JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 4, pp. 1-8, April 2024
https://doi.org/10.9708/jksci.2024.29.04.001

A Combined Greedy Neighbor Generation Method of Local Search for the Traveling Salesman Problem

Yongho Kim*, Junha Hwang*

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

[Abstract]

The traveling salesman problem(TSP) is one of the well known combinatorial optimization problems. Local search has been used as a method to solve TSP. Greedy Random Insertion(GRI) is known as an effective neighbor generation method for local search. GRI selects some cities from the current solution randomly and inserts them one by one into the best position of the current partial solution considering only one city at a time. We first propose another greedy neighbor generation method which is named Full Greedy Insertion(FGI). FGI determines insertion location one by one like GRI, but considers all remaining cities at once. And then we propose a method to combine GRI with FGI, in which GRI or FGI is randomly selected and executed at each iteration in simulated annealing. According to the experimental results, FGI alone does not necessarily perform very well. However, we confirmed that the combined method outperforms the existing local search methods including GRI.

▸ **Key words**: Greedy neighbor generation, Combined method, Local search, Traveling salesman problem, Simulated annealing

[요 약]

순회 외판원 문제(TSP)는 잘 알려진 조합 최적화 문제 중 하나이다. 지역 탐색은 TSP를 해결하기 위한 한 가지 방법으로 사용되어 왔다. Greedy Random Insertion(GRI)은 지역 탐색을 위한 효과적인 이웃해 생성 방법으로 알려져 있다. GRI는 현재해로부터 일부 도시들을 무작위로 선택하고 그 도시들을 한 번에 하나의 도시만 고려하여 현재 부분해의 최적 위치로 삽입한다. 본 논문에서는 먼저 Full Greedy Insertion(FGI)이라는 또 다른 그리디 이웃해 생성 방법을 제안한다. FGI는 GRI와 마찬가지로 삽입 위치를 하나씩 결정하되 남은 모든 도시들을 한꺼번에 고려하여 결정한다. 그리고 본 논문에서는 GRI와 FGI를 결합하는 방법을 제시한다. 결합 방법에서는 시뮬레이티드 어닐링 내에서 매 반복 시 GRI 또는 FGI를 무작위로 선택하여 실행한다. 실험 결과에 의하면, FGI 단독으로는 성능이 매우 우수한 것은 아니다. 그러나 결합 방법은 GRI를 포함한 기존의 지역 탐색 방법들보다 우수한 성능을 발휘함을 확인하였다.

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

• First Author: Yongho Kim, Corresponding Author: Junha Hwang
*Yongho Kim (dokebi@kumoh.ac.kr), Dept. of Computer Engineering, Kumoh National Institute of Technology
*Junha Hwang (jhhwang@kumoh.ac.kr), Dept. of Computer Engineering, Kumoh National Institute of Technology
• Received: 2024. 02. 19, Revised: 2024. 03. 25, Accepted: 2024. 03. 27.

Copyright © 2024 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

순회 외판원 문제는 조합 최적화 문제 중 하나로 시작 도시에서 출발하여 모든 도시들을 한 번씩 방문한 뒤 다시 시작 도시로 되돌아오되 가장 짧은 경로를 찾는 문제이다[1]. 순회 외판원 문제는 NP-hard(Non-deterministic Polynomial-time hard) 문제로 알려져 있어 도시의 수가 증가함에 따라 전역 최적해(global optimum)를 찾는 것이 매우 어려워진다. 따라서 전역 최적해가 아니더라도 준최적해를 찾기 위한 탐색 기법을 사용하는 것이 일반적이다.

지역 탐색(local search)은 순회 외판원 문제와 같은 최적화 문제를 해결하기 위한 반복적 개선 탐색 방법 중 하나이다. 기본적인 지역 탐색은 현재해로부터 이웃해를 만들고 이 이웃해가 현재해보다 좋다면 이를 현재해로 선택하는 과정을 반복적으로 수행한다[2]. 그러나 이 방법은 지역 최적해(local optimum)를 쉽게 찾을 수 있지만, 이는 오히려 지역 최적해에 갇혀 전역 최적해를 찾지 못하는 단점으로 작용한다. 이러한 단점을 보완하기 위해 시뮬레이티드 어닐링(simulated annealing)과 같은 다양한 지역 탐색 기법들이 제시되어 왔다.

