**유전 알고리즘을 사용한 외판원문제(TSP)의 해결**

**(Traveling Salesman Problem) TSP Solution Using Genetic Algorithm**

**김유리 김윤수 이석영 장서윤**

**202138114 202035131 201935310 202135824**

**Yuri Kim Yoonsoo Kim Seokyeong Lee Seoyoon Jang**

**가천대학교 컴퓨터공학과, 심화프로그래밍**

**(지도교수: 윤유림)**

**요 약**

본 연구에서는 외판원 문제(TSP) 해결을 위해 Genetic Algorithm(GA)을 사용한다. TSP는 N개의 도시를 한 번씩 방문하는 최소 경로를 찾는 NP-Hard 문제로, 생물학적 진화의 원리를 모방한 Genetic Algorithm이 이를 해결하는 메타 휴리스틱 기법으로 활용된다. 연산자의 한계로 한계에 도달하는데 어려움이 존재했던 이전 GA의 성능을 개선하기 위해Permutation encoding, Ordinal crossover, PMX crossover, Ordered crossover, Edge Recombination, 2-OPT, 3-OPT, 5-OPT 여러가지 Method 및 Operator를 추가하고 조합한다. 성능 판단을 위해 Optimal Route이 존재하는 데이터세트 2개와 새롭게 생성한 데이터 세트 2개를 사용하여 각 16번, 총 64번의 실험을 수행했다. 그 결과, Ordered Crossover와 2-OPT 또는 3-OPT의 Local Search를 조합한 경우가 가장 우수한 성능을 나타냈다.

키워드: TSP, Crossover, Hybrid Genetic Algorithm, Local Search

**Ⅰ. 서 론**

Genetic Algorithm(GA)은 생물학적 진화 원리를 모방하여 최적화 문제를 해결하는 최적화 알고리즘의 일종으로, 메타 휴리스틱 기법 중 하나다. GA의 일반적인 구조는 다음과 같다. 우선 N개의 임의의 해를 해집단으로 생성한다. 다음 해집단으로부터 Encoding, Crossover, Mutation 등의 연산 프로세스를 거쳐 k개의 새로운 해를 생성한다. 마지막으로 k개의 해를 해집단 내의 k의 해와 대치한 후, 임의의 조건이 만족될 때까지 수행하고 해집단에 남은 해 중 가장 좋은 해를 반환하는 방식으로 동작한다. GA는 복잡하고 다양한 해를 탐색하는 데 효과적이며, 주로 최적화 문제에 활용된다.[1] Traveling Salesman Problem(TSP)은 N개의 도시와 도시 사이의 거리가 주어질 때 어떤 도시에서 시작해서 모든 도시를 단 한 번만 방문하고 다시 출발 도시로 되돌아오는 최단 거리를 찾는 문제이다. 방문 도시수가 증가함에 따라 계산량이 기하급수적으로 증가하게 되는 문제로 인해 NP-Hard Problem으로 분류된다.[2] Hybrid GA (HGA)는 GA와 다른 최적화 기법을 조합하여 적용하는 방식으로 여러 최적화 알고리즘과의 조합으로 최적해를 구한다. 각각의 최적화 기법이 가지고 있는 장점을 결합하여 최종적으로 GA의 성능 향상을 가져올 수 있다.[8]

연구의 목적은 도시 데이터 셋을 기반으로 GA를 활용하여 TSP의 최단 경로를 찾는 것이다. 뿐만 아니라 기존 GA의 한계를 극복하고 성능을 향상시키기 위한 다양한 연구를 시도함에 있다. 원래 적용했던 GA와 연산자의 한계 등으로 인한 TSP 최적해에 도달하는 데 어려움이 있다. 따라서 본 논문에서는 유전 알고리즘과 다른 최적화 기법들을 혼합한 새로운 하이브리드 알고리즘(HGA)을 설계하고 이를 평가하기 위한 실험을 수행한다. 이에 따라 GA의 성능을 개선하기 위해 GA에 여러가지 기법 및 연산자(Operator)들을 추가하고 조합하여 여러 개의 도시 데이터 셋에 적용하여 새로운 결과를 도출한다. 도출된 결과를 통해 기존 GA와 HGA를 비교함으로써 TSP 최적해를 발전시키는 데 어떠한 진전이 있는지를 확인하며, 궁극적으로는 기존 GA의 한계를 극복하고 성능을 향상시키는 것이다.

