JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 9, pp. 69-76, September 2022
https://doi.org/10.9708/jksci.2022.27.09.069

Greedy-based Neighbor Generation Methods of Local Search for the Traveling Salesman Problem

Junha Hwang*, Yongho Kim*

*Professor, Dept. of Computer Engineering, Kumoh National Institute of Technology, Gumi, Korea
*Student, Dept. of Computer Engineering, Kumoh National Institute of Technology, Gumi, Korea

[Abstract]

The traveling salesman problem(TSP) is one of the most famous combinatorial optimization problem. So far, many metaheuristic search algorithms have been proposed to solve the problem, and one of them is local search. One of the very important factors in local search is neighbor generation method, and random-based neighbor generation methods such as inversion have been mainly used. This paper proposes 4 new greedy-based neighbor generation methods. Three of them are based on greedy insertion heuristic which insert selected cities one by one into the current best position. The other one is based on greedy rotation. The proposed methods are applied to first-choice hill-climbing search and simulated annealing which are representative local search algorithms. Through the experiment, we confirmed that the proposed greedy-based methods outperform the existing random-based methods. In addition, we confirmed that some greedy-based methods are superior to the existing local search methods.

▸ **Key words**: Neighbor generation, Greedy-based, Local search, Traveling salesman problem, Simulated annealing

[요 약]

순회 외판원 문제는 가장 유명한 조합 최적화 문제 중 하나이다. 지금까지 이 문제를 해결하기 위해 많은 메타휴리스틱 탐색 알고리즘들이 제안되어 왔으며, 그중의 하나가 지역 탐색이다. 지역 탐색에 있어서 매우 중요한 요소 중 하나가 이웃해 생성 방법으로 주로 역전(inversion)과 같은 무작위 기반 이웃해 생성 방법들이 사용되어 왔다. 본 논문에서는 4가지의 새로운 그리디 기반 이웃해 생성 방법들을 제안한다. 3가지 방법은 그리디 삽입 휴리스틱을 기반으로 하는데, 선택된 도시들을 하나씩 차례로 현재 가장 좋은 위치로 삽입한다. 나머지 하나는 그리디 회전을 기반으로 한다. 제안된 방법들은 대표적인 지역 탐색 알고리즘인 first-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링에 적용된다. 실험을 통해 제안된 그리디 기반 방법들이 기존의 무작위 기반 방법들보다 성능이 우수함을 확인하였다. 또한 일부 그리디 기반 방법들은 기존의 지역 탐색 기법들보다 더 우수함을 확인하였다.

▸ **주제어**: 이웃해 생성, 그리디 기반, 지역 탐색, 순회 외판원 문제, 시뮬레이티드 어닐링

• First Author: Junha Hwang, Corresponding Author: Junha Hwang
*Junha Hwang (jhhwang@kumoh.ac.kr), Dept. of Computer Engineering, Kumoh National Institute of Technology
*Yongho Kim (dokebi@kumoh.ac.kr), Dept. of Computer Engineering, Kumoh National Institute of Technology
• Received: 2022. 08. 23, Revised: 2022. 09. 16, Accepted: 2022. 09. 16.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

순회 외판원 문제는 n개의 도시와 도시 사이의 거리가 주어진 경우 모든 도시들을 단 한 번만 방문하고 출발 도시로 되돌아오되 최단 경로를 찾는 문제로 정의된다. 순회 외판원 문제는 도시의 개수가 증가함에 따라 탐색 공간의 크기가 기하급수적으로 증가하게 된다. 순회 외판원 문제는 NP-hard 문제로 알려져 있으며, 이에 따라 오랫동안 컴퓨터 공학 분야의 최적화 기법을 테스트하기 위한 문제로 많이 활용되어 왔다[1]. 또한 순회 외판원 문제를 위한 해법은 차량 경로 문제나 로봇 경로 계획 문제와 같은 실세계 문제로의 적용이 가능하여 산업적으로도 큰 가치가 있다[2, 3].

지금까지 순회 외판원 문제를 해결하기 위해 유전 알고리즘, Ant 시스템, 시뮬레이티드 어닐링 등 다양한 최적화 기법들이 제시되어 왔다. 그중에서 유전 알고리즘이나 Ant 시스템과 같은 집단 기반 탐색 기법들의 성능이 우수한 것으로 알려져 있다[4, 5]. 이는 현재 우수한 경로들의 공통된 특징을 기반으로 새로운 해를 만들 수 있기 때문인 것으로 해석된다. 반면에 시뮬레이티드 어닐링과 같은 지역 탐색은 집단 기반 탐색에 비해 성능이 저조한 것으로 나타나고 있다. 지역 탐색은 하나의 해를 계속해서 수정해 나가기 때문에 집단 기반 탐색과 달리 해들의 공통된 특징을 활용할 수가 없다.