시뮬레이티드 어닐링은 기본적인 지역 탐색인 언덕 오르기 탐색을 기반으로 한다. 즉, 이웃해가 현재해보다 더 좋다면 이웃해로 이동하게 된다. 다만 이웃해가 현재해보다 좋지 않더라도 확률적으로 이동할 수 있다[3]. 시뮬레이티드 어닐링은 이를 통해 지역 최적해를 벗어나 새로운 영역으로 이동함으로써 전역 최적해를 찾기 위한 보다 광범위한 탐색을 시도한다. 시뮬레이티드 어닐링의 성능에 가장 큰 영향을 미치는 요소로는 이웃해 생성 방법과 이동 확률을 결정하는 온도 스케쥴링(scheduling)이 있다[4, 5].

본 논문에서는 순회 외판원 문제를 해결하기 위한 시뮬레이티드 어닐링의 적용 방안으로 그리디(greedy) 기반의 새로운 이웃해 생성 방법 두 가지를 제안한다. 첫 번째는 Full Greedy Insertion(FGI)으로 현재해로부터 임의 개수의 도시를 선택하고 각 도시의 삽입 위치를 결정하되 아직 남아있는 모든 도시들을 대상으로 가장 좋은 위치를 결정하게 된다. 두 번째는 기존 연구 [6]에서 제안된 Greedy Random Insertion(GRI)과 FGI를 결합한 방법이다. 여기서는 하나의 이웃해 생성 시 GRI와 FGI 중 하나를 무작위로 선택하고 해당 방법에 따라 이웃해를 생성한다. 비록 FGI가 단독으로는 매우 좋은 성능을 발휘하지 못할 수도 있지만 GRI와의 결합을 통해 전체 성능을 향상시킬 수 있는 중요한 역할을 담당할 것으로 기대한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 먼저 지역 탐색과 시뮬레이티드 어닐링에 대해 소개하고, 순회 외판원 문제를 위한 GRI 등 대표적인 이웃해 생성 기법들을 설명한다. 3장에서는 본 논문에서 제안하는 그리디 기반의 이웃해 생성 방법들에 대해 기술한다. 4장에서는 실험 결과를 제시하고 분석하며, 마지막으로 5장에서 결론 및 향후 과제에 대해서 기술한다.

II. Related Works

1. Local Search and Simulated Annealing

지역 탐색은 하나의 해로부터 출발하여 이웃해로 이동하면서 더 좋은 해를 도출하는 탐색 기법이다[2]. 가장 단순한 지역 탐색 기법으로는 언덕 오르기 탐색이 있으나 전역 최적해를 찾지 못하고 지역 최적해에 머문다는 단점이 있다. 이를 개선하기 위해 Tabu 탐색, 시뮬레이티드 어닐링과 같은 다양한 지역 탐색 기법들이 등장하였다. Tabu 탐색은 생성 가능한 모든 이웃해들 중 가장 좋은 해로 이동하는 Steepest-ascent 언덕 오르기 탐색을 기반으로 한다. 그러나 본 논문에서 제안하는 이웃해 생성 방법의 경우 이웃해의 개수가 매우 많기 때문에 Tabu 탐색과 같은 이웃해 탐색 기법의 적용이 어렵다. 반면 시뮬레이티드 어닐링은 이웃해 하나를 생성한 후 이동 여부를 결정하는 First-choice 언덕 오르기 탐색을 기반으로 하기 때문에 본 논문에서 제안하는 이웃해 생성 방법의 적용이 보다 자연스럽다.

본 논문에서는 순회 외판원 문제 해결을 위한 기본 알고리즘으로 시뮬레이티드 어닐링을 사용한다. 최소화 문제를 대상으로 한 시뮬레이티드 어닐링 알고리즘은 Fig. 1과 같다[6].