Table 1. 사용한 Crossover operator과 Local search

|  |  |
| --- | --- |
| **Crossover operator** | **Local search** |
| Ordinal crossover | No Local Search |
| PMX crossover | 2-OPT |
| Ordered crossover | 3-OPT |
| Edge Recombination | 5-OPT |

Crossover Operator는 4가지를 사용하였다. Ordinal Crossover는 각 부모로부터 일부 도시를 선택하여 그 순서를 유지한 채로 나머지 부분을 교환한다. 자식은 두 부모의 도시 순서 중 일부를 유지하면서도 새로운 도시 조합을 얻을 수 있다. PMX crossover는 일부 부모의 일치하는 구간을 선택하여 대응되는 위치에 매핑한 후, 나머지 부분을 교환한다. 중복되지 않고 두 부모의 정보가 채워져 자식 리스트를 완성한다. Ordered crossover는 두 부모에서 특정 구간을 선택한 후, 그 구간을 그대로 유지하면서 나머지 부분을 부모의 순서를 유지하면서 교환한다. 자식은 일부 부모의 특성을 유지하면서도 새로운 도시 조합을 생성할 수 있다. Edge Recombination은 부모 개체 간의 간선(도로) 정보를 활용하여 자식을 생성한다. 각 도시에 연결된 간선을 기준으로 교차를 수행하면서, 중복되지 않는 간선을 선택한다.

Local search 기법은 3가지를 사용하였다. 2-OPT 방식은 현재 경로에서 두 개의 엣지를 선택하여 교환함으로써 새로운 경로를 생성한다. 3-OPT 방식은 현재 경로에서 세 개의 엣지를 선택하여 교환함으로써 2-OPT보다 더 복잡한 엣지 교환을 통해 최적화된 해를 찾는다. 5-OPT 방식은 현재 경로에서 다섯 개의 엣지를 선택하여 교환한다. 2,3-OPT와 비교해 탐색 공간이 더 넓게 확장되어 최적해에 더 빠르게 수렴한다.

**Ⅱ. 연관 연구**

참고된 이전 연구 중 Uthayasuriyan[3]는 TSP의 해결을 위해 GA와 진화 양자 학습 알고리즘을 비교하고 있다. 실험에서는 사용자가 지정한 custom데이터와 att48 데이터가 사용되었으며, 두 알고리즘의 속도와 정확도를 평가한다. 결과는 유전 알고리즘이 TSP를 해결하는데 Epsilon Greedy Q-Learning Algorithm(EQLA)보다 더 효과적이다. 특히 EQLA는 유전 알고리즘에 비해 더 많은 시간과 메모리를 소비하며, EQLA가 찾은 솔루션은 덜 정확하고, 반면에 GA는 더 최적의 솔루션을 찾아냈다.

L. Fang[9]은 전통적인 PSO 알고리즘의 단점을 극복하기 위해 Simulated Annealing(SA)를 PSO와 결합하는 방법을 사용했다. 네 가지 알고리즘(Basic SA, Basic GA, Basic ACA, PSO-SA)을 Oliver30, Att48 데이터와 비교하며, PSO-SA 알고리즘이 가장 우수한 성능을 보여준다.

소규모, 중규모, 대규모 데이터세트(P15, ATT48, SGB128, R200(무작위 데이터세트))에 대해 Greedy 알고리즘, 2-OPT 알고리즘, 유전 알고리즘(GA)을 평가한 He [10]의 연구 결과는 다음과 같다. Greedy와 2-OPT가 작은 데이터세트에 효과적으로 솔루션을 계산하고, GA는 대규모 데이터세트에 대해 우수한 성능을 보였다.

Verma[4]는 TSP에서 최적의 경로를 찾기 위해 Directed Search Guided Optimization (DSGO) 기법을 사용하고 GA및 Discrete Particle Swarm Optimization(DPSO)과 비교하는 연구를 진행했다. 결과는 DSGO가 수렴 및 최적의 솔루션을 찾는 측면에서 GA, DPSO보다 뛰어난 성능을 보였고, d0, p01, gr17, fri26, dantzig42와 같은 다양한 데이터를 사용했다.

TSP 해결을 위해 다양한 데이터세트(ftv33, dantzig42, ft53, rbg323 등 12가지)에 GA를 적용한 Hussain[11]의 연구가 존재한다. 기존 GA에 새로운 교차 연산자들(CX2, PMX, OX)을 소개했다. 성능 비교 결과, CX2가 PMX 및 OX보다 우수한 성능을 나타냈다.

