JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 10, pp. 27-35, October 2021
https://doi.org/10.9708/jksci.2021.26.10.027

Neighbor Generation Strategies of Local Search for Permutation-based Combinatorial Optimization

Junha Hwang*

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

[Abstract]

Local search has been used to solve various combinatorial optimization problems. One of the most important factors in local search is the method of generating a neighbor solution. In this paper, we propose neighbor generation strategies of local search for permutation-based combinatorial optimization, and compare the performance of each strategies targeting the traveling salesman problem. In this paper, we propose a total of 10 neighbor generation strategies. Basically, we propose 4 new strategies such as Rotation in addition to the 4 strategies such as Swap which have been widely used in the past. In addition, there are Combined1 and Combined2, which are made by combining basic neighbor generation strategies. The experiment was performed by applying the basic local search, but changing only the neighbor generation strategy. As a result of the experiment, it was confirmed that the performance difference is large according to the neighbor generation strategy, and also confirmed that the performance of Combined2 is the best. In addition, it was confirmed that Combined2 shows better performance than the existing local search methods.

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

[요 약]

지역 탐색은 다양한 조합 최적화 문제들을 해결하기 위해 활용되어 왔다. 지역 탐색에 있어서 가장 중요한 요소 중 하나가 이웃해를 생성하는 방법이다. 본 논문에서는 순열 기반 조합 최적화를 위한 지역 탐색의 이웃해 생성 전략들을 제안하고, 순회 외판원 문제를 대상으로 각 전략들의 성능을 비교한다. 본 논문에서는 총 10가지 이웃해 생성 전략을 제안한다. 기본적으로 기존에 많이 사용했던 Swap 등 4가지 전략 이외에 Rotation 등 4가지 기법을 새롭게 제안한다. 이외에 기본 이웃해 생성 전략들을 결합하여 만든 Combined1과 Combined2가 있다. 실험은 기본적인 지역 탐색을 적용하되 이웃해 생성 전략만 변경하여 수행하였다. 실험 결과, 이웃해 생성 전략에 따라 성능 차이가 큰 것을 확인하였으며 아울러 Combined2의 성능이 가장 좋음을 확인하였다. 뿐만 아니라 Combined2는 기존의 지역 탐색 기법들보다 더 좋은 성능을 발휘함을 확인하였다.

▸ **주제어**: 이웃해 생성, 지역 탐색, 순열 기반 조합 최적화, 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
• Received: 2021. 08. 31, Revised: 2021. 10. 14, Accepted: 2021. 10. 18.

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

I. Introduction

조합 최적화 문제는 n개의 변수와 도메인, 목적 함수 및 제약조건들로 표현된다. 도메인은 각 변수가 가질 수 있는 값들의 집합을 의미한다. 조합 최적화는 변수들이 제약조건을 모두 만족하면서 가질 수 있는 가능한 후보해들 중 목적 함수에 따른 최적의 해를 찾는 것이 목표이다. 문제에 따라 목적 함수 값을 최대화 또는 최소화하는 해를 찾기 위해 노력한다.

조합 최적화 문제들 중 하나의 후보해가 순열로 표현되는 문제를 순열 기반 조합 최적화 문제라 한다. 즉, n개의 변수에 대한 각 변수의 도메인은 1부터 n까지의 정수값을 가지며, 하나의 후보해는 1부터 n까지 값으로 이루어진 순열 중 하나로 표현된다. 대표적인 순열 기반 조합 최적화 문제로는 순회 외판원 문제가 있으며, 플로우 샵 스케줄링 문제, 선형 순서 문제, 2차 할당 문제 등이 순열 기반 조합 최적화 문제에 포함된다[1].

지금까지 조합 최적화를 위해 메타휴리스틱 알고리즘들이 많이 사용되어 왔는데, 이는 크게 지역 탐색과 집단 기반 탐색으로 나뉜다[2]. 지역 탐색은 하나의 후보해를 현재해로 설정하고 현재해를 수정하여 만든 이웃해를 기반으로 반복적으로 해를 개선해 나가는 탐색 기법으로 언덕 오르기 탐색, 시뮬레이티드 어닐링, 타부 탐색 등이 있다. 집단 기반 탐색은 여러 개의 후보해들로 이루어진 집단으로부터 출발하여 집단 내 후보해들끼리 정보를 교환하면서 집단을 개선해 나가는 탐색 기법으로 유전 알고리즘, 개미 시스템 등이 있다.

본 논문에서는 순열 기반 조합 최적화를 위한 지역 탐색의 이웃해 생성 전략들을 제안한다. 지역 탐색 기법들에 있어서 공통적으로 가장 중요한 요소는 이웃해를 생성하는 방법이라 할 수 있다. 그러나 지금까지 대부분의 연구에서는 각 탐색 기법 별로 해당 기법이 갖고 있는 고유의 탐색 요소들을 발전시키기 위해 노력해 왔다. 시뮬레이티드 어닐링의 경우 온도 냉각 스케줄링이나 응용 문제로의 적용에 관한 연구가 많았고[3, 4], 타부 탐색의 경우 이전 해를 저장하기 위한 메모리 구조나 응용 문제로의 적용에 관한 연구가 주를 이루었다[5]. 따라서 본 논문에서는 지역 탐색의 기본 요소인 이웃해 생성 전략에 초점을 맞추어 다음과 같은 두 가지 연구 목적을 제시한다. 첫째, 순열 기반 조합 최적화를 위해 일반적으로 적용 가능한 다양한 이웃해 생성 전략을 제안한다. 둘째, 순회 외판원 문제를 대상으로 각 이웃해 생성 전략들의 성능을 비교 검토한다.