본 논문에서는 순회 외판원 문제를 위한 지역 탐색의 성능 향상 방안으로 그리디 기반 이웃해 생성 방법들을 제시한다. 이웃해 생성 방법은 지역 탐색 기법들에 있어서 매우 중요한 요소 중 하나이다. 지금까지의 많은 연구에서 사용한 대표적인 이웃해 생성 방법으로는 Swap, Inversion, EdgeInsertion, BlockInsertion이 있다[6, 7]. Swap은 임의의 2개 도시의 위치를 교환하는 것이고 Inversion은 임의의 2개 도시 사이의 도시들을 역전시키는 것이다. EdgeInsertion은 무작위로 하나의 도시를 선택하여 임의의 위치로 삽입하는 것이고 BlockInsertion은 임의의 2개 도시 사이의 도시들을 다른 임의의 위치로 이동하는 것이다. 4가지 방법 모두 무작위 기반(random-based) 방법이라 할 수 있다. 한편 기존 연구[8]에서는 이웃해 생성 방법으로 BlockSwap, Rotation, RandomShuffle, GreedyOrdering을 추가로 제시하였다. 이 중에서 GreedyOrdering만 그리디 기반(greedy-based) 방법이며 나머지는 모두 무작위 기반 방법이다.

본 논문에서는 총 4가지의 새로운 그리디 기반 이웃해 생성 방법을 제시한다. 먼저 기존의 BlockInsertion을 수정하여 만든 GreedyBlockWholeInsertion과 GreedyBlockEachInsertion이 있으며, 기존의 Rotation을 수정하여 만든 GreedyBlockRotation이 있다. 나머지 하나는 기존의 RandomShuffle을 수정하여 만든 GreedyRandomInsertion이다. 4가지 방법 모두 그리디 휴리스틱을 기반으로 만든 여러 개의 후보해들 중 가장 좋은 후보해를 다음해로 선택한다. 각 이웃해 생성 방법들은 대표적인 지역 탐색 기법인 First-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링에 적용된다. 이를 통해 무작위 기반 이웃해 생성 방법들과 그리디 기반 이웃해 생성 방법들의 성능을 비교하며, 아울러 기존의 지역 탐색 기법들과도 그 성능을 비교 검토한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 지역 탐색과 순회 외판원 문제에 대해 기술한다. 3장에서는 본 논문에서 제안하는 그리디 기반 이웃해 생성 방법들에 대해 자세히 설명한다. 4장에서는 실험 결과를 제시하고 분석하며, 마지막으로 5장에서 결론 및 향후 과제에 대해 논의한다.

II. Related Works

1. Local Search

지역 탐색은 하나의 해로부터 출발하여 이웃해로 이동하면서 더 좋은 해를 도출하는 탐색 기법이다. 가장 기본적인 지역 탐색 기법으로는 언덕 오르기 탐색이 있으며, 언덕 오르기 탐색은 크게 Steepest-ascent 언덕 오르기 탐색과 First-choice 언덕 오르기 탐색으로 나뉜다. Steepest-ascent 언덕 오르기 탐색은 모든 이웃해들 중 가장 좋은 해로 이동하며, First-choice 언덕 오르기 탐색은 이웃해를 하나씩 생성하면서 더 좋은 해가 나타나면 바로 이동한다. 두 가지 기법 모두 현재해보다 더 좋은 해가 존재하지 않으면 현재해를 최종 도출해로 반환한다.

언덕 오르기 탐색은 전역 최적해에 도달하지 못하고 지역 최적해에 머문다는 단점이 있다. 이를 개선하기 위해 등장한 탐색 기법으로 타부 탐색과 시뮬레이티드 어닐링이 있다. 타부 탐색은 Steepest-ascent 언덕 오르기 탐색을 기반으로 하되 타부 리스트에 이전 해의 특징을 저장함으로써 방문했던 해로 되돌아가는 것을 방지한다[9]. 이 과정에서 현재해보다 좋지 않은 해로의 이동을 허용한다. 시뮬레이티드 어닐링은 First-choice 언덕 오르기 탐색을 기반으로 하되 이웃해가 현재해보다 좋지 않더라도 주어진 확률에 따라 이동할 수 있다[10].

Steepest-ascent 언덕 오르기 탐색과 타부 탐색은 기본적으로 모든 이웃해들을 대상으로 가장 좋은 해를 선택한다. 그런데, 본 연구에서 제시하는 이웃해 생성 방법들에 의해 생성되는 이웃해들의 개수는 매우 많다. 예를 들어, 이웃해 생성 방법 중 GreedyRandomInsertion은 전체 도시들 중 임의의 개수만큼 도시들을 선택한 후 다시 삽입하는 과정을 거치게 되는데, 도시 개수가 100개인 경우 임의의 개수만큼 도시를 선택하는 방법만 하더라도 약 1.267×10^{30}개나 된다. 따라서 본 연구에서는 이웃해 생성 방법들의 효과를 검증하기 위한 지역 탐색 기법으로 First-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링을 사용한다.