본 연구에서는 관련 연구에서 수행한 GA와 본 연구에서 수행하는 GA의 성능을 비교한다. 또한 최적해가 존재하는 데이터인 att48과 dantzig42를 이용하여 이전 연구의 거리 비교와 같은 성능 평가를 진행한다. 또한 최적해가 없는 사용자가 지정하는 랜덤 데이터 2개를 생성하여 성능을 제시하고 GA의 유효성을 확인한다.

**Ⅲ. 방법론**

**3.1. Permutation Encoding**

Permutation Encoding방식은 문제의 해를 순열로 표현하는 방식이다. 본 연구에서는 경로를 순열로 표현하였다. 사용하는 데이터의 도시 개수를 인자로 받아 도시를 방문하는 무작위 순열을 생성한다.

**3.2. Rank**

Rank 방식은 각 개체의 상대적인 우수성에 따라 순위를 매겨 선택하는 방식이다. 이 방식은 적합도(fitness)가 높은 개체에 먼저 순위를 부여한다. 적합도 계산 방식은 다음과 같다. 개체의 거리를 구한 뒤 거리 값의 역수를 취하여 적합도를 정한다. 따라서 개체의 거리와 적합도가 반비례한다. 이렇게 구해진 적합도는 내림차순으로 정렬되어 우수성에 따른 순위가 매겨진다. 이후 사용자가 입력한 엘리트 개체만큼 선택되어 다음 개체에 적용된다.

**3.3. Roulette wheel selection**

Roulette wheel selection은 높은 적합도를 가진 개체일수록 선택 확률이 높아지는 특징을 가진 방식이다. 앞서 언급된 엘리트 개체를 제외한 나머지 개체를 형성하는데 사용된다. 개체들의 적합도에 따라 선택 확률을 할당한다. 0부터 100까지의 랜덤 값을 부여하고 만약 랜덤 값이 확률보다 작거나 같으면 해당 개체가 선택된 것으로 간주하고 그 개체의 인덱스를 추가한다. 중복을 방지하기 위해 선택된다면 해당 루프를 더 이상 진행하지 않는다.

**3.4. Crossover**

Crossover 방식은 Ordinal Crossover, PMX, Ordered Crossover, Edge Recombination 총 4 가지 방법을 사용했다.

*3.4.1 Ordinal Crossover*

|  |
| --- |
| **Pseudocode** |
| **function** ordinal\_crossover(parent1, parent2, crossover\_point):  offspring = empty array of the same length as parents  copy elements before crossover\_point from parent1 to offspring  fill remaining positions in offspring with elements from parent2 not  in offspring  **return** offspring |

네 가지 방법 중에 가장 간단한 Ordinal crossover는 무작위로 교차점을 선택하고, 교차점 앞의 원소들은 한 부모로부터 복사되고, 나머지 원소들은 다른 부모에서 나타나는 순서대로 중복되지 않고 채워진다. 이전 부모 염색체의 순서를 그대로 보존하는 방식이다.

*3.4.2 Partially Matched Crossover (PMX)*

|  |
| --- |
| **Pseudocode** |
| **function** pmx\_crossover(parent1, parent2, crossover\_point1, crossover\_point2):  offspring = empty array of the same length as parents  copy elements between crossover\_point1 and crossover\_point2  from parent1 to offspring  for each unmatched element between crossover\_point1 and  crossover\_point2 in parent2:  if the element is not in offspring:  map it to the corresponding position in parent1  fill that position in offspring with the mapped element  fill remaining positions in offspring with elements from parent2 not  in offspring  **return** offspring |

두 개의 교차점을 선택하고, 부모1의 교차점 사이 Subset 1의 염색체를 그대로 자손에게 복사를 한다. 그 이후 나머지 염색체들은 부모 2에게서 복사를 하는데, 그 과정에서 Subset 1과 중복되는 염색체는 순서대로 부모 2 교차점 사이의 Subset 2의 유전자와의 Mapping을 통해서 중복을 해소한다.[6]

*3.4.3 Ordered Crossover*

|  |
| --- |
| **Pseudocode** |
| **function** ordered\_crossover(parent1, parent2, subset\_start, subset\_end):  offspring = empty array of the same length as parents  copy subset of genes from parent1 to offspring (from subset\_start  to subset\_end)  fill remaining positions in offspring with genes from parent2 not in    offspring  **return** offspring |

두 개의 교차점을 선택하고, 부모 1의 Subset을 그대로 자손에게 추가를 한 후 부모 2에게서 Subset 속 염색체와 중복되지 않는 나머지 염색체들을 순서대로 자손에게 추가를 한다.