본 논문에서 제안하는 이웃해 생성 전략은 총 10가지이다. 먼저 기존에 순회 외판원 문제를 위해 많이 사용되었던 Swap, Inversion, EdgeInsertion, BlockInsertion이 있으며, 이외에는 본 연구를 통해 개발한 방법들이다. 새로 제안하는 기본 이웃해 생성 전략으로는 BlockSwap, Rotation, RandomShuffle, GreedyOrdering이 있다. 또한 Swap, Inversion, EdgeInsertion, BlockInsertion을 함께 사용하는 Combined1과 GreedyOrdering을 추가로 함께 사용하는 Combined2를 제안한다. 지역 탐색 기법으로는 First-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링을 사용하며, 각각 기본적인 알고리즘 내에서 이웃해 생성 전략만 변경하여 적용해 봄으로써 각 이웃해 생성 전략들의 성능을 비교한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 지역 탐색과 순회 외판원 문제에 대해 기술한다. 3장에서는 본 논문에서 제안하는 지역 탐색의 이웃해 생성 전략들에 대해 자세히 설명하며, 4장에서 이에 대한 실험 결과를 살펴본다. 마지막으로 5장에서 결론 및 향후 과제에 대해 논의한다.

II. Related Works

1. Local Search

가장 기본적인 지역 탐색 기법으로는 언덕 오르기 탐색이 있다. 언덕 오르기 탐색은 현재해로부터 이웃해를 만들고 현재해보다 더 좋다면 이웃해를 현재해로 설정한 후 동일한 과정을 반복 수행한다. 언덕 오르기 탐색 기법은 크게 두 가지 버전이 존재한다[6]. 하나는 Steepest-ascent 언덕 오르기 탐색이며 다른 하나는 First-choice 언덕 오르기 탐색이다. Steepest-ascent 언덕 오르기 탐색은 현재해에 대한 모든 이웃해들 중 가장 좋은 해로 이동하되 현재해보다 좋은 해가 존재하지 않으면 프로그램은 종료된다. First-choice 언덕 오르기 탐색은 이웃해를 하나만 생성하고 현재해보다 좋으면 바로 이동하게 된다. 두 가지 버전의 언덕 오르기 탐색 모두 전역 최적해에 도달하지 못하고 지역 최적해에 머문다는 문제점이 있다.

언덕 오르기 탐색의 지역 최적화 문제를 해결하기 위해 다양한 지역 탐색 기법들이 등장하였으며, 대표적인 지역 탐색 기법으로 타부 탐색과 시뮬레이티드 어닐링이 있다. 타부 탐색은 Steepest-ascent 언덕 오르기 탐색과 같이 모든 이웃해들 중 가장 좋은 이웃해로 이동하되 그 해가 현재해보다 좋지 않더라도 이동하게 된다. 시뮬레이티드 어닐링은 First-choice 언덕 오르기 탐색과 마찬가지로

하나의 이웃해만 생성하고 현재해보다 더 좋으면 바로 이동하게 된다. 단, 현재해보다 좋지 않다 하더라도 확률적으로 이동할 수 있다.

이상의 모든 지역 탐색 기법들에 대해 본 논문에서 제안하는 이웃해 생성 전략들을 적용해 볼 수도 있지만, Steepest-ascent 언덕 오르기 탐색과 타부 탐색은 모든 이웃해들 또는 여러 개의 이웃해들을 어떻게 만들 것인지에 대한 추가적인 고려가 있어야 한다. 따라서 하나의 이웃해를 대상으로 탐색을 진행하는 First-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링이 이웃해 생성 전략에 대한 효과를 검증하는 데 보다 적합한 것으로 판단하였다. 이에 따라 본 논문에서는 이웃해 생성 전략의 효과를 검증하기 위해 지역 탐색 기법들 중 First-choice 언덕 오르기 탐색과 시뮬레이티드 어닐링을 사용하였다.

