치의 해 사이에서 벗어나지 못하고 있는 것을 볼 수 있다. Couting Sort를 제외한 나머지 4개의 Sorting 알고리즘은 대체적으로 비슷한 품질을 도출하고 있으나, 그 중 intelligent quick sort 알고리즘이 best fitness, average fitness 모두 가장 좋은 품질을 찾아가고 있다. Merge sort, quick sort, paranoid quick sort 모두 best fitness값은 비슷하지만 average fitness가 가장 좋은 알고리즘은 paranoid quick sort이다. 따라서 전체적으로 5가지의 정렬 알고리즘에서 나의 GA에 가장 적합하고 좋은 정렬 알고리즘에 순위를 붙인다면, 1. Intelligent quick sort, 2. Paranoid quick sort, 3. Basic quick sort, 4. Merge sort, 5. Counting sort라고 볼 수 있다.

**<Discussion>**

assignment02는 python으로 작성했는데 생각보다 많이 느리고 이번 과제의 데이터 양이 많은 것을 보고 c++로 작성하게 되었다. Python의 코드를 C++로 변경한 후 chimera데이터를 돌려보았는데 품질이 음수에서 머물렀다. 품질이 양수가 되어야 좋은 품질임을 깨닫고 처음에는 음수 fitness 자체를 cut하는 방식으로 수정했는데 해당 방식은 데이터를 훼손하는 방식이라 생각되어 수업시간에 배운 벨만 포드를 적용시켜 음수 가중치를 다루는 방식으로 접근하게 되었다. 그러나 벨만 포드 알고리즘은 maxcut문제를 해결하는데 적합하지 않았고 GA의 연산자를 수정했다. 이후 룰렛휠 선택 연산에 지수함수를 적용시켜 선택 연산 시 fitness를 양수로 바꿔서 선택하게 하는 방식으로 수정했는데 best fitness는 세대를 거듭할수록 좋은 품질로 발전하였으나 average fitness 값이 best fitness와 차이가 컸다. 평균적인 fitness가 좋지 않아서 고민해보다 엘리티즘 비율을 조정하여 상위 개체들이 다음 세대에 보존되는 비율을 늘리면 average fitness가 좋아질 것이라 생각되어 엘리티즘 비율을 0.1부터 시작하여 하나씩 높여갔다. 엘리티즘 비율을 0.5로 설정해보니 세대를 거듭할수록 좋은 품질로 발전되는 경향이 명확했으며 best fitness 또한 더 좋은 품질의 결과를 도출했다. 이후 더 좋은 결과를 도출하기 위해 crossover 연산도 uniform crossover 연산으로 수정하고 이와 함께 선택 연산을 토너먼트 방식으로 변경해보니 훨씬 더 좋은 품질을 도출했다. 연산자들 간에 더 좋은 연산이는 없고 연산자들 간의 조합이 중요하다고 생각이 들었다. 토너먼트 방식보다 룰렛휠 방식이 더 뛰어나다고 생각을 했는데 입력하는 데이터에 따라 더 적합한 연산자가 존재하며 연산자들 사이의 조합 또한 중요하게 영향을 미친다는 사실을 알게 되었다.

-참고문헌-

현수환, 김용혁, & 서기성. (2010). [MAX CUT 문제를 위한 신장트리 기반 인코딩을 이용한 새로운 유전 알고리즘]. *한국지능시스템학회 학술발표 논문집 학술대회자료*, 20(2), 231-234.