*3.4.4 Edge Recombination*

|  |
| --- |
| **Pseudocode** |
| **function** edge\_recombination(parent1, parent2):  offspring = empty array of the same length as parents  create pool of edges from parents  start with a random vertex from parent1  add the vertex to offspring  remove the vertex and its edges from the pool  repeat until the offspring is complete:  find the nearest neighbor of the last vertex added to offspring  add the nearest neighbor to offspring  remove the nearest neighbor and its edges from the pool  **return** offspring |

TSP 해결을 위해 고안된 Edge Recombination은 부모의 엣지 정보만을 이용해서 자손을 생성한다. 무작위로 하나의 도시를 선택 후 선택된 도시 와 연결 가능한 도시들 중에서 엣지 표에 남아 있는 엣지의 개수가 가장 작은 도시를 선택한다. 이런 과정을 자손이 채워질 때까지 반복한다.[7]

**3.5. Mutation**

Local search 적용 여부에 따라 사용되는 Swap Mutation 방식이 두가지 있다. 첫 번째로, Local search가 적용되지 않았을 때의 Mutation 방식이다. 랜덤으로 추출한 값이 사용자에게 입력 받은 주어진 확률보다 작을 때 Mutation이 실행된다. 각 유전자에 대해 무작위로 Mutation을 진행하며 선택된 다른 유전자와 교환한다.

두 번째로, Local search가 적용되었을 때의 Mutation 방식이다. 이 또한 랜덤으로 추출한 값이 사용자에게 입력 받은 주어진 확률보다 작을 때 Mutation이 실행된다. Local search는 K-OPT 방식으로 무작위로 선택된 K개 지점을 선택하여 K-OPT 함수를 호출하고 Mutation 개체를 생성한다.

**3.6. Local search**

Local search는 K-OPT로 k개의 노드를 교환하며 최적해를 찾는 방식이다. 적용된 방식은 3가지로 2-OPT, 3-OPT, 5-OPT를 사용한다.

K-OPT 알고리즘은 주어진 경로에서 i부터 k까지의 도시를 선택하고 이 부분을 {i, i+1, …, k}로 표현한다. 그리고 선택한 도시 부분을 뒤집어 경로의 일부분이 {k, k-1, …, i}로 바뀐다. 이후 두 부분인 {i, i+1, …, k}와 {k, k-1, …, i}을 이어 붙여 새로운 경로를 생성하고 새로운 경로의 길이가 더 짧으면 현재 경로로 설정한다. 아래는 본 설명을 기반으로 실행된 2-OPT, 3-OPT, 5-OPT이다.

*3.6.1. 2-OPT*

임의의 두 도시를 선택하고 교환하여 새로운 경로를 생성한다. 만약 새로운 경로가 더 짧다면 이를 현재 경로로 대체한다.

|  |
| --- |
| **Pseudocode** |
| **procedure** 2optSwap(route, v1, v2) {  1. take route[0] to route[v1] and add them in order to new\_route  2. take route[v1+1] to route[v2] and add them in reverse order to  new\_route  3. take route[v2+1] to route[start] and add them in order to  new\_route  **return** new\_route;  } |

*3.6.2. 3-OPT*

임의의 세 도시를 선택하고 교환하여 새로운 경로를 생성한다. 만약 새로운 경로가 더 짧다면 이를 현재 경로로 대체한다.

|  |
| --- |
| **Pseudocode** |
| **procedure** three\_opt\_swap(route, v1, v2, v3){  1. take route[0] to route[v1] and add them in order to new\_route  2. take route[v2+1] to route[v3] and add them in reverse order to  new\_route  3. take route[v1+1] to route[v2] and add them in order to  new\_route  4. take route[v3+1] to route[start] and add them in order to  new\_route  5. **return** new\_route;  } |

*3.6.3. 5-OPT*

임의의 다섯 도시를 선택하고 교환하여 새로운 경로를 생성한다. 만약 새로운 경로가 더 짧다면 이를 현재 경로로 대체한다.

|  |
| --- |
| **Pseudocode** |
| **procedure** five\_opt\_swap(route, v1, v2, v3, v4, v5){  1. take route[0] to route[v1] and add them in order to new\_route  2. take route[v4+1] to route[v5] and add them in reverse order to  new\_route  3. take route[v3+1] to route[v4] and add them in order to  new\_route  4. take route[v2+1] to route[v3] and add them in reverse order to  new\_route  5. take route[v1+1] to route[v2] and add them in order to  new\_route  6. take route[v5+1] to route[start] and add them in order to  new\_route  7. **return** new\_route;  } |

**3.7. Replacement**

Replacement는 다음 세대의 개체를 선택하는 방식이며 Rank의 결과에 따라 Elitism이 적용된다. Rank 방식이 반환한 현재 세대의 경로들의 순위를 얻는다. Elite 개체를 상위 순위부터 선택한 뒤, 다음 세대 개체에 포함시킨다. 남은 개체들은 앞서 언급된 연산을 통해 새로운 개체를 생성한다. 이렇게 얻은 결과로 다음 세대의 개체를 반환한다.

