▩ 연구논문

비대칭 외판원 문제에서 3-Opt를 응용한 새로운 발견적 알고리듬

- A New Heuristic Algorithm for the Asymmetric Traveling Salesman Problem Using 3-Opt -

권 상 호**
Kwon, Sang Ho
강 맹 규**
Kang, Maing Kyu

Abstract

The asymmetric traveling salesman problem is a representative NP-Complete problem. Polynomial algorithm for this problem has not been yet found. So, many heuristic methods have been researched in this problem.

We need heuristic methods that produce good answers for some larger problems in reasonable times. 3-opt is well known for the effective local-search heuristic method. It has been used in many applications of the asymmetric traveling salesman problem.

This paper discusses 3-opt's properties and ineffective aspects and presents a highly effective heuristic method. 3-opt does not consider good arcs(shorter distance or little cost). This paper presents a fast heuritic algorithm compared with 3-opt by inserting good arcs and deleting related arcs later.

1. 서론

외판원문제(traveling salesman problem)는 고전적인 최적화 문제로서 외판원이 고객이 위치하고 있는 N개의 모든 지점을 오직 한 번씩만 방문하는 순환로(이를 해밀턴 순환로(Hamilton cycle)라고 함) 중에서 가장 비용이 싼 순환로를 결정하는 문제이다. 외판원문제는 제약조건을 이완시키거나 변형시킴으로써 다양한 형태의 문제로 바꿀 수 있어 수리계획법의 주된 관심대상으로서 많은 연구가 이루어져 왔다.

외판원문제는 NP-Complete문제의 전형적인 예로서 최적해를 구하는 다항식(polynomial) 계산량을 갖는 알고리듬이 발견되지 않고 있다. 즉, 문제의 크기가 커질수록 최적해를 구하는데 걸리는 시간은 지수적으로 증가한다. 최근 외판원문제의 연구에서는 계산시간, 해의 최적성 등에서 큰 진보를 이루었으나, 프로그램과 데이터구조의 복잡성은 더욱 커지는 결과를 초래하고 있다[3]. 따라서, 최적화 알고리듬대신 발견적 알고리듬(heuristic algorithm)연구가 대부분이

* 본 연구는 1999년 한양대학교 교내 연구비에 의해 연구되었음.

** 한양대학교 산업공학과

다.

발견적 알고리듬으로는 경로구성(tour construction)방법과 경로개선(tour improvement)방법으로 구분할 수 있다[5, 6, 7, 9]. 경로구성방법은 빠른 시간에 국지탐색(local search)을 할 수 있지만 경로개선방법에 비해 해가 좋지 않다[5]. 따라서, 경로구성방법으로 형성한 순환로를 일반적으로 경로개선방법의 초기 순환로로 사용한다[7, 11].

경로개선방법 알고리듬 중 k-opt 알고리듬과 Lin-Kernighan[11] 알고리듬이 좋은 해를 구하는 알고리듬으로 알려져 있다. k-opt 알고리듬은 고전적인 알고리듬으로서 간결하고 빠른 시간에 최적해에 근접하는 해를 제시하고 있어 널리 사용되고 있다[1]. 또한 다른 알고리듬의 부분 알고리듬으로 사용되고 있다. Kanellakis[8]는 Lin-Kernighan 알고리듬을 비대칭 외판원문제에 적용하였다. k-opt는 매우 많이 사용되는 알고리듬임에도 불구하고 1965년 Lin[10]에 의해 소개된 후 알고리듬 자체에 대한 연구는 많지 않다.

최근에 일반화된 발견적 알고리듬엔 Guided Local Search 알고리듬[15], Tabu Search 알고리듬, Genetic 알고리듬, Neural Network 알고리듬 등이 외판원문제에 적용되고 있다[5, 7].

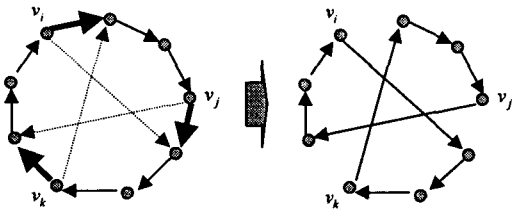
2. 3-Opt 알고리듬

k-opt 알고리듬의 기본 개념은 임의의 해밀턴 순환로에서 k개의 호를 제거하고 다른 k개의 호로 삽입하여 이웃(neighborhood)한 해밀턴 순환로를 형성하는 것이다. 원래의 해밀턴 순환로를 t라고 하면 이웃한 순환로를 $N_k(t)$라고 표현한다. 본 연구에서 사용하는 기호를 정의하면 다음과 같다.