먼저 초기해(current)를 만들고 시작 온도를 설정한 후(2~3라인) 중단 조건이 될 때까지 다음 내용들을 반복 수행한다(4~12라인). 현재해로부터 이웃해(next)를 생성하고 두 해를 비교한다. 만약 이웃해가 현재해보다 더 좋으면 이웃해를 현재해로 선택하고, 그렇지 않으면 확률적으로 선택 여부를 결정한다. 이 확률은 현재해와 이웃해의 차이인 ΔE와 현재 온도 T에 의해 결정되며, 온도 T는 온도 스케쥴링에 따라 감소된다. 온도 T는 미리 설정한 기준 아래로 감소하지 않도록 설정할 수 있다.

```
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 성능은 이웃해 생성 방법과 온도 스케쥴링 방법에 큰 영향을 받는다. 이웃해 생성 방법이 우수할수록 현재해보다 더 좋은 해의 도출 가능성이 높아지며, 온도를 적절히 낮추어야 현재해의 활용뿐만 아니라 새로운 탐색 공간으로의 이동이 원활해진다[7]. 본 논문에서는 새로운 이웃해 생성 방법을 제안하고 있다. 따라서 온도 스케쥴링 방법으로는 기존 연구 [6]과 마찬가지로 식 1과 같이 가장 기본적인 방법인 Geometric 방식을 사용한다. 즉, 현재 온도에 온도 변화율 계수 α(<1.0)를 곱해 다음 온도를 결정한다.

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

2. Greedy Neighbor Generation

순회 외판원 문제를 해결한다는 것은 다음 두 가지 중 하나를 의미한다. 첫 번째는 경로 생성이고 두 번째는 경로 개선이다.

경로 생성은 Fig. 1의 2라인에서 초기해를 만드는 과정을 의미한다. 이때 무작위 순서로 경로를 생성할 수도 있지만 보다 짧은 경로를 생성하기 위해 특별한 알고리즘을 적용할 수도 있다. 그러나 어떤 방법을 사용하더라도 초기해 생성만으로 만족할 만한 경로를 생성하는 것은 매우 어려운 일이다. 따라서 지역 탐색에서와 같이 경로를 개선하는 과정이 필수적이다. 본 연구에서는 초기해를 무작위로 생성한 후 시뮬레이티드 어닐링을 사용하여 경로를 개선해나간다.

경로 개선은 Fig. 1의 4~12라인과 같이 현재해를 기반으로 이웃해를 생성하고 더 좋은 해로의 이동을 반복 수행하는 과정이다. 이때 이웃해 생성 방법은 지역 탐색의 성능에 큰 영향을 미치게 된다[6, 8, 9]. 기존 연구에서 제안된 이웃해 생성 방법은 크게 무작위 생성 방법과 그리디 생성 방법으로 구분할 수 있다.

무작위 생성 방법으로는 Swap, Inversion, Edge Insertion 등이 있다[4]. Swap은 현재해의 경로상에서 2개 도시를 무작위로 선택한 후 서로 교환하며, Inversion은 선택된 2개 도시 사이의 값들을 모두 역전시킨다. Edge Insertion은 1개 도시를 무작위로 선택한 후 임의의 위치로 삽입한다. 그리디 생성 방법으로는 Greedy Block Each Insertion, Greedy Random Insertion 등이 있다[6]. 그리디 생성 방법은 현재해의 경로상에서 무작위로 선택한 도시들을 경로 길이 증가가 최소화되는 위치를 찾아 삽입하는 방식을 사용한다. 그리디 생성 방법의 경우 도시를 재배치하는 과정에서 경로의 거리 증가를 미리 평가해 본다는 특징이 있다.

기존 연구 [6]에서는 Greedy Random Insertion(GRI) 이웃해 생성 방법이 가장 좋은 성능을 발휘함을 확인하였다. GRI 알고리즘은 Fig. 2와 같다. 먼저 현재해로부터 k개 도시를 무작위로 선택하고 섞는다(3~5라인). 그리고 선택되지 않은 도시들만으로 부분 경로를 만든다(6라인). 다음으로는 무작위로 선택된 도시들을 순서대로 하나씩 부분 경로에 삽입하되 거리가 가장 적게 증가하는 도시 사이에 삽입한다(7~9라인). k개의 도시가 모두 삽입되면 하나의 이웃해가 완성된다.

```
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. 2. Greedy Random Insertion (GRI) Algorithm

본 논문에서는 Full Greedy Insertion(FGI)이라는 새로운 그리디 이웃해 생성 방법을 제안하며, 아울러 기존의 GRI와 FGI를 결합하여 적용함으로써 시뮬레이티드 어닐링의 성능을 향상시키는 방안을 제시한다.

III. Combined Greedy Neighbor Generation Method

본 장에서는 먼저 새로운 그리디 이웃해 생성 방법인 Full Greedy Insertion에 대해 설명하며, 이어서 Greedy Random Insertion과 Full Greedy Insertion을 결합하여 적용하는 방법에 대해 설명한다.

1. Full Greedy Insertion(FGI)

Full Greedy Insertion(FGI) 알고리즘은 Fig. 3과 같다. FGI는 기본적으로 GRI와 매우 유사하다. GRI와 마찬가지로 현재해로부터 k개 도시들을 무작위로 선택하고 나머지 부분 경로를 대상으로 k개의 도시를 전체 거리가 가장 적게 증가하는 edge에 하나씩 삽입한다. 다만 다음 두 가지 측면에서 차이가 있다.