시뮬레이티드 어닐링의 기본 알고리즘은 Fig. 1과 같다[8]. 참고로 Fig. 1은 최소화 문제를 기준으로 기술한 것이다. Fig. 1에서 9~12라인을 제외하면 First-choice 언덕 오르기 탐색이 된다. 즉, 시뮬레이티드 어닐링은 이웃해가 현재해보다 좋지 않더라도 10라인과 같이 $e^{-\Delta E/T}$의 확률로 이동하게 된다.

```
 1: Algorithm SimulatedAnnealing
 2: current ← Make an initial solution
 3: T ← Tstart
 4: while stopping criterion is not satisfied do
 5:     next ← Generate a neighbor solution
 6:     ΔE ← ObjValue(next) - ObjValue(current)
 7:     if ΔE ≤ 0 then
 8:         current ← next
 9:     else
10:         current ← next only with probability e^(-ΔE/T)
11:     T ← Update temperature
12:     if T < Tmin then T = Tmin
13: return current
```

Fig. 1. Basic Simulated Annealing Algorithm

시뮬레이티드 어닐링의 성능에 가장 큰 영향을 미치는 요소로는 이웃해 생성 방법과 더불어 온도 스케줄링이 있다. 온도 T에 의해 현재해보다 좋지 않은 해로의 이동 확률이 결정되는데 T 값이 클수록 이동 확률은 커지게 된다. 탐색이 진행될수록 T 값을 점점 감소시켜 나가는데, 온도를 감소시키는 방식을 온도 스케줄링이라 한다. 본 연구에서는 이웃해 생성 방법의 효과에 초점을 맞추기 위해 온도 스케줄링 방식으로는 가장 기본적인 방법인 Geometric 방식을 사용한다[11]. 이는 식 1과 같으며, α 값은 보통 0.5와 1 사이의 상수값을 사용한다. 이를 포함하여 Fig. 1에 존재하는 상수 파라미터인 Tstart, Tmin, α 값은 실험을 통해 결정하였다.

$$T_{k+1} = \alpha T_k \tag{1}$$

2. Traveling Salesman Problem

순회 외판원 문제를 해결한다는 것은 다음 두 가지 중 하나라 할 수 있다. 첫 번째는 경로 생성이고, 두 번째는 경로 개선이다. 경로 생성은 첫 번째 도시부터 다음으로 이동할 도시를 하나씩 선택하다가 하나의 해, 즉 경로가 완성되면 알고리즘이 종료된다. 경로 개선은 무작위 또는 경로 생성 알고리즘을 통해 초기해를 생성한 후 해를 반복적으로 개선하는 방식으로 시뮬레이티드 어닐링, 유전 알고리즘 등 다양한 탐색 기법들이 활용될 수 있다.

본 연구에서는 경로 생성 알고리즘들 중 그리디 삽입 휴리스틱의 일종인 Nearest Insertion 알고리즘을 활용한다[12]. 구체적으로는 본 연구에서 제시하는 4가지 이웃해 생성 방법들 중 GreedyBlockRotation을 제외한 3가지 방법에서 Nearest Insertion 알고리즘을 활용한다. Nearest Insertion 알고리즘에서는 가장 짧은 엣지(edge)를 포함하는 2개의 도시로 이루어진 경로로부터 출발하여 이외의 도시들을 하나씩 경로에 추가해 나간다. 이때 현재 경로(tour)로부터 거리가 가장 짧은 도시를 먼저 선택하고, 해당 도시를 현재 경로로 추가하되 거리가 가장 적게 늘어나는 엣지로 추가한다. 본 연구에서도 이웃해 생성 시 선택된 도시를 현재 경로로 추가할 때 거리가 가장 적게 늘어나는 엣지로 추가한다.

경로 개선 알고리즘은 시뮬레이티드 어닐링과 같은 지역 탐색을 포함한다. 지역 탐색에 있어서 가장 중요한 요소는 이웃해 탐색 방법이라 할 수 있다. 기존 연구 [8]에서는 전통적인 이웃해 생성 방법을 포함하여 다음과 같은 다양한 이웃해 생성 방법들을 제시하고 비교하였다.

- **Swap** : 2개의 위치를 무작위로 선택하여 서로 교환한다.
- **Inversion** : 2개의 위치를 무작위로 선택하고 그 사이에 있는 값들을 역전시킨다.
- **EdgeInsertion** : 1개의 위치를 무작위로 선택하고 해당 값을 임의의 위치에 삽입한다.
- **BlockInsertion** : 무작위로 2개의 위치를 선택하고 사이에 있는 값들을 그대로 임의의 다른 위치로 삽입한다.
- **BlockSwap** : 4개의 위치를 무작위로 선택하고 첫 번째와 두 번째 위치 사이의 값들을 세 번째와 네 번째 사이의 값들과 교환한다.
- **RandomShuffle** : k개의 위치를 무작위로 선택하고 해당 위치의 값들을 무작위로 섞는다.
- **GreedyOrdering** : 2개의 위치를 무작위로 선택하고 그 사이의 값들을 재정렬한다.