**3.8. Parameter**

GA 실행을 위해 5가지 입력을 받는다. 데이터, 한 세대 당 개체 수, 엘리트 개체 수, Mutation 비율, 전체 세대 수로 구성되어 있다. 데이터는 실험에 따라 변경되지만 그 외 4가지는 각각 아래와 같이 고정한다.

Table 2. Parameter set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| routeLength | popSize | eliteSize | mutationRate | generations |
| Input data | 100 | 20 | 0.01 | 500 |

**Ⅳ. 실험 구성**

성능 평가를 위해 새로운 데이터 세트 4개를 구축한다. 먼저, 최적해가 존재하는 데이터 2개이다.[5] 첫 번째는 US의 48개의 수도 데이터로 att48의 명칭을, 두 번째는 US의 42개 도시 데이터로 dantzig42명칭을 갖고 있다. 각 10628과 699의 최적해를 가진다. 다음으로 최적해가 존재하지 않는 커스텀 데이터 2개이다. 첫번째는 25개의 노드를 가지며 custom25 명칭을, 두 번째는100개의 노드를 가지며 custom100 명칭을 갖고 있다. 4개 데이터 모두 X, Y 좌표를 나타낸다.

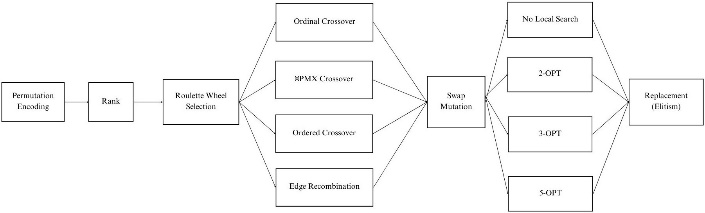


Figure 1. Method overview

Table 3. Datasets

|  |  |  |
| --- | --- | --- |
| Instances | Nodes | Optimal Solution |
| att48 | 48 | 10628 |
| dantzig42 | 42 | 699 |
| custom25 | 25 | X |
| Custom100 | 100 | X |

실험 가짓수는 각 데이터 당 16개이다. Permutation encoding, Rank, Roulette wheel selection, Swap mutation, Replacement는 모두 동일하게 사용되며 변경 여부는 Crossover와 Local search에 각각 4가지 존재한다. Crossover는 Ordinal crossover, PMX crossover, Ordered crossover, Edge recombination crossover이 시행된다. Local search은 2-OPT, 3-OPT, 4-OPT, 5-OPT가 있으며 마지막으로 Local search를 적용하지 않은 경우 또한 시행한다. 모든 실험은 데이터 시각화를 지원하는 오픈 소스 웹 응용 프로그램인 주피터 노트북에서 시행되며 파이썬을 사용한다.

**Ⅴ. 결론**

총 64번의 실험들의 Initial Distance와 Final Distance를 출력하여 가장 최단 거리의 경로를 출력하는 조합을 찾고 비교했다.

Optimal이 존재하는 att48 데이터에는 Ordered crossover + 2-OPT Local search 조합을, 그리고 dantzig42 데이터에는 Ordered crossover + 3-OPT Local search 조합을 적용한 결과, att48에서는 40263.0438의 최단 거리를, dantzig42에서는 726.0249의 최단 거리를 얻었다.

Optimal이 존재하지 않는 custom25 데이터에는 Ordinal crossover + 2-OPT Local search 조합을, 그리고 custom100 데이터에는 Ordered crossover + 3-OPT Local search 조합을 적용한 결과, custom25에서는 8645.9648의 최단 거리를, custom100에서는 29624.6202의 최단 거리를 얻었다.