최소화 문제를 대상으로 한 기본적인 시뮬레이티드 어닐링 알고리즘은 Fig. 1과 같다. 먼저 초기해를 만들고 온도(T)를 Tstart 값으로 초기화한다. 그리고 종료 조건을 만족할 때까지 다음과 같이 이웃해로 반복하여 이동한다. 이웃해를 하나 생성하고 목적함수 값(ObjValue)이 현재해보다 더 좋다면, 즉 목적함수 값이 더 작다면 이웃해로 이동한다(7~8라인). 만약 이웃해가 현재해보다 좋지 않다면 $e^{-\Delta E/T}$의 확률로 이동하게 된다(9~10라인). 이때 e는 자연로그 상수를 의미한다. 그리고 나서 온도를 변경하되 온도가 최소값(Tmin)보다 작다면 최소값으로 설정한다.

```
 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 것이 온도 스케줄링으로 알려져 있다. 온도는 시간이 갈수록 작아지게 되는데 Linear 방식, Geometric 방식, Logarithmic 방식 등 다양한 방식이 개발되어 왔다[4]. 본 연구에서는 Fig. 1의 11라인에 있는 온도 업데이트를 위해 가장 많이 사용되고 있는 Geometric 방식을 사용한다. 이는 식 1과 같으며 α 값은 보통 0.5와 1 사이의 상수값을 사용한다. 결국 Fig. 1에 존재하는 상수 파라미터로는 Tstart, Tmin, α가 있으며, 본 연구에서 각 값은 실험을 통해 결정하였다.

$$T = \alpha T \tag{1}$$

최대화 문제인 경우 Fig. 1에서 7라인의 부등호를 ≥와 같이 변경하고 10라인의 확률을 $e^{\Delta E/T}$와 같이 변경하면 된다. 그리고 Fig. 1에서 3라인과 9~12라인만 제외하면 First-choice 언덕 오르기 탐색이 된다.

2. Traveling Salesman Problem

순회 외판원 문제는 n개의 도시가 주어질 때 모든 도시들을 단 한 번만 방문하고 출발 도시로 되돌아오되 방문거리를 최소화하는 문제로 정의된다[7]. 예를 들어 6개의 도시를 방문한다고 가정할 때 0번 도시부터 출발한다면 하나의 해, 즉 나머지 5개의 도시를 방문하는 순서는 (4, 2, 5, 1, 3)과 같이 1부터 5까지의 순열로 표현된다. 이와 같이 n개의 도시가 주어질 때 순회 외판원 문제의 모든 후보해의 개수는 $(n-1)!$이 된다.

순회 외판원 문제를 해결하기 위한 기법으로 정수 계획법과 같이 최적해의 도출을 보장하는 알고리즘에 관한 연구도 있지만[8], 도시의 개수가 많아짐에 따라 고려해야 할 후보해의 개수가 기하급수적으로 늘어나기 때문에 최적해를 도출하기까지의 소요 시간이 과도해질 수 있다. 따라서 주로 유전 알고리즘, 개미 시스템, 시뮬레이티드 어닐링, 타부 탐색 등과 같은 메타휴리스틱 알고리즘의 적용에 관한 연구 및 메타 휴리스틱 기법들을 결합하는 방법에 대한 연구가 활발히 진행되어 왔다[9].

순회 외판원 문제 해결을 위해 시뮬레이티드 어닐링과 같은 지역 탐색 알고리즘을 적용하는 경우 주로 Swap, Inversion, EdgeInsertion, BlockInsertion과 같은 이웃해 생성 전략이 사용되어 왔다. 기존 연구 [10]에서는 Nearest Neighbor 방법을 사용하여 여러 개의 초기해를 만들고 시뮬레이티드 어닐링을 사용하여 개선하는 방법을 사용하였는데, 시뮬레이티드 어닐링의 이웃해 생성 전략으로 Swap, Inversion, EdgeInsertion을 사용하였다. 가변 이웃 탐색 알고리즘을 사용한 기존 연구 [11]에서는 Swap을 기반으로 한 이웃해 생성 전략을 사용하였다.

본 연구에서는 순열 기반 조합 최적화를 위한 지역 탐색의 다양한 이웃해 생성 전략을 제시하고 대표적인 순열 기반 조합 최적화 문제인 순회 외판원 문제에 적용한 결과를 검토한다.

III. Neighbor Generation Strategies

본 논문에서 제안하는 모든 이웃해 생성 전략은 순회 외판원 문제뿐만 아니라 다른 순열 기반 조합 최적화 문제에도 그대로 적용이 가능하다. 다만 GreedyOrdering은 대상 문제에 대한 지식을 필요로 하기 때문에 대상 문제에 따라 적용 방법이 달라질 수 있다. 그렇다 하더라도 복잡한 처리를 필요로 하지 않으므로 다른 문제로의 적용이 어렵지 않을 것으로 판단된다. 본 논문에서 제안하는 10가지 이웃해 생성 전략은 다음과 같다.

Swap : 2개의 위치를 무작위로 선택하여 해당 값을 서로 교환한다. 예를 들어 Fig. 2와 같이 8개의 변수로 이루어진 현재해가 (7, 4, 1, 5, 2, 8, 6, 3)일 때 위치 3과 위치 7이 선택되었다면 1과 6의 값을 서로 교환한다.

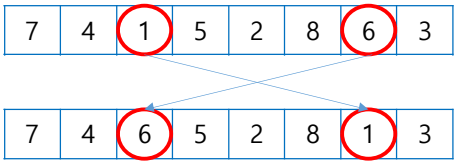

Fig. 2. Neighbor Generation with Swap

Inversion : 현재해로부터 무작위로 2개의 위치를 선택하고 그 사이에 있는 값들을 역전시킨다. Fig. 3은 위치 3과 7이 선택되었을 때의 예를 보인 것이다.

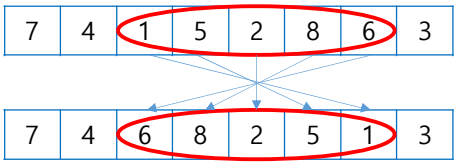

Fig. 3. Neighbor Generation with Inversion

EdgeInsertion : 1개의 위치를 무작위로 선택하고 해당 값을 임의의 위치에 삽입한다. Fig. 4의 예는 위치 3이 선택된 후 위치 6과 7 사이에 삽입된 상황이다.