첫 번째 차이점은 이웃해 생성을 위해 현재해로부터 선택되는 도시의 개수이다(3라인). GRI는 n개의 도시들 중 최대 $(n-1)$개 도시가 선택될 수 있지만, FGI는 최대 개수를 m개로 제한한다. 물론 m 값을 $(n-1)$로 설정할 수도 있지만 m 값을 크게 설정할 경우 한 번의 이웃해 이동을 위해 필요한 수행 시간이 과도하게 클 수 있기 때문에 비교적 작은 값으로 설정하는 것이 바람직하다. m의 값에 따른 수행 시간 증가는 바로 이어서 설명할 GRI와 FGI의 두 번째 차이점으로부터 기인한다.

```
1: Algorithm FullGreedyInsertion(current)
2: next ← current
3: s_indexes ← Select k random positions (1≤k≤m)
4: s_cities ← Get cities in next[i] for i in s_indexes
5: next ← Make subtour by removing cities in
       s_cities from next
6: while s_cities is not empty
7:     city, edge ← Find a city in s_cities and an
           edge in next such that the increased
           length by inserting the city between the
           edge's cities is minimal
8:     next ← Insert city to edge in next
9:     remove city from s_cities
10: return next
```

Fig. 3. Full Greedy Insertion (FGI) Algorithm

두 번째 차이점은 다음으로 삽입할 도시와 edge를 선택하는 방식이다. GRI는 선택된 도시들을 임의 순서로 재정렬하여 하나씩 순서대로 삽입 위치를 결정한다. 즉, k개의 선택 도시 중 i번째 도시가 먼저 재배치된 후 $(i+1)$번째 도시가 재배치된다. 그러나 FGI는 현재 남아있는 모든 도시들을 대상으로 가장 좋은, 즉, 선택 도시의 순서에 관계없이 거리가 가장 적게 증가하는 도시와 edge를 결정한다(7~9라인). GRI와 FGI에서 거리 증가가 가장 적은 첫 번째 (도시, edge)의 조합을 찾기 위한 평가 수를 비교해 보면, 선택 도시가 k개일 때 삽입 대상 edge가 $(n-k)$개이므로 GRI는 총 $\{1 \times (n-k)\}$회의 평가가 필요하고 FGI는 $\{k \times (n-k)\}$회의 평가가 필요하다. 이와 같이 FGI는 선택 도시 수가 많아짐에 따라 하나의 이웃해를 생성하기 위한 소요 시간이 대폭 증가한다는 단점이 있다.

Fig. 4는 총 7개 도시로 구성된 순회 외판원 문제에서 GRI와 FGI에 의한 이웃해 생성 과정을 그림으로 나타낸 것이다. 현재해는 (a)와 같이 1-3-4-2-6-7-5이며 총 거리는 31.8이다. 이때 이웃해 하나를 생성하기 위해 4번과 7번 도시가 선택되었다고 가정하자. 그러면 이 도시들을 제외한 서브투어는 (b)와 같이 1-3-2-6-5로 구성되며 총거리는 23.7이 된다. Fig. 4에서 (s:)는 서브투어에 삽입할 도시들을 의미한다. 이후 (c.1)과 (c.2)는 GRI의 실행 단계를 나타낸 것이며, (d.1)과 (d.2)는 FGI의 실행 단계를 나타낸 것이다.

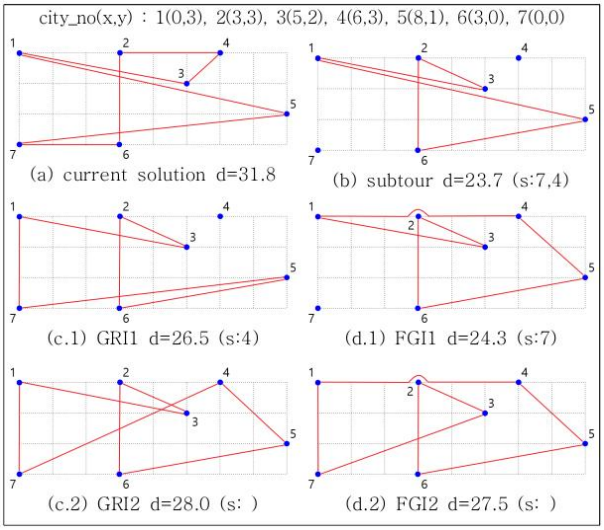

Fig. 4. An Example of GRI and FGI

GRI의 경우 7번, 4번 도시 순으로 삽입하는 것으로 결정되었다고 가정하자. 먼저 7번 도시는 (c.1)과 같이 거리 증가가 가장 적은 1번과 5번 도시 사이에 삽입되며, 이때 총거리는 2.8이 증가되어 26.5가 된다. 반면에 FGI는 7번과 4번 도시 모두를 대상으로 거리 증가가 가장 적은 edge를 선택한다. 그 결과 (d.1)과 같이 4번 도시를 1번과 5번 도시 사이에 삽입하며, 총거리는 24.3이 된다. 다음으로 GRI와 FGI는 각각 남은 1개 도시를 삽입한다. 최종적으로 GRI는 (c.2)와 같이 4번 도시를 5번과 7번 도시 사이

에 삽입하여 총거리는 28.0이 되며, FGI는 (d.2)와 같이 7번 도시를 1번과 3번 도시 사이에 삽입하여 총거리는 27.5가 된다. 이와 같이 때로는 FGI를 통해 GRI보다 더 나은 이웃해를 만들 수 있을 것으로 기대된다.