본 연구에서는 이를 확장하여 그리디 휴리스틱에 기반한 4가지 이웃해 생성 방법을 제시한다.

III. Greedy-based Neighbor Generation Methods

본 논문에서 제안하는 그리디 기반 이웃해 생성 방법은 총 4가지로 GreedyBlockWholeInsertion, GreedyBlock EachInsertion, GreedyBlockRotation, GreedyRandom Insertion이다. 각 이웃해 생성 알고리즘은 First-choice 언덕 오르기 탐색이나 시뮬레이티드 어닐링 내에서 하나의 이웃해를 생성하기 위해 호출된다.

1. GreedyBlockWholeInsertion (GBWI)

GreedyBlockWholeInsertion 알고리즘은 Fig. 2와 같다. 먼저 현재 경로 상의 무작위 위치 2개를 선택하는데(3라인), 그 사이에 있는 경로를 block이라 한다(4라인). 다음으로 현재해 next로부터 block을 제거하고 next를 서브투어(subtour)로 만든다(5라인). 이제 현재 next 서브투어의 경로 상에 block이 들어갈 수 있는 모든 엣지(edge)들을 고려하여 block이 삽입되었을 경우 거리가 가장 적게 증가하는 엣지를 찾는다(6라인). 단, 현재해에 block이 위치하던 엣지는 제외한다. 최종적으로 해당 엣지로 block을 삽입함으로써 이웃해를 완성한다(7라인).

```
1: Algorithm GreedyBlockWholeInsertion(current)
       current : Current solution
2: next ← current
3: i1, i2 ← Select 2 random positions
4: block ← Get the path from next[i1] to next[i2]
5: next ← Make subtour by removing block
       from next
6: edge ← Find an edge in next except for the
       current edge such that the increased length
       by inserting block between the edge's cities
       is minimal
7: next ← Insert block to edge in next
8: return next
```

Fig. 2. GreedyBlockWholeInsertion Algorithm

Fig. 3은 7개의 도시로 이루어진 순회 외판원 문제를 대상으로 GreedyBlockWholeInsertion 알고리즘의 동작 예를 나타낸 것이다. Fig. 3 (a)와 같이 현재해는 [0, 1, 2, 5, 6, 3, 4]이다. 네 번째 위치의 5번 도시와 여섯 번째 위치의 3번 도시가 선택되어 block은 (b)와 같이 [5, 6, 3]이 되고, 이를 제외한 서브투어는 (c)가 된다. 이제 (c)의 서브투어에서 block이 삽입될 수 있는 모든 엣지들 중 원래 위치인 e3을 제외하고 e1, e2, e4 중 가장 짧은 경로가 만들어지는 엣지로 block을 삽입하게 된다. 예를 들어, 가장 짧은 경로가 만들어지는 엣지가 e2라면 최종적으로 e2로 삽입되어 (d)와 같은 이웃해 하나가 완성된다.

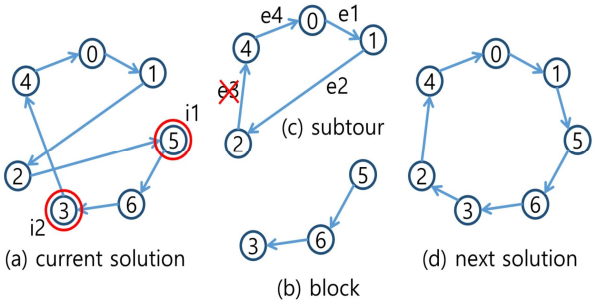

Fig. 3. An Example of GreedyBlockWholeInsertion

2. GreedyBlockEachInsertion (GBEI)

GreedyBlockEachInsertion 알고리즘은 Fig. 4와 같이 GreedyBlockWholeInsertion과 매우 유사하다. 차이점은 다음 두 가지이다. 첫 번째는 선택된 block에 포함되어 있는 경로를 무작위로 섞는다는 것이다(5라인). 두 번째는 block에 포함되어 있는 각 도시에 대해 하나씩 순서대로 현재 서브투어의 경로 상에 가장 좋은 엣지를 찾아 삽입한다는 것이다(7~9라인).

```
1: Algorithm GreedyBlockEachInsertion(current)
2: next ← current
3: i1, i2 ← Select 2 random positions
4: block ← Get the path from next[i1] to next[i2]
5: block ← Shuffle the cities in block
6: next ← Make subtour by removing block
       from next
7: for city in block
8:     edge ← Find an edge in next such that the
           increased length by inserting city
           between the edge's cities is minimal
9:     next ← Insert city to edge in next
10: return next
```

Fig. 4. GreedyBlockEachInsertion Algorithm

3. GreedyBlockRotation (GBR)

GreedyBlockRotation 알고리즘은 Fig. 5와 같다. 임의의 두 위치를 기준으로 block을 선택하는 것까지는 앞서 설명한 2가지 알고리즘과 동일하다(3~4라인). 이후로 block 내에서 원형으로 가능한 모든 회전을 시도하여 만든 이웃해들 중 가장 좋은 이웃해를 선택한다(5라인). 예를