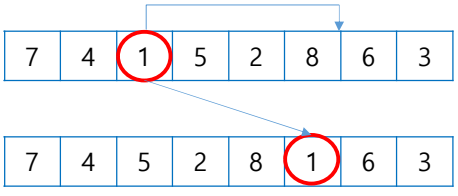

Fig. 4. Neighbor Generation with EdgeInsertion

BlockInsertion : 무작위로 2개의 위치를 선택하고 그 사이에 있는 값들을 그대로 임의의 다른 위치로 삽입한다. Fig. 5는 위치 2와 4 사이의 값들이 위치 6과 7 사이에 삽입된 예를 보여준다.

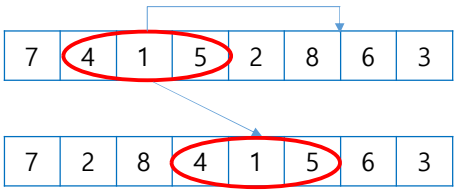

Fig. 5. Neighbor Generation with BlockInsertion

BlockSwap : 4개의 위치를 무작위로 선택하고 첫 번째와 두 번째 위치 사이의 값들을 세 번째와 네 번째 사이의 값들과 교환한다. Fig. 6은 위치 2, 4, 6, 7이 선택되어 위치 2~4의 값들이 위치 6~7의 값들과 교환된 예이다.

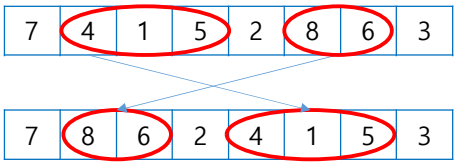

Fig. 6. Neighbor Generation with BlockSwap

Rotation : 2개의 위치를 무작위로 선택하고 그 사이의 값들을 왼쪽 방향으로 k 번 이동한다. k 값은 1 이상이고 선택된 값들의 개수보다 작은 값들 중 무작위 값으로 결정된다. Fig. 7의 예는 위치 3, 7이 선택된 후 왼쪽으로 2만큼 이동한 상황을 보여준다.

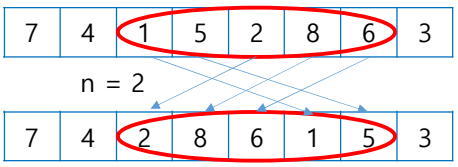

Fig. 7. Neighbor Generation with Rotation

RandomShuffle : k개의 위치를 무작위로 선택하고 해당 위치의 값들을 무작위로 섞는다. 이때 k 값이 너무 클 경우 무작위 탐색과 같아질 수 있으므로 k 값을 2 이상 4 이하의 값으로 제한한다.

GreedyOrdering : 2개의 위치를 무작위로 선택하고 그 사이의 값들을 재정렬한다. Fig. 8은 위치 3과 6이 선택되

었을 때 그 사이 값들인 (1, 5, 2, 8)을 재정렬하는 방법을 보여준다. 먼저 (1, 5, 2, 8)을 현재해에서 삭제한다. 그리고 첫 번째 값인 1부터 마지막 값인 8까지 하나씩 차례로 원래 위치로 삽입한다. 이때 현재 삽입 중인 값이 삽입되었을 때 가장 좋다고 판단되는 위치에 삽입하게 된다. 예를 들어 step 1에서 첫 번째 값인 1이 삽입될 수 있는 위치는 단 하나이므로 선택의 여지가 없다. step 2에서는 두 번째 값인 5가 삽입될 때 2개 위치 중 하나로 삽입되며, step 4의 경우 마지막 값인 8이 삽입될 수 있는 4개 위치 중 하나로 삽입된다. 가장 좋은 삽입 위치를 판단하는 것은 대상 문제마다 달라질 수 있다. 순회 외판원 문제의 경우 현재 삽입되는 도시를 삽입했을 때 거리가 가장 적게 증가하는 위치를 가장 좋은 삽입 위치로 판단한다.

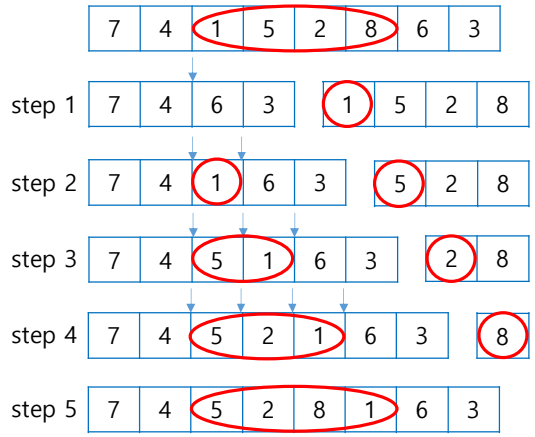

Fig. 8. Neighbor Generation with GreedyOrdering

Combined1 : Swap, Inversion, EdgeInsertion, BlockInsertion 중 하나를 무작위로 선택하여 적용한다.

Combined2 : Swap, Inversion, EdgeInsertion, BlockInsertion, GreedyOrdering 중 하나를 무작위로 선택하여 적용한다.

IV. Experimental Results

1. Experimental Environment

본 연구에서는 실험 데이터로 TSPLIB에서 제공하는 순회 외판원 문제 데이터들 중 Table 1과 같이 10개를 선별하여 사용하였다[12]. 각 데이터는 도시 개수가 58개부터 318개까지 다양하게 분포되어 있다. 현재 모든 데이터에 대한 최적값이 알려져 있으며, Table 1의 Optimal 칼럼에 각 데이터의 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 어닐링(SA)을 위해 필요한 파라미터는 초기 *T*값(Tstart), 최종 *T*값(Tmin), 온도 감소율(α)이 있으며 본 연구에서는 데이터에 따라 Table 2와 같이 이 값들을 설정하였다. 모든 실험에 있어서 Tmin은 1.0으로 고정하였으며, 적절한 Tstart와 α 값은 예비 실험을 통해 결정하였다. 예비 실험에서는 이웃해 생성 전략들 중 Inversion, EdgeInsertion, Combined1, Combined2를 대상으로 하였다. 각 이웃해 생성 전략 및 각 데이터를 대상으로 Tstart 값을 10, 100, 1000 그리고 α 값을 0.99999, 0.999995, 0.999999로 설정한 조합인 총 9가지 경우에 대해 각각 1회씩 실험을 수행하였다. 각 실험 당 최대 수행 시간은 30분으로 제한하였다.