2. Combined Method(CombGRI&FGI)

본 논문에서 제안하는 GRI와 FGI의 결합 방법(CombGRI&FGI)은 Fig. 5와 같이 매우 단순하다. 2라인의 randint 함수는 0 또는 1의 정수 중 하나를 무작위로 반환하는 함수이다. 즉, 현재해에 대한 이웃해 생성 시 GRI와 FGI 중 하나를 무작위로 선택하고 이를 적용하여 이웃해를 생성한다.

```
1: Algorithm CombGRI&FGI(current)
2: method = randint(0, 1)
3: if method = 0
4:       next ← GreedyRandomInsertion(current)
5: else
6:       next ← FullGreedyInsertion(current)
7: return next
```

Fig. 5. CombGRI&FGI Algorithm

최적화 문제를 해결하기 위한 탐색 과정에서 탐험(exploration)과 활용(exploitation)의 균형을 유지하는 것이 매우 중요하다[10]. 탐험을 위해서는 무작위적인 요소의 도입을 통한 다양한 영역으로의 이동이 필요하며, 활용을 위해서는 현재해의 정보를 잘 활용할 필요가 있다. GRI나 FGI와 같은 그리디 이웃해 생성 방법은 기본적으로 현재해보다 더 좋은 이웃해의 생성을 기대하는 방법으로 활용 측면을 강조하고 있다. 그러나 활용을 강조하다 보면 지역 최적해로부터 벗어나기 어렵다는 단점이 있다. 본 논문에서 제시하는 결합 방법인 CombGRI&FGI는 각각의 이웃해 생성 방법이 활용을 강조하면서도 상대방 생성 방법에게 탐험의 기회를 제공할 수 있을 것으로 판단된다.

IV. Experimental Results

1. Experimental Environment

본 연구의 실험 데이터로는 TSPLIB에서 제공하는 순회외판원 문제 데이터들 중 Table 1과 같이 10개의 데이터를 사용하였다[11]. 각 열은 순서대로 데이터 이름, 도시 수 그리고 알려진 최적값을 의미한다.

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

본 논문의 모든 실험은 Intel Core i7-6700K CPU, 8GB RAM, 윈도우 10 64비트의 PC 환경에서 수행하였으며, Python 3.10으로 프로그램을 구현하였다.

기존 연구 [6]에서 이웃해 생성 방법별로 시뮬레이티드 어닐링의 파라미터인 Tstart, Tmin, α 값에 따라 성능에 차이가 있음을 확인하였다. 따라서 본 연구에서는 [6]에서 비교적 좋은 성능을 보인 파라미터 값을 적용하여 실험을 수행하였다. 즉, Tstart 값으로는 100과 1000, α 값으로는 0.99999와 0.999999를 사용하였으며, Tmin은 1.0으로 고정하였다. 즉, Tstart 값과 α 값에 따라 4가지 실험 조합을 사용하였다. 거기에 FGI와 CombGRI&FGI의 파라미터인 선택 도시 수의 상한값 m의 값으로 3, 5, 10, 30, 50, (n-1)을 각각 적용하였다. n은 TSP 데이터의 전체 도시 수이다. 따라서 m이 (n-1)이라는 것은 1개를 제외한 모든 도시가 이웃해 생성을 위해 선택될 수 있음을 의미한다. 최종적으로 Tstart, α, m 값에 따라 총 24가지 실험 조합을 대상으로 실험을 수행하였다. 모든 파라미터 조합에 대한 실험 결과에 의하면 m 값에 관계없이 Full Greedy Insertion(FGI)은 Tstart가 1000이고 α 값이 0.999999일 때 성능이 가장 우수하였으며, 결합 방법인 CombGRI&FGI는 Tstart가 1000이고 α가 0.99999일 때 가장 우수하였다. 참고로 각 파라미터 조합 사이의 성능 비교는 각 파라미터별로 5회 실험을 수행한 후 그 결과에 대한 평균값을 바탕으로 하였으며, 각 실험당 수행 시간은 30분(1800초)으로 제한하였다. 따라서 이후 FGI와 CombGRI&FGI에 대한 실험 결과는 Table 2와 같은 Tstart와 α 값을 적용한 때의 결과를 제시하며, m의 경우 필요에 따라 해당 값들과 결과를 제시한다.