들어, block 내에 5개의 도시가 존재한다면 1-left shift, 2-left shift, 3-left shift, 4-left shift를 통해 만든 이웃해들 중 가장 짧은 경로를 최종 이웃해로 결정한다.

```
1: Algorithm GreedyBlockRotation(current)
2: next ← current
3: i1, i2 ← Select 2 random position
4: block ← Get the path from next[i1] to next[i2]
5: k ← Find a k shift left circularly in block
        from 1 to |i2 - i1| such that the increased
        length by shifting block is minimal
6: next ← Shift left by k circularly in block in next
7: return next
```

Fig. 5. GreedyBlockRotation Algorithm

4. GreedyRandomInsertion (GRI)

GreedyRandomInsertion 알고리즘은 Fig. 6과 같이 GreedyBlockEachInsertion과 유사하다. 차이점은 모든 도시들 중 무작위로 k개를 선택한다는 것이다(3~4라인). 1개부터 (n-1)개까지의 위치가 무작위로 선택되어 해당 위치에 있는 도시들이 s_cities에 포함된다. 이후로는 GreedyBlockEachInsertion과 동일하다. s_cities에 포함된 도시들은 무작위로 재정렬되며(5라인), s_cities에 포함된 도시들을 제외하고 서브투어가 만들어진다(6라인). 이제 s_cities에 포함되어 있는 각 도시는 하나씩 현재 서브투어의 경로 상에 가장 좋은 엣지로 삽입된다(7~9라인).

```
1: Algorithm GreedyRandomInsertion(current)
2: next ← current
3: s_indexes ← Select k random positions
       (1≤k<total_city_count)
4: s_cities ← Get cities in next[i] for i in s_indexes
5: s_cities ← Shuffle the cities in s_cities
6: next ← Make subtour by removing cities in
       s_cities from next
7: for city in s_cities
8:       edge ← Find an edge in next such that the
             increased length by inserting city
             between the edge's cities is minimal
9:       next ← Insert city to edge in next
10: return next
```

Fig. 6. GreedyRandomInsertion Algorithm

IV. Experimental Results

1. Experimental Environment

본 연구의 실험 데이터로는 TSPLIB에서 제공하는 순회외판원 문제 데이터들 중 Table 1과 같이 10개의 데이터를 사용하였다[13]. "# of Cities"와 "Optimal Value" 칼럼에는 각 데이터의 도시 개수와 최적값이 기술되어 있다.

Table 1. Experimental Data

Data	# of Cities	Optimal
brazil58.tsp	58	25395
eil76.tsp	76	538
rat99.tsp	99	1211
pr136.tsp	136	96772
kroA150.tsp	150	26524
u159.tsp	159	42080
kroB200.tsp	200	29437
pr264.tsp	264	49135
pr299.tsp	299	48191
lin318.tsp	318	42029

시뮬레이티드 어닐링은 초기 온도 Tstart와 최종 온도 Tmin 그리고 온도 감소율 α 값에 따라 실험 결과에 큰 차이가 있을 수 있다. 따라서 본 연구에서는 Tstart 값으로는 100과 1000을 적용한 경우와 α 값으로는 0.99999와 0.999999를 사용한 경우를 고려하여 총 4가지 조합에 대해 각 이웃해 생성 방법별로 5회 실험 후 평균적으로 가장 좋은 파라미터 값을 결정하였다. Table 2는 이와 같이 결정된 최종 Tstart와 α를 나타낸 것이다. 이후 시뮬레이티드 어닐링의 실험 결과는 모두 각 이웃해 생성 방법별로 Table 2의 파라미터 값을 사용한 결과를 의미한다. 참고로 Tmin은 모두 1.0을 사용하였다.

Table 2. Parameter Settings for SA

Neighboring Method	Tstart	α
GreedyBlockWholeInsertion (GBWI)	1000	0.999999
GreedyBlockEachInsertion (GBEI)	100	0.999999
GreedyBlockRotation (GBR)	100	0.999999
GreedyRandomInsertion (GRI)	1000	0.99999

모든 실험은 Intel Core I7-6700K CPU 4.0GHz, 8GB RAM 및 Windows 10 64비트 운영체제 PC에서 수행되었으며, 프로그램은 Python 3.10을 사용하여 개발되었다.