실험 결과, Combined2는 Tstart가 100이고 α 값이 0.999999일 때 평균적으로 성능이 가장 좋았으며 나머지는 Tstart가 1000이고 α 값이 0.999999일 때 성능이 가장 좋았다. 따라서 최종적으로 Tstart를 100, α를 0.999999로 설정한 경우와 Tstart를 1000, α를 0.999999로 설정한 두 가지 경우에 대해 본 실험을 수행하였다. 결국 본 실험에서 Combined2와 BlockSwap만 Tstart가 100일 때 성능이 더 우수했으며, 나머지는 Tstart가 1000일 때 더 우수했다. 따라서 이후로 기술하는 실험 결과는 Table 2와 같이 Combined2와 BlockSwap은 Tstart가 100이고 α 값이 0.999999일 때, 나머지는 Tstart가 1000이고 α 값이 0.999999일 때의 결과를 기술한 것이다. 물론 예비 실험에서는 매우 제한적인 파리미터 값의 조합에 대해서만 실험을 수행하였기 때문에 이웃해 생성 전략이나 데이터에 따라 더 좋은 파리미터 값이 존재할 가능성은 남아 있다. 그

러나 Table 2의 파라미터 설정은 이웃해 생성 전략들 사이의 성능 차이를 파악하는 데 충분할 것으로 판단된다.

Table 2. Parameter Settings for SA

Neighbor Strategy	Tstart	Tmin	α
Swap	1000	1	0.999999
Inversion	1000	1	0.999999
EdgeInsertion	1000	1	0.999999
BlockInsertion	1000	1	0.999999
BlockSwap	100	1	0.999999
Rotation	1000	1	0.999999
RandomShuffle	1000	1	0.999999
GreedyOrdering	1000	1	0.999999
Combined1	1000	1	0.999999
Combined2	100	1	0.999999

본 연구의 모든 실험은 Intel Core I7-6700K CPU 4.0GHz, 8GB RAM 및 Windows 10 64비트 운영체제 PC 상에서 수행되었으며, 지역 탐색 기법과 이웃해 생성 전략 등 모든 프로그램은 Python 언어로 작성되었다.

2. Experimental Results

Table 3은 각 데이터에 대해 First-choice 언덕 오르기 탐색(FCHC) 적용 시 이웃해 생성 전략을 달리하여 실험한 결과이며, Table 4는 시뮬레이티드 어닐링(SA)을 적용했을 때의 실험 결과이다. 각 수치는 각 데이터 및 이웃해 생성 전략 별로 총 5회의 실험을 실시한 결과에 대한 평균값이다. 1회 실험 시 최대 수행 시간은 1800초(30분)로 제한하였다. 모든 데이터에 있어서 30분은 어느 정도 최종해로 수렴하는 데 충분한 시간으로 판단하였다. 실제로는 각 데이터 별로 차이가 나지만 대부분 훨씬 짧은 시간 내에 최종해가 도출되었다. 예를 들면, 시뮬레이티드 어닐링에서 Swap을 사용한 경우 brazil58.tsp 데이터는 최종해가 도출되기까지 약 30초가 소요되었으며, lin318.tsp는 약 350초가 소요되었다. Combined2의 경우 brazil58.tsp는 1초 이내에 최종해가 도출되었고 lin318.tsp는 약 1600초가 소요되었다. Fig. 9는 각 이웃해 생성 전략 별로 모든 데이터에 대한 평균값을 그래프로 나타낸 것이다. 즉, Table 3과 Table 4의 평균(Average) 행을 의미한다. 각 데이터 별로 모든 이웃해 생성 전략들 중 가장 좋은 결과에 대해서는 볼드체 및 이탤릭체로 표시하였으며, Combined1과 Combined2를 제외한 단일 이웃해 생성 전략들 중 가장 좋은 결과에 대해서는 볼드체로 표시하였다.

FCHC와 SA 모두 Combined2의 성능이 가장 우수하고 다음으로는 Combined1의 성능이 우수하다. 여러 개의 이웃해 생성 전략을 함께 사용함으로써 성능이 향상될 수 있음을 알 수 있다. 단일 이웃해 생성 전략의 성능에 있어서는 FCHC와 SA의 결과가 다소 다르게 나타났다. FCHC의 경우 BlockInsertion의 성능이 가장 우수하며, SA의 경우 GreedyOrdering이 가장 우수하다. FCHC와 SA 공통적으로 GreedyOrdering, BlockInsertion, Rotation이 전반적으로 우수한 성능을 보였다. 특이한 점은 본 연구에서 새로 추가한 BlockSwap과 Rotation의 성능이다. BlockSwap의 경우 평균적으로는 다른 우수한 이웃해 생성 전략들에 미치지 못하지만 FCHC에서 데이터에 따라 매우 우수한 성능을 보였다. Rotation의 경우 FCHC에서 평균적으로 BlockInsertion 다음으로 성능이 가장 우수하였으며, SA에서도 GreedyOrdering 다음으로 가장 우수하였다. RandomShuffle은 예상한 바와 같이 성능이 매우 저조하지만, 이외의 새로 추가된 이웃해 생성 전략들은 대상 문제에 따라 효과적일 수 있음을 알 수 있다.

Fig. 10은 lin318.tsp 데이터에 대한 각 이웃해 생성 전략 별 수렴 추세를 나타낸 것이다. 각 이웃해 생성 전략 별로 5회 실험 중 3등을 차지한 결과를 대상으로 하였다. 수행 시간은 수렴 추세 확인의 편의상 300초까지만 표시하였다. Fig. 10에 의하면 GreedyOrdering과 Combined2만 실행 초기 1초 이내의 매우 빠른 시간 내에 목적 함수 값 50,000 이하의 우수한 해를 도출할 수 있었으며, 이외의 다른 이웃해 생성 전략들은 1초 이내에 350,000 정도까지 매우 빠르게 감소하다 이후로 완만하게 감소하고 있음을 알 수 있다. Combined2와 GreedyOrdering은 성능이 우수할 뿐만 아니라 수렴 추세 또한 매우 유사하게 나타났다. Combined2는 GreedyOrdering을 포함하고 있다. 따라서 이는 GreedyOrdering이 매우 빠르게 수렴하는 특성을 가지고 있음을 보여준다.