이전의 att48 데이터를 유전 알고리즘 최적화 연구로 37,076.68의 거리를 얻은 연구와 비교했을 때 본 연구의 거리 (40263.0438)가 더 길었음을 알 수 있다.[3]

하지만dantzig42데이터의 최적화를 목표로 한 연구와 비교했을 때, 이전 연구에서 제시한 유전 알고리즘으로 얻은 거리 929.28와 그것보다 성능이 좋다고 제시된 DCGO (Discreet Social Group Optimization)을 이용한 거리인 771.35 보다 본 연구가 726.0249로 더 짧은 거리를 얻었다.[4]

여러 차례의 실험 후, Crossover는 대부분 Ordered crossover가 가장 좋은 성능을 보여주었고, Local search의 유무는 결과에 큰 차이를 일으켰다. 하지만 5-OPT Local search보다는 Edge를 더 적게 변경하는 2-OPT 및 3-OPT 알고리즘이 더 효과적인 것을 알 수 있다. 이하는 64개의 실험 정보 및 결과를 나타낸 표이다.

Table 4. att48 results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Crossover** | **Local Search** | **Initial Distance** | **Final Distance** | **Rank** |
| ORL | None | 134953.2088 | 48607.8326 | 9 |
| ORL | 2-OPT | 134983.9370 | 56907.1664 | 13 |
| ORL | 3-OPT | 114501.5100 | 53105.4358 | 11 |
| ORL | 5-OPT | 123274.0662 | 61897.4595 | 15 |
| PMX | None | 132818.0549 | 42900.97601 | 2 |
| PMX | 2-OPT | 128951.6629 | 43933.9950 | 5 |
| PMX | 3-OPT | 132160.0394 | 45285.4069 | 7 |
| PMX | 5-OPT | 135053.9666 | 42985.0507 | 3 |
| ORD | None | 132644.6321 | 44543.4156 | 6 |
| **ORD** | **2-OPT** | **127186.84611** | **40263.0438** | **1** |
| ORD | 3-OPT | 130845.1381 | 43596.3433 | 4 |
| ORD | 5-OPT | 134393.4085 | 46058.1596 | 8 |
| ER | None | 125026.6859 | 49376.37056 | 10 |
| ER | 2-OPT | 119136.3573 | 53963.6236 | 12 |
| ER | 3-OPT | 127218.1160 | 59137.3081 | 14 |
| ER | 5-OPT | 130220.2334 | 69192.0688 | 16 |

Table 5. dantzig42 results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Crossover** | **Local Search** | **Initial Distance** | **Final Distance** | **Rank** |
| ORL | None | 2402.2604780424417 | 1137.7982091673034 | 12 |
| ORL | 2-OPT | 2268.4199923873666 | 738.4010932574884 | 2 |
| ORL | 3-OPT | 2384.066035036455 | 778.7934966538793 | 3 |
| ORL | 5-OPT | 2392.3033533648804 | 1783.7420025730894 | 16 |
| PMX | None | 2421.1309920081826 | 1072.074641847386 | 11 |
| PMX | 2-OPT | 2448.7822720105546 | 872.6474983348434 | 7 |
| PMX | 3-OPT | 2521.3242068103946 | 857.1693067038971 | 6 |
| PMX | 5-OPT | 2334.541297749859 | 892.626881752271 | 9 |
| ORD | None | 2449.8534038236016 | 827.1613300595646 | 4 |
| ORD | 2-OPT | 2415.8562000336706 | 1233.9558014396894 | 13 |
| **ORD** | **3-OPT** | **2431.503911719786** | **726.0249016499029** | **1** |
| ORD | 5-OPT | 2582.8175407611584 | 837.8938719192383 | 5 |
| ER | None | 2407.7695134991877 | 1052.6411186651198 | 10 |
| ER | 2-OPT | 2358.7670430760377 | 1497.4951492911541 | 15 |
| ER | 3-OPT | 2321.3433737090045 | 1240.8186251661232 | 14 |
| ER | 5-OPT | 2367.3163564756705 | 945.0971764670678 | 8 |