2. Comparison of Random-based Methods with Greedy-based Methods

Table 3은 First-choice 언덕 오르기 탐색(FCHC)의 실험 결과이며, Table 4는 시뮬레이티드 어닐링(SA)의 실험 결과이다. 각 수치는 각 데이터 및 이웃해 생성 방법별로 총 5회의 실험을 실시한 결과의 평균값이며, SA를 위한 파리미

터는 Table 2를 따른다. 1회 실험 시 최대 수행 시간은 1800초(30분)로 제한하였는데, 모든 데이터에 있어서 충분히 수렴이 가능한 시간으로 판단하였다. 무작위 기반(Random-based) 방법들과 그리디 기반(Greedy- based) 방법들 중 GreedyOrdering은 기존 연구 [8]의 결과로서 개발 환경 및 수행 시간 등 모든 실험 조건은 본 연구의 실험 조건과 동일하다. 각 데이터별로 모든 이웃해 생성 방법들 중 가장 좋은 결과에 대해서는 빨간색 볼드체 및 이탤릭체로 표시하였다. Fig. 7은 각 이웃해 생성 방법별로 모든 데이터에 대한 평균값(Average)을 그래프로 나타낸 것이다.

Table 3 및 Fig. 7에 의하면 FCHC에 있어서 가장 좋은 결과를 보인 이웃해 생성 방법은 GRI이며, 다음으로는 GBEI, GBWI, GBR, BlockInsertion, Rotation, GreedyOrdering, Inversion 순으로 나타났다. 본 연구에서 제시한 4가지 그리디 기반 이웃해 생성 방법은 FCHC에 있어서 기존의 무작위 기반 이웃해 생성 방법 및 GreedyOrdering보다 성능이 뛰어남을 알 수 있다. 특히 GRI는 모든 데이터에 있어서 가장 좋은 결과를 보이는 등 매우 우수한 성능을 보였다.

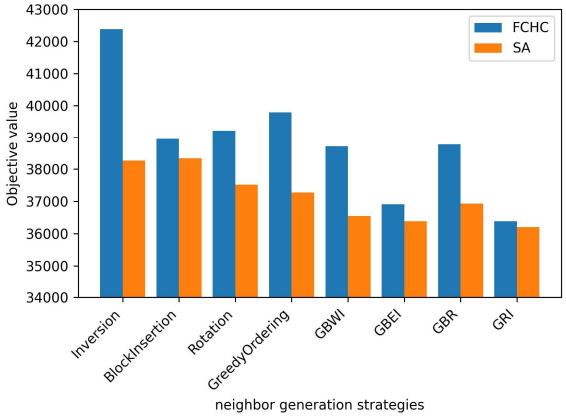

Fig. 7. Average Objective Value Comparison Graph of Neighbor Generation Methods

Table 4 및 Fig. 7에 의하면 SA의 경우 가장 좋은 이웃해 생성 방법은 GRI이고 다음으로는 GBEI, GBWI, GBR 순으로 FCHC와 동일하며, 이후로는 GreedyOrdering, Rotation, Inversion, BlockInsertion 순으로 나타났다. 본 논문에서 제시한 4가지 그리디 기반 이웃해 생성 방법 등 그리디 기반 이웃해 생성 방법들이 무작위 기반 이웃해 생성 방법들보다 우수함을 알 수 있다. Table 3, 4 및 Fig. 7에 의하면 GRI의 경우 FCHC의 결과조차 GRI와 GBEI를 제외한 다른 이웃해 생성 방법들에 의한 SA 결과보다 더 우수한 것으로 나타났다. 이를 통해 그리디 기반 생성 방법들 중 특히 GRI의 성능이 매우 우수함을 알 수 있다.

3. Comparison with Other Methods

Table 5는 본 연구에서 제시한 이웃해 생성 방법들 중 GRI와 GBEI를 기존의 지역 탐색 기반 연구들인 [8], [7] 및 [6]과 비교한 것이다. [8]의 경우 여러 가지 이웃해 생성 방법들 중 가장 좋은 성능을 발휘한 Combined의 결과를 나타낸 것으로 Combined는 이웃해 생성 방법들 중 Swap, Inversion, EdgeInsertion, BlockInsertion, GreedyOrdering을 무작위로 선택하여 적용하는 방법이다. 각 연구 별로 가장 좋은 해(Best)와 평균값(Average)을 기술하였다. 각 데이터별로 평균값들 중 가장 좋은 값들을 빨간색 볼드체 및 이탤릭체로 표시하였고, Best 해들 중 최적해에 대해서는 파란색 볼드체로 표시하였다.

Table 5의 평균값에 의하면 pr264.tsp를 제외한 모든 데이터 및 전체 평균에 있어서 GRI의 성능이 가장 우수함을 알 수 있으며, GBEI 또한 기존 연구 결과보다 우수함을 알 수 있다. 이를 통해 기본적인 지역 탐색 알고리즘의 틀 내에서 우수한 그리디 기반 이웃해 생성 방법을 개발하여 적용함으로써 전체적인 탐색 성능을 크게 향상시킬 수 있음을 확인할 수 있다.

V. Conclusions and Future Works

본 논문에서는 순회 외판원 문제를 위한 지역 탐색에 있어서 그리디 기반 이웃해 생성 방법들을 제시하였다. 실험을 통해 본 논문에서 제시한 그리디 기반 이웃해 생성 방법들이 기존의 무작위 기반 이웃해 생성 방법들보다 훨씬 효과적임을 확인하였다. 이뿐만 아니라 기본적인 시뮬레이티드 어닐링의 틀 내에서 그리디 기반 이웃해 생성 방법의 적용만으로도 기존의 지역 탐색 기법들보다 우수한 성능을 발휘함을 확인하였다.