Fig. 10에서 한 가지 특이한 사항은 BlockSwap의 수렴 추세이다. Combined2와 GreedyOrdering에는 미치지 못하지만 이외의 다른 이웃해 생성 전략들보다 빠르게 수렴하고 있음을 알 수 있다. BlockSwap의 경우 앞서 설명한 바와 같이 평균적으로는 우수한 성능을 발휘하지 못하였지만 대상 문제 및 데이터에 따라 효과적인 성능을 발휘할 수도 있을 것으로 판단된다.

Table 3. Experimental Results for FCHC

Data	Swap	Inversion	Edge Insertion	Block Insertion	Block Swap	Rotation	Random Shuffle	Greedy Ordering	Combined 1	Combined 2
brazil58.tsp	36370.4	27528.2	29943.2	**26240.4**	26413.4	26387.8	34361.4	26904.0	25699.0	***25464.6***
eil76.tsp	779.0	606.8	613.2	565.6	**560.8**	565.6	678.8	573.0	562.4	***552.8***
rat99.tsp	2174.0	1418.8	1479.6	1299.4	**1273.2**	1292.8	1777.6	1350.6	1258.8	***1237.2***
pr136.tsp	189761.6	113929.4	133912.6	102896.4	**101039.8**	104506.8	163477.0	102565.2	101959.2	***100751.8***
kroA150.tsp	55897.2	30766.8	37244.0	28432.8	**27566.0**	28976.8	51016.6	28102.8	27957.6	***27009.8***
u159.tsp	96779.0	49729.2	57296.4	45071.4	**44120.2**	45515.0	84849.0	47660.2	44993.2	***43542.4***
kroB200.tsp	64240.6	34621.0	44942.4	**31407.4**	32587.4	31869.4	61047.2	32849.4	31231.2	***30384.6***
pr264.tsp	218864.6	57762.6	106589.8	**54707.8**	61538.8	54831.8	193934.0	54744.4	53471.8	***50574.4***
pr299.tsp	131563.2	57631.8	71479.0	52832.6	62828.6	**52676.6**	116560.8	55697.4	51280.4	***50582.8***
lin318.tsp	109536.0	49885.6	69852.8	45993.6	54217.4	**45533.6**	108531.6	47462.2	44612.2	***43635.8***
Average	90596.6	42388.0	55335.3	**38944.7**	41214.6	39215.6	81623.4	39790.9	38302.6	***37373.6***

Table 4. Experimental Results for SA

Data	Swap	Inversion	Edge Insertion	Block Insertion	Block Swap	Rotation	Random Shuffle	Greedy Ordering	Combined 1	Combined 2
brazil58.tsp	26456.0	25725.0	25614.0	25415.0	25582.0	25435.0	27506.0	***25395.0***	***25395.0***	***25395.0***
eil76.tsp	562.8	538.8	541.2	**538.4**	542.8	541.4	573.0	545.8	***538.0***	***538.0***
rat99.tsp	1334.2	1258.0	1249.8	1269.0	1249.2	1232.8	1372.8	**1223.6**	1218.4	***1215.8***
pr136.tsp	115402.0	100862.6	101970.4	100213.2	101505.4	99642.8	119309.4	***97089.6***	97985.2	98518.8
kroA150.tsp	32230.0	27562.8	28128.6	28013.4	27531.6	27479.0	34171.0	**27150.4**	26855.2	***26687.0***
u159.tsp	50117.2	44331.4	43873.0	44802.6	44897.0	43151.4	52729.0	**42794.2**	42795.4	***42546.0***
kroB200.tsp	37570.8	31301.4	31566.2	31205.0	32182.4	**30854.2**	39465.4	31006.4	30030.8	***29666.4***
pr264.tsp	89765.0	53334.0	55562.2	53367.6	59187.8	51368.0	103889.4	**49909.6**	51523.6	***49135.0***