Table 6. custom25 results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Crossover** | **Local Search** | **Initial Distance** | **Final Distance** | **Rank** |
| ORL | None | 21228.525195 | 9723.892257 | 6 |
| **ORL** | **2-OPT** | **21652.426269** | **8645.964769** | **1** |
| ORL | 3-OPT | 20780.325253 | 9696.508252 | 4 |
| ORL | 5-OPT | 21325.854175 | 10304.032054 | 11 |
| PMX | None | 21242.046180 | 9525.110426 | 3 |
| PMX | 2-OPT | 20593.325962 | 10066.708696 | 8 |
| PMX | 3-OPT | 20975.453314 | 10208.286956 | 10 |
| PMX | 5-OPT | 22280.619633 | 9763.306627 | 5 |
| ORD | None | 22027.682312 | 9854.607646 | 7 |
| ORD | 2-OPT | 21195.734294 | 9361.988903 | 2 |
| ORD | 3-OPT | 20873.229102 | 10191.553544 | 9 |
| ORD | 5-OPT | 20782.435136 | 10417.643032 | 12 |
| ER | None | 22431.057443 | 11270.956292 | 14 |
| ER | 2-OPT | 20724.676919 | 10709.662816 | 13 |
| ER | 3-OPT | 21007.298973 | 11936.248381 | 16 |
| ER | 5-OPT | 21734.618940 | 11624.978312 | 15 |

Table 7. custom100 results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Crossover** | **Local Search** | **Initial Distance** | **Final Distance** | **Rank** |
| ORL | None | 91951.32862971921 | 56429.34044740168 | 13 |
| ORL | 2-OPT | 95086.4569602524 | 34041.747541915356 | 5 |
| ORL | 3-OPT | 95218.46768130791 | 34128.63198388943 | 6 |
| ORL | 5-OPT | 93779.12210774641 | 34398.41384866743 | 8 |
| PMX | None | 95907.71788974172 | 63615.3452198741 | 15 |
| PMX | 2-OPT | 93055.39774016943 | 38928.09062981483 | 9 |
| PMX | 3-OPT | 88566.24250522684 | 34000.13585930772 | 4 |
| PMX | 5-OPT | 95285.9317452007 | 34267.98076179785 | 7 |
| ORD | None | 94661.62924760714 | 67100.79953727724 | 16 |
| ORD | 2-OPT | 92918.52184496356 | 32784.62689488955 | 3 |
| **ORD** | **3-OPT** | **93303.8286716847** | **29624.62023605982** | **1** |
| ORD | 5-OPT | 88391.23130632387 | 31936.046903294788 | 2 |
| ER | None | 93348.21233622456 | 46796.22729142792 | 10 |
| ER | 2-OPT | 92493.53420896742 | 62263.27367801268 | 14 |
| ER | 3-OPT | 92731.62901297462 | 54923.541760139655 | 12 |
| ER | 5-OPT | 91877.40104652107 | 49121.48231139381 | 11 |

본 연구를 참고해 더 많은 Local search 방식과 적절한 Parameter 조정을 하고 이전보다 빠르고 정확하게 최적을 구하는 방향으로 발전시킬 수 있다. 다양한 유전알고리즘을 설계하고 평가함으로써 TSP문제를 비롯한 더 복잡한 NP-완전 군의 문제들을 해결할 수 있음을 기대한다.

**참고문헌**

[1] Bryant, Kylie, "Genetic Algorithms and the Travelling Salesman Problem" (2000). *HMC Senior Theses*. 126.

[2] 강래구, 임희경 and 정채영. (2006). 최적의 TSP문제 해결을 위한 유전자 알고리즘의 새로운 집단 초기화 및 순차변환 기법. *한국정보통신학회논문지*, 10(3), 622-627.

[3] Uthayasuriyan, Agash & G, Hema & UV, Kavvin & Mahitha, Sabbineni & Jeyakumar, Gurusamy. (2023). A Comparative Study on Genetic algorithm and Reinforcement Learning to Solve the Traveling Salesman Problem. *Research Reports on Computer Science*. 1-12.

[4] Verma, S., Jena, J.J., Satapathy, S.C., & Rout, M. (2020). Solving Travelling Salesman Problem using Discreet Social Group Optimization. *Journal of Scientific & Industrial Research*.

[5] Reinelt, G. "TSPLIB--A Traveling Salesman Problem Library." *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 376-384. Fall 1991.

[6] 이강구, 한승기, 이성환. (1995). TSP를 위한 유전자 알고리즘. *(구)정보과학회논문지*, *22*(4), 559-566.

[7] 석상문, 안병하. (2003). 조합최적화 문제를 위한 새로운 유전연산자. *한국정보과학회 학술발표논문집*, 개최지.

[8] 김정숙, 홍영식. (1998). 외판원 문제(TSP)를 위한 분산 하이브리드 알고리즘. *정보과학회논문지*(A), 25(2), 136-144.

[9] L. Fang, P. Chen, and S. Liu, “Particle swarm optimization with simulated annealing for tsp,” in *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '07)*, pp. 206–210, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wis, USA, 2007.

[10] He, Y., & Xiang, M. (2017). An empirical analysis of approximation algorithms for the euclidean traveling salesman problem. *arXiv preprint arXiv:1705.09058*.