Table 2. Parameter Settings for SA

Neighboring Method	Tstart	α
Full Greedy Insertion(FGI)	1000	0.999999
Combined Method(CombGRI&FGI)	1000	0.99999

2. Comparison of FGI with CombGRI&FGI

Table 3은 FGI와 CombGRI&FGI에 대한 실험 결과이다. FGI는 선택도시 수 최대값인 m으로 3, 10, 30, (n-1)에 대한 실험 결과를 기술하였으며, CombGRI&FGI는 3, 10, 30, 50에 대한 실험 결과를 기술하였다. 해당 m 값들만으로도 각 이웃해 생성 기법의 특성을 충분히 파악할 수 있는 것으로 판단된다. Table 3의 각 값은 5회 실험에 대한 평균값을 의미한다. 파란색 볼드체는 해당 데이터의 알려진 최적값, 빨간색 볼드-이탤릭체는 모든 이웃해 생성 기법과 모든 m 값으로 실험한 결과 중 가장 좋은 값, 검은색 볼드-이탤릭체는 해당 이웃해 생성 기법 내에서 서로 다른 m 값으로 실험한 결과 중 가장 좋은 값을 의미한다.

FGI는 m 값이 (n-1)일 때 평균 36420.8로 가장 좋은 결과를 보였다. 가장 나쁜 결과는 m 값이 3일 때로 m 값이 클수록 결과가 더 좋음을 알 수 있다. 그리고 각 데이터별 결과를 보면 eil76.tsp 데이터를 제외한 나머지 9개 데이터에서 m 값이 (n-1)일 때 성능이 가장 좋음을 확인할 수 있었다. m 값이 클수록 최적의 삽입 장소를 찾기 위한 (도시, edge) 조합이 더 많아져서 한 번의 이웃해 생성을 위해 더 많은 시간이 소비되지만, 많은 조합 중에서 가장 좋은 것을 선택하는 데 따른 이득이 이를 상쇄할 만큼 더 큰 것으로 해석된다.

반면에 CombGRI&FGI의 경우 m 값이 10일 때의 성능이 평균 36175.9로 가장 좋았으며, m 값이 가장 큰 50일 때의 성능이 가장 좋지 않았다. 참고로 m 값이 (n-1)일 때의 평균은 36400.7로 m 값들 중 가장 좋지 않은 결과를 보였다. 이웃해 하나를 만들 때 FGI는 GRI보다 더 많은 시간을 필요로 한다. 특히 m 값이 매우 큰 경우 이웃해 하나를 만드는 데 소요되는 평균 시간은 GRI에 비해 매우 커지게 된다. 그러나 FGI를 통해 많은 시간을 사용하여 하나의 이웃해를 생성하는 것이 전체적으로는 크게 도움이 되지 않는 것으로 보인다. 오히려 m 값을 비교적 작게 설정하되 어느 정도 다양성을 보장하는 적절한 값으로 설정할 때 FGI의 수행 시간도 단축되고 GRI에게는 또 다른 방향으로의 탐색이 가능해져 전체적으로 더 좋은 해가 도출될 가능성이 커지는 것으로 분석된다. 마찬가지로 GRI 역시 FGI에게 다른 방향으로 탐색할 수 있는 기회를 제공한다고 볼 수 있다. 다만 본 연구에서는 CombGRI&FGI의 경우 Tstart가 1000, α가 0.99999, m이 10일 때 가장 좋은 결과를 보였지만, 주어진 문제의 상황에 따라 적절한 Tstart, α 및 m 값을 설정할 필요가 있다.

Table 3에서 FGI와 CombGRI&FGI를 비교해보면 m 값에 관계없이 모든 경우에 있어서 CombGRI&FGI가 FGI보다 우수한 성능을 발휘함을 확인할 수 있다.

3. Comparison with Other Methods

Table 3을 통해 CombGRI&FGI가 FGI보다 우수함을 확인하였지만, CombGRI&FGI의 우수성을 확인하기 위해서는 기존 연구인 GRI 이웃해 생성 방법 등과 비교하여 종합적으로 파악할 필요가 있다. Table 4는 본 연구에서 제시한 FGI와 CombGRI&FGI를 GRI 등 기존의 지역 탐색 기반 연구들인 [6], [8], [9]와 비교한 결과이다. 각 연구 결과별로 가장 좋은 해(Best)와 평균값(Average)을 기술하였다. 즉, FGI는 Tstart가 1000, α가 0.999999, m이 50일 때의 결과를 표시하였으며, CombGRI&FGI의 경우 Tstart가 1000, α가 0.99999, m이 10일 때의 결과를 표시하였다. 기존 연구인 GRI, RNN-SA, HVNS의 실험 결과는 본 연구와 동일한 환경에서 실험한 기존 연구 [6]을 참고하였다. 각 데이터별로 Best와 평균값들 중 알려진 최적값은 파란색 볼드체로 표시하였고, 이외에 가장 좋은 결과값에 대해서는 빨간색 볼드-이탤릭체로 표시하였다.