pr299.tsp	67871.0	52307.6	52030.2	52987.4	61378.8	**51014.8**	72224.4	51849.0	49913.8	***48822.8***
lin318.tsp	61741.0	45383.8	48635.4	45561.0	54583.6	**44428.8**	66198.4	45784.6	43714.6	***42886.4***
Average	48305.0	38260.5	38917.1	38337.3	40864.1	37514.8	51743.9	**37274.8**	36997.0	***36541.1***

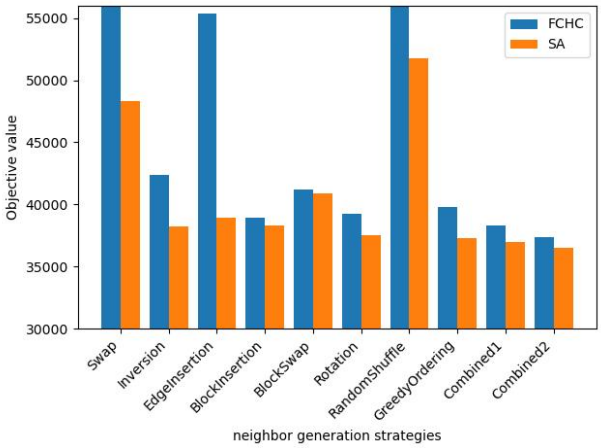

Fig. 9. Average Objective Value Comparison Graph of Neighbor Generation Strategies

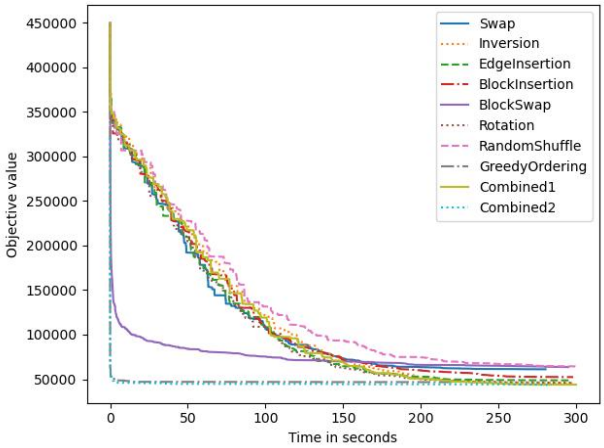

Fig. 10. Convergence Graph with lin318.tsp

Table 5. Comparison of Basic SA(Combined2) with Other Algorithms

Data	Optimal	Basic SA(Combined2)			HVNS(2018) [11]			RNN-SA(2021) [10]		
		Best	Average	Worst	Best	Average	Worst	Best	Average	Worst
brazil58.tsp	25395	**25395.0**	***25395.0***	25395.0	25425.0	25592.7	25664.0	**25395.0**	25440.4	25622.0
eil76.tsp	538	**538.0**	***538.0***	538.0	545.4	552.6	566.5	544.4	549.2	556.0
rat99.tsp	1211	**1211.0**	***1215.8***	1223.0	1240.4	1241.3	1242.4	1219.2	1229.3	1232.7
pr136.tsp	96772	97729.0	98518.8	99208.0	97979.1	***97985.8***	98012.8	96922.4	100335.2	101924.3
kroA150.tsp	26524	26528.0	***26687.0***	26990.0	26943.3	26947.2	26962.6	26821.8	27008.2	27330.0
u159.tsp	42080	42396.0	42546.0	42960.0	42436.2	***42467.6***	42467.6	42162.8	42547.7	42715.5
kroB200.tsp	29437	29489.0	***29666.4***	29979.0	30447.3	30453.2	30472.4	29825.2	30033.0	30247.7
pr264.tsp	49135	**49135.0**	***49135.0***	49135.0	51155.4	51197.1	51364.2	49197.3	49375.8	49779.6
pr299.tsp	48191	48639.0	***48822.8***	49349.0	50271.7	50373.1	50778.9	48811.5	49003.9	49137.7
lin318.tsp	42029	42457.0	***42886.4***	43225.0	43924.1	43964.9	44128.4	42862.5	43041.1	43167.1
Average	36131.2	36351.7	***36541.1***	36800.2	37036.8	37077.6	37166.0	36376.2	36856.4	37171.3

Table 5는 Combined2의 실험 결과와 기존 연구 [11] 및 [10]을 서로 비교한 것이다. 실험을 위한 수행 시간과 관련하여 [11]의 경우 해당 논문에 Best 해를 찾기까지의 소요 시간이 기술되어 있다. 예를 들어, brazil58.tsp가 401초, rat99.tsp가 828초, lin318.tsp가 1734초 등이다. 이는 본 연구에서 Best 해를 찾기까지의 소요 시간보다 더 많이 걸린 것으로 보인다. [10]은 수행 시간이 따로 명시되어 있지 않다. 보다 중요한 것은 적절한 시간 내에 얼마나 좋은 해를 찾을 수 있는가이다. Table 5에서 각 데이터 별로 평균값(Average)들 중 가장 좋은 값들을 볼드체 및 이탤릭체로 표시하였으며, Best 해들 중 최적해에 대해서는 볼드체로 표시하였다.

Table 5에 의하면 pr136.tsp와 u159.tsp을 제외한 모든 데이터에 있어서 Combined2의 성능이 가장 우수함을 알 수 있다. 전체적인 평균값에 있어서도 Combined2의 성능이 가장 우수하며 [10]과 [11]의 순으로 성능이 우수함을 알 수 있다. 이를 통해 기본적인 지역 탐색 알고리즘을 적용하더라도 적절한 이웃해 생성 전략을 개발하여 활용함에 따라 우수한 성능을 발휘할 수 있음을 확인할 수 있다.

V. Conclusions and Future Work