본 연구에서는 그리디 이웃해 생성 방법의 도입을 통해 기본적인 지역 탐색 알고리즘만으로도 지역 탐색의 성능을 향상시킬 수 있었다. 따라서 향후 그리디 이웃해 생성 방법을 우수한 온도 스케쥴링 방법 등 기존의 우수한 지역 탐색 전략들과 접목함으로써 지역 탐색의 성능을 더욱 향상시킬 수 있을 것으로 예상된다. 한편 본 연구를 통해 개발된 그리디 이웃해 생성 방법들은 순회 외판원 문제와 같은 순열 기반 조합 최적화 문제에 적합하다는 한계가 있다. 따라서 순열 기반 조합 최적화 문제뿐만 아니라 집합 커버링 문제, 배낭 문제 등 보다 다양한 조합 최적화 문제로의 적용을 위한 지속적인 연구가 필요하다.

Table 3. Experimental Results for FCHC

Data	Random-based [8]			Greedy-based				
	Inversion	Block Insertion	Rotation	Greedy Ordering [8]	GBWI	GBEI	GBR	GRI
brazil58.tsp	27528.2	26240.4	26387.8	26904.0	26642.8	25567.6	26194.0	*25395.0*
eil76.tsp	606.8	565.6	565.6	573.0	565.8	552.4	570.8	*538.0*
rat99.tsp	1418.8	1299.4	1292.8	1350.6	1310.0	1238.8	1291.4	*1211.0*
pr136.tsp	113929.4	102896.4	104506.8	102565.2	102798.0	98359.2	103856.8	*96814.6*
kroA150.tsp	30766.8	28432.8	28976.8	28102.8	27924.0	26953.6	28190.2	*26543.6*
u159.tsp	49729.2	45071.4	45515.0	47660.2	45956.2	44257.8	45754.6	*42206.4*
kroB200.tsp	34621.0	31407.4	31869.4	32849.4	31496.8	30255.4	31841.6	*29823.0*
pr264.tsp	57762.6	54707.8	54831.8	54744.4	53091.6	*49365.4*	53314.6	49472.4
pr299.tsp	57631.8	52832.6	52676.6	55697.4	52277.4	49207.0	51489.0	*48631.4*
lin318.tsp	49885.6	45993.6	45533.6	47462.2	45042.8	43336.2	45148.8	*43236.6*
Average	42388.0	38944.7	39215.6	39790.9	38710.5	36909.3	38765.2	*36387.2*

Table 4. Experimental Results for SA

Data	Random-based [8]			Greedy-based				
	Inversion	Block Insertion	Rotation	Greedy Ordering [8]	GBWI	GBEI	GBR	GRI
brazil58.tsp	25725.0	25415.0	25435.0	*25395.0*	25415.0	*25395.0*	25397.2	*25395.0*
eil76.tsp	538.8	538.4	541.4	545.8	*538.0*	546.6	*538.0*	*538.0*
rat99.tsp	1258.0	1269.0	1232.8	1223.6	1220.0	1224.2	1215.4	*1211.0*
pr136.tsp	100862.6	100213.2	99642.8	97089.6	97322.0	96828.6	99870.0	*96772.0*
kroA150.tsp	27562.8	28013.4	27479.0	27150.4	26775.4	26674.8	26791.8	*26524.4*
u159.tsp	44331.4	44802.6	43151.4	42794.2	42575.4	42376.2	43108.0	*42080.0*
kroB200.tsp	31301.4	31205.0	30854.2	31006.4	29998.4	29622.0	30033.2	*29451.2*
pr264.tsp	53334.0	53367.6	51368.0	49909.6	49715.6	49168.8	50002.4	*49144.0*
pr299.tsp	52307.6	52987.4	51014.8	51849.0	48986.6	48812.0	49051.2	*48405.2*
lin318.tsp	45383.8	45561.0	44428.8	45784.6	42899.8	43165.6	43262.0	*42526.0*
Average	38260.5	38337.3	37514.8	37274.8	36544.6	36381.4	36926.9	*36204.7*