Table 4의 평균값에 의하면 CombGRI&FGI의 경우 10

Table 3. Experimental Results for FGI and CombGRI&FGI

Data	m / Optimal	FGI (Tstart=1000, α=0.999999)				CombGRI&FGI (Tstart=1000, α=0.99999)			
		3	10	30	*n-1*	3	10	30	50
brazil58.tsp	25395	26445.0	25476.8	**25395.0**	**25395.0**	**25395.0**	**25395.0**	**25395.0**	**25395.0**
eil76.tsp	538	560.8	540.2	**538.0**	546.0	**538.0**	**538.0**	**538.0**	540.6
rat99.tsp	1211	1318.4	1308.0	1215.0	***1214.0***	**1211.0**	**1211.0**	**1211.0**	1211.4
pr136.tsp	96772	114377.4	108517.8	101769.2	***96803.2***	**96772.0**	**96772.0**	**96772.0**	**96772.0**
kroA150.tsp	26524	30604.8	29104.6	27854.8	***26678.4***	**26524.0**	26524.2	**26524.0**	**26524.0**
u159.tsp	42080	51680.2	50227.8	44757.0	**42080.0**	**42080.0**	**42080.0**	**42080.0**	**42080.0**
kroB200.tsp	29437	35322.8	34201.2	32272.2	***29914.2***	29447.4	***29441.8***	***29441.8***	29489.4
pr264.tsp	49135	82508.4	69702.6	64040.2	***49196.0***	**49135.0**	**49135.0**	49138.2	49272.0
pr299.tsp	48191	62763.4	60636.0	57369.4	***48878.0***	48366.8	***48296.4***	48420.0	48515.2
lin318.tsp	42029	57762.0	53771.8	50334.6	***43502.8***	42429.2	***42365.2***	42587.0	42846.8
Average	36131.2	46334.3	43348.7	40554.5	***36420.8***	36189.8	***36175.9***	36210.7	36264.6

Table 4. Comparison of FGI and CombGRI&FGI with Other Algorithms

Data	Optimal	CombGRI&FGI		FGI		GRI(2022) [6]		RNN-SA(2021) [8]		HVNS(2018) [9]	
		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**	**25395.0**	25440.4	25425.0	25592.7
eil76.tsp	538	**538.0**	**538.0**	544.0	546.0	**538.0**	**538.0**	544.4	549.2	545.4	552.6
rat99.tsp	1211	**1211.0**	**1211.0**	1212.0	1214.0	**1211.0**	**1211.0**	1219.2	1229.3	1240.4	1241.3
pr136.tsp	96772	**96772.0**	**96772.0**	96772.0	96803.2	**96772.0**	**96772.0**	96922.4	100335.2	97979.1	97985.8
kroA150.tsp	26524	**26524.0**	***26524.2***	26600.0	26678.4	**26524.0**	26524.4	26821.8	27008.2	26943.3	26947.2
u159.tsp	42080	**42080.0**	**42080.0**	**42080.0**	**42080.0**	**42080.0**	**42080.0**	42162.8	42547.7	42436.2	42467.6
kroB200.tsp	29437	29439.0	***29441.8***	29732.0	29914.2	***29438.0***	29451.2	29825.2	30033.0	30447.3	30453.2
pr264.tsp	49135	**49135.0**	**49135.0**	49155.0	49196.0	**49135.0**	49144.0	49197.3	49375.8	51155.4	51197.1
pr299.tsp	48191	***48220.0***	***48296.4***	48480.0	48878.0	48331.0	48405.2	48811.5	49003.9	50271.7	50373.1
lin318.tsp	42029	***42167.0***	***42365.2***	43174.0	43502.8	42443.0	42526.0	42862.5	43041.1	43924.1	43964.9
Average	36131.2	***36148.1***	***36175.9***	36314.4	36420.8	36186.7	36204.7	36376.2	36856.4	37036.8	37077.6

개 데이터 중 6개에서 알려진 최적값을 도출할 수 있었고 나머지 4개에서는 다른 기법들보다 우수한 결과를 보였다. Best 결과를 비교해 보면 CombGRI&FGI는 7개 데이터에서 알려진 최적값을 찾았고 2개 데이터에서 다른 기법들보다 우수한 결과를 보였다. 다만 kroB200.tsp 데이터에 대해서만 GRI가 CombGRI&FGI에 비해 근소한 차이로 우수한 성능을 발휘하였는데, 이는 비슷한 수준으로 볼 수 있다. 전체 평균에 의하면 CombGRI&FGI가 36175.9로 다른 기법들보다 우수함을 확인할 수 있다.