[11] Abid Hussain, Yousaf Shad Muhammad, M. Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, Showkat Gani, "Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator", *Computational Intelligence and Neuroscience*, vol. 2017, Article ID 7430125, 7 pages, 2017.

**추가자료**

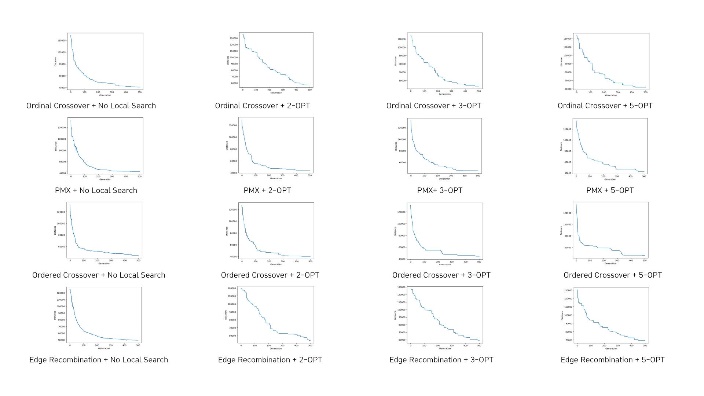


Figure 2. att48 Shortest Distance per Generation

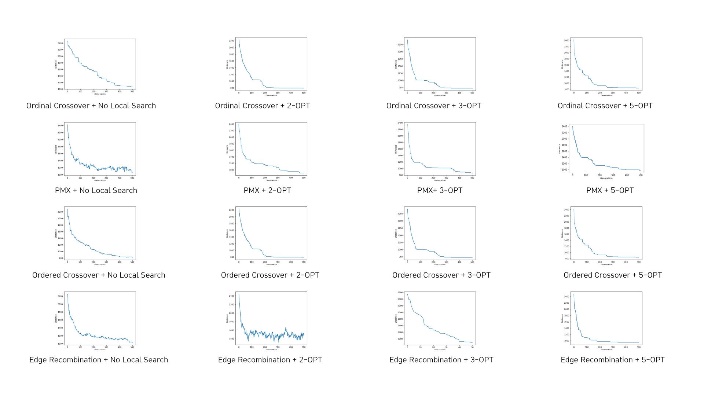


Figure 3. dantzig42 Shortest Distance per Generation

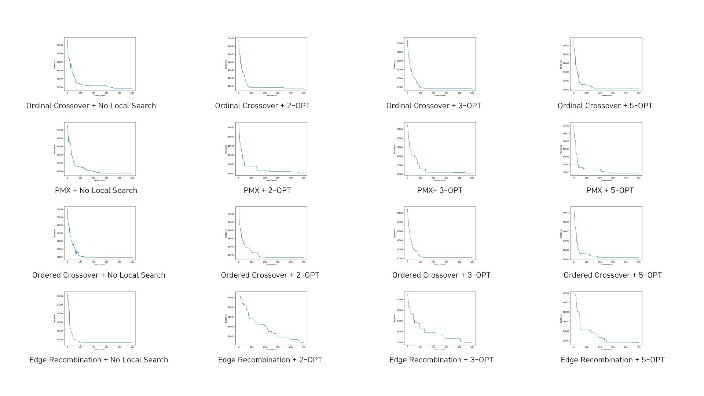


Figure 4. custom25 Shortest Distance per Generation

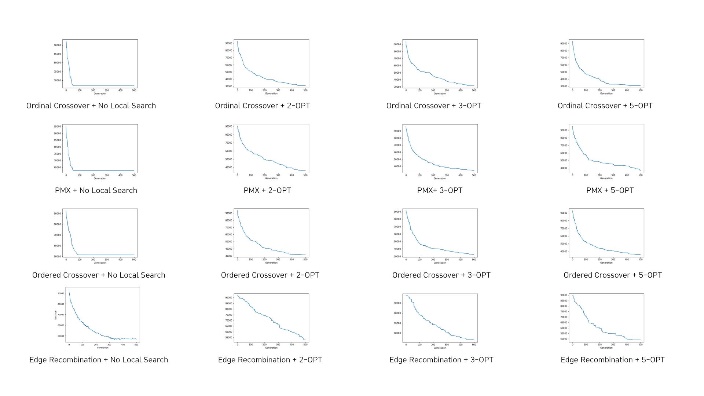


Figure 5. custom100 Shortest Distance per Generation