Table 5. Comparison of Greedy-based Methods with Other Algorithms

Data	Optimal	Greedy-based				Combined(2021) [8]		HVNS(2018) [7]		RNN-SA(2021) [6]	
		GRI		GBEI							
		Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
brazil58.tsp	25395	**25395.0**	*25395.0*	**25395.0**	*25395.0*	**25395.0**	*25395.0*	25425.0	25592.7	**25395.0**	25440.4
eil76.tsp	538	**538.0**	*538.0*	545.0	546.6	**538.0**	*538.0*	545.4	552.6	544.4	549.2
rat99.tsp	1211	**1211.0**	*1211.0*	1219.0	1224.2	**1211.0**	1215.8	1240.4	1241.3	1219.2	1229.3
pr136.tsp	96772	**96772.0**	*96772.0*	96781.0	96828.6	97729.0	98518.8	97979.1	97985.8	96922.4	100335.2
kroA150.tsp	26524	**26524.0**	*26524.4*	**26524.0**	26674.8	26528.0	26687.0	26943.3	26947.2	26821.8	27008.2
u159.tsp	42080	**42080.0**	*42080.0*	**42080.0**	42376.2	42396.0	42546.0	42436.2	42467.6	42162.8	42547.7
kroB200.tsp	29437	29438.0	*29451.2*	29542.0	29622.0	29489.0	29666.4	30447.3	30453.2	29825.2	30033.0
pr264.tsp	49135	**49135.0**	49144.0	49143.0	49168.8	**49135.0**	*49135.0*	51155.4	51197.1	49197.3	49375.8
pr299.tsp	48191	48331.0	*48405.2*	48569.0	48812.0	48639.0	48822.8	50271.7	50373.1	48811.5	49003.9
lin318.tsp	42029	42443.0	*42526.0*	42815.0	43165.6	42457.0	42886.4	43924.1	43964.9	42862.5	43041.1
Average	36131.2	36186.7	*36204.7*	36261.3	36381.4	36351.7	36541.1	37036.8	37077.6	36376.2	36856.4

ACKNOWLEDGEMENT

This research was supported by Kumoh National Institute of Technology(2021).

REFERENCES

[1] S. P. Tiwari, S. Kumar, and K. K. Bansal, "A Survey of Metaheuristic Algorithms for Travelling Salesman Problem," International Journal Of Engineering Research & Management Technology, Vol. 1, No. 5, Sep. 2014.

[2] R. A. Palhares, and M. C. B. AraUjo, "Vehicle Routing: Application of Travelling Salesman Problem in a Dairy," in 2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 1421-1425, 2018.

[3] G. Wang, J. Wang, M. Li, H. Li, and Y. Yuan, "Robot Path Planning Based On The Travelling Salesman Problem," Chemical Engineering Transactions, Vol. 46, pp. 307-312, 2015. doi: 10.3303/CET1546052

[4] A. Roy, A. Manna, and S. Maity, "A novel memetic genetic algorithm for solving traveling salesman problem based on multi-parent crossover technique," Decision Making. Applications in Management and Engineering, Vol. 2, No. 2, pp. 100-111, 2019.

[5] W. Gao, "New Ant Colony Optimization Algorithm for the Traveling Salesman Problem," International Journal of Computational Intelligence Systems, Vol. 13, No. 1, pp. 44-55, 2020.

[6] M. A. Rahman, and H. Parvez, "Repetitive Nearest Neighbor Based Simulated Annealing Search for Traveling Salesman Problem," Open Access Library Journal, Vol. 8, No. 6, e7520, June 2021.

[7] S. Hore, A. Chatterjee, and A. Dewanji, "Improving variable neighborhood search to solve the traveling salesman problem," Applied Soft Computing, Vol. 68, pp. 83-91, July 2018.

[8] J. Hwang, "Neighbor Generation Strategies of Local Search for Permutation-based Combinatorial Optimization," Journal of the Korea Society of Computer and Information, Vol. 26, No. 10, pp. 27-35, 2021. doi:10.9708/JKSCI.2021.26.10.027

[9] F. Glover, "Tabu Search: A Tutorial," Interfaces, Vol. 20, No. 4, pp. 74-94, 1990.

[10] R. A. Rutenbar, "Simulated annealing algorithms: An overview," IEEE Circuits and Devices magazine, Vol. 5, No. 1, pp. 19-26, 1989.

[11] D. Abramson, M. Krishnamoorthy, and H. Dang, "Simulated Annealing Cooling Schedules for the School Timetabling Problem," Asia Pacific Journal of Operational Research, Vol. 16, pp. 1-22, 1999.

[12] C. Nilsson, "Heuristics for the Traveling Salesman Problem," Technical Report, Linköping University, Sweden, 2003.

[13] TSPLIB, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Authors

Junha Hwang received the B.S., M.S. and Ph.D. degrees in Computer Engineering from Pusan National University, Korea, in 1995, 1997 and 2002, respectively.

Dr. Hwang joined the faculty of the Department of Computer Engineering at Kumoh National Institute of Technology, Gumi, Korea, in 2002. He is currently a Professor in the Department of Computer Engineering, Kumoh National Institute of Technology. He is interested in AI, combinatorial optimization, and machine learning.

Yongho Kim received the B.S and M.S degrees in Computer Engineering from Kumoh National Institute of Technology, Gumi, Korea in 2001, 2003, respectively.

Kim also received the Ph.D. Candidate degree in Computer Engineering from Kumoh National Institute of Technology in 2006 and is currently a lecturer in the Department of Computer Engineering, Kumoh National Institute of Technology. He is interested in AI, combinatorial optimization.