한편 Table 4에서 FGI를 다른 기법과 비교해보면 Best와 평균값 모두에 있어서 RNN-SA나 HVNS보다 우수하지만, GRI보다는 성능이 떨어짐을 알 수 있다. 그러나, CombGRI&FGI가 GRI보다 우수한 결과를 보인 점으로 보아 상대적으로 성능이 떨어지는 FGI를 통해 FGI보다 우수한 GRI의 성능을 더욱 향상시킬 수 있었다고 볼 수 있다. 이는 GRI와 FGI를 결합하여 적용함으로써 서로의 탐색 특성이 잘 보완되어 전체적으로 성능이 향상될 수 있었던 것으로 해석된다.

V. Conclusions and Future Works

본 논문에서는 순회 외판원 문제의 해결을 위한 지역 탐색의 이웃해 생성 방안으로 그리디 기반 이웃해 생성 방법인 Full Greedy Insertion(FGI) 알고리즘을 제안하였으며, 아울러 기존의 Greedy Random Insertion(GRI)과 FGI를 결합한 이웃해 생성 방안을 제안하였다. FGI는 단독으로는 GRI보다 우수한 성능을 발휘하지 못했지만 GRI와의 간단한 결합을 통해 GRI를 비롯한 기존의 지역 탐색 기법들보다 우수한 결과를 도출할 수 있음을 확인하였다.

본 논문에서는 이웃해 생성 방안으로 GRI와 FGI를 결합하는 방안을 제시하였지만, 이웃해 생성 방법 특히 그리디 이웃해 생성 방법들을 서로 결합하는 방안은 매우 다양할 수 있다. 향후 우수한 이웃해 생성 방법의 개발과 함께 이웃해 생성 방법들을 결합하는 방안에 대해서도 추가 연구를 수행할 필요가 있다. 또한 본 논문에서 제시한 이웃해 생성 방안은 순회 외판원 문제뿐만 아니라 다른 조합 최적화 문제에 대해서도 충분히 효과적일 것으로 예상된다. 따라서 다양한 조합 최적화 문제에 대한 그리디 이웃해 생성 방법의 개발 및 결합 방안에 대한 지속적인 연구가 필요하다.

ACKNOWLEDGEMENT

This research was supported by Kumoh National Institute of Technology(2022~2023).

REFERENCES

[1] D. S. Johnson, and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," Local Search in Combinatorial Optimization, John Wiley & Sons, pp. 215-310, 1997.

[2] T. Stützle, "Local Search Algorithms for Combinatorial Problems," Darmstadt University of Technology Ph.D Thesis, 20, 1998.

[3] M. Baghel, S. Agrawal, and S. Silakari, "Survey of Metaheuristic Algorithms for Combinatorial Optimization," International Journal of Computer Applications, Vol. 58, No. 19, pp. 21-31, Nov. 2012. DOI:10.5120/9391-3813

[4] 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

[5] R. Khairuddin, and Z. M. Zainuddin, "A Comparison of Simulated

Annealing Cooling Strategies for Redesigning a Warehouse Network Problem," Journal of Physics: Conference Series, Vol. 1366. No. 1, p. 012078, 2019. DOI:10.1088/1742-6596/1366/1/012078

[6] J. Hwang, and Y. Kim, "Greedy-based Neighbor Generation Methods of Local Search for the Traveling Salesman Problem," Journal of the Korea Society of Computer and Information, Vol. 27, No. 9, pp. 69-76, Sep. 2022. DOI:10.9708/JKSCI.2022.27.09.069

[7] F. Busetti, Simulated annealing overview, https://www.researchgate.net/publication/238690391_Simulated_annealing_overview, 2001.

[8] M. A. Rahman, and H. Parvez, "Repetitive Nearest Neighbor Based Simulated Annealing Search Optimization Algorithm for Traveling Salesman Problem," Open Access Library Journal, Vol. 8, No. 6, e7520, June 2021. DOI:10.4236/oalib.1107520

[9] 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. DOI: 10.1016/j.asoc.2018.03.048

[10] J. Xu, and J. Zhang, "Exploration-Exploitation Tradeoffs in Metaheuristics: Survey and Analysis," Proceedings of the 33rd Chinese Control Conference, IEEE, pp. 8633-8638, July 2014. DOI:10.1109/chicc.2014.6896450

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

Authors

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.

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.