본 논문에서는 순열 기반 조합 최적화를 위한 지역 탐색에 있어서 다양한 이웃해 생성 전략을 제시하였다. 대표적인 순열 기반 조합 최적화 문제인 순회 외판원 문제를 통해 이웃해 생성 전략들의 성능을 검증 및 비교하였는데, 지역 탐색 기법으로는 기본적인 First-choice 언덕오르기 탐색과 시뮬레이티드 어닐링을 사용하였다. 실험 결과, 본 논문에서 새롭게 제시한 GreedyOrdering과 Rotation이 기존의 이웃해 생성 전략들보다 우수함을 확인하였고, BlockSwap 또한 경우에 따라 효과적일 수 있음을 확인하였다. 최종적으로 GreedyOrdering과 Inversion 등을 결합한 이웃해 생성 전략인 Combined2가 가장 우수한 성능을 발휘하였으며, 이 결과는 기존 연구들에 비해서도 훨씬 우수함을 확인하였다.

본 연구에서는 기본적인 지역 탐색 알고리즘만으로도 이웃해 생성 전략에 따라 우수한 성능을 발휘할 수 있음을 확인하였다. 이를 통해 기존의 지역 탐색 관련 연구 성과들을 본 연구와 접목할 경우 더욱 우수한 성과가 도출될 수 있을 것으로 예상할 수 있다. 따라서 향후로 보다 효과적인 이웃해 생성 전략들을 개발함과 동시에 다음과 같은 추가 과제들을 수행하고자 한다. 먼저 순회 외판원 문제 외에 다른 순열 기반 조합 최적화 문제에 적용해 봄으로써 대상 문제에 따른 각 이웃해 생성 전략의 특성을 재확인한다. 다음으로는 본 논문에서 제시한 이웃해 생성 전략과 기존의 지역 탐색 관련 우수 연구 성과들을 결합함으로써 어떤 효과가 발생하는지를 확인한다. 아울러 순열 기반 조합 최적화 문제뿐만 아니라 일반적인 조합 최적화 문제에 적용 가능한 이웃해 생성 전략에 대해서도 관심을 가질 필요가 있다.

ACKNOWLEDGEMENT

This research was supported by Kumoh National Institute of Technology(202001920001).

REFERENCES

[1] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems," Progress in Artificial Intelligence, Vol. 1, No. 1, pp. 103-117, Jan. 2012.

[2] 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.

[3] D. Abramson, H. Krishnamoorthy, and H. Dang, "Simulated annealing cooling schedules for the school timetabling problem," Asia-Pacific Journal of Operational Research, Vol. 16, pp. 1-22, May 1999.

[4] 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, Article ID 012078, Nov. 2019.

[5] V. K. Prajapati, M. Jain, and L. Chouhan, "Tabu search algorithm (TSA): A comprehensive survey," Proceedings of 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE), pp. 1-8, Feb. 2020.

[6] A. Panghal, "A comparative study of searching and optimization techniques in artificial intelligence," International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), Vol. 6, No. 3, pp. 72-80, May-June 2020.

[7] A. Hussain, Y. S. Muhammad, M. N. Sajid, I. Hussain, A. M. Shoukry, and S. Gani, "Genetic algorithm for traveling salesman Problem with modified cycle crossover operator," Computational Intelligence and Neuroscience, Vol. 2017, Article ID 7430125, Oct. 2017.

[8] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," European Journal of Operational Research, Vol. 59, No. 2, pp. 231-247, June 1992.

[9] R. Purkayastha, T. Chakraborty, A. Saha, and D. Mukhopadhyay, "Study and analysis of various heuristic algorithms for solving travelling salesman problem—A Survey," Proceedings of the Global AI Congress 2019, pp. 61-70, 2020.

[10] 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.

[11] 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.

[12] 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 artificial intelligence, combinatorial optimization, machine learning, and programming language.