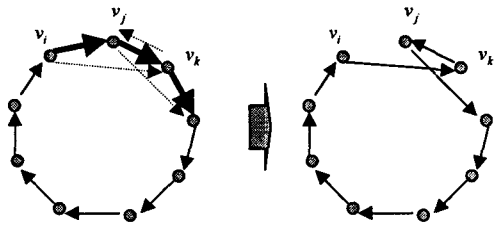
N : 교점의 수
S : 전체호의 집합
t : 임의의 해밀턴 순환로
T : 순환로 t의 호의 집합
$v_1, v_2, \ldots, v_i, \ldots, v_j, \ldots, v_k, \ldots, v_n$: 순환로 T의 교점순서
t' : t의 이웃순환로 $= N_k(t)$
$f(T)$: T의 전체 순환거리(비용)

k=3인 3-opt은 가장 일반적으로 사용된다. 3-opt는 그림2.1에 보인 바와 같이 제거하려는 호가 집합 T에 포함되어 있는 호 (v_i, v_{i+1}), (v_j, v_{j+1}), (v_k, v_{k+1})이면 삽입후보 호(제거하려는 3개의 호를 대체하려는 3개의 호임)는 집합 S-T에 포함되어 있는 호 (v_i, v_{j+1}), (v_j, v_{k+1}), (v_k, v_{i+1})이다. T집합과 S-T집합은 항상 상호 배타적이다. 만약 이웃 순환로 t' 의 비용이 순환로 t보다 적으면 순환로 t를 순환로 t' 로 대체한다. 이때 호 (v_i, v_{i+1}), (v_j, v_{j+1}), (v_k, v_{k+1})은 T집합에서 제거하고 호 (v_i, v_{j+1}), (v_j, v_{k+1}), (v_k, v_{i+1})은 S-T집합에서 제거한다. 3개의 호를 대체할 수 있는 방법은 오직 이 한 가지 방법밖에 없다.

i, j, k값을 순차적으로 증가 시켜가며 이웃 순환로를 찾는다. i, j, k가 가질 수 있는 값은 각각 i=1, 2, ..., n-2, j=i+1, i+2, ..., n-1, k=j+1, j+2, ..., n-1, n이다. 따라서 조사할 수 있는 모든 이웃 순환로의 개수는 $\binom{n}{3}$이다. 모든 이웃 순환로를 조사한 결과 순환로 t의 비용 $f(t)$를 개선할 수 있는 순환로를 발견하지 못하면 알고리듬을 끝내며, 이 때의 순환로 t가 3-opt의 해이다[1]. 만약 발견하면 이 순환로를 새로운 초기해로 하여 위의 과정을 반복한다.



(a)



(b)

그림 2.1 3-opt 알고리듬의 호의 대체

3. k-Opt 알고리듬의 분석

일반적으로 k-opt의 계산량은 $O(N^k)$이다. 따라서, k값이 커지면 계산량이 많아져 효율적인 알고리듬이라 할 수 없다.

2-opt는 대칭 네트워크 모델에서 빠른 시간에 국지탐색 해를 구해 준다. 3-opt는 비대칭 네트워크 모델에 사용하는 알고리듬으로 적절한 시간에 최적해에 근사한 해를 구해 준다[4].

4-opt는 3-opt에 비해 좋은 해를 구할 수 있지만 교점 수가 많아질 경우 효율적인 알고리듬이 되지 못한다. 4-opt는 계산량이 많기 때문에 국지탐색 알고리듬으로 사용하기보다는 3-opt의 국지탐색 해에서 벗어나는 수단으로 사용하기도 한다[12]. 4-opt에서 이웃 순환로를 구성할 수 있는 방법은 3-opt과 같이 한 가지 밖에 없다. 5-opt는 이와 달리 여덟 가지의 다른 방법으로 이웃 순환로를 조사해 갈 수 있다.

k-opt 알고리듬에는 일정한 규칙성이 있다. 첫째, 하나의 초기 순환로에 대한 모든 이웃 순환로에서의 호의 사용빈도는 일정한 규칙성을 가지고 있다. 둘째, 초기 순환로가 같을 경우 모든 이웃 순환로에는 중복이 없다. 그림3.1은 교점 수가 10개인 하나의 초기 순환로(1→2→3→4→5→6→7→8→9→10→1)에 대한 모든 이웃 순환로에서 호의 사용빈도를 나타내는데 초기 순환로에 포함된 호들이 이 초기 순환로의 모든 이웃 순환로에서 다시 사용되는 빈도는 84로 가장 많다. 여기서, k-opt는 초기 순환로에 매우 의존적임을 알 수 있다. 이 외 다른 호들의 사용빈도 또한 일정한 규칙을 보이고 있다.

3-opt에서 제거하려는 3개의 호를 선별하는 기준은 초기 순환로의 교점순서이다. 삽입후보 호인 3개의 호는 제거하려는 3개의 호와 초기 순환로의 교점순서에 따라 결정된다. 따라서, 3-opt에서 삽입후보 호들은 비용이 적은 호들이라고 말할 수 없다.

일반적으로, 어떤 네트워크의 최적 순환로에 포함되지 않는 호들을 찾아낼 수 있다[14]. 따라서, 어떤 네트워크의 최적의 해밀턴 순환로는 각각의 교점에서 나가는 호들의 비용이 적은 순서로 ($\alpha \cdot 100$)%안에 포함되는 호들만으로 구성된다(여기서 α값은 0에서 1사이의 값임).

	1	2	3	4	5	6	7	8	9	10
1	0	84	8	7	6	5	4	3	2	1
2	1	0	84	8	7	6	5	4	3	2
3	2	1	0	84	8	7	6	5	4	3
4	3	2	1	0	84	8	7	6	5	4
5	4	3	2	1	0	84	8	7	6	5
6	5	4	3	2	1	0	84	8	7	6
7	6	5	4	3	2	1	0	84	8	7
8	7	6	5	4	3	2	1	0	84	8
9	8	7	6	5	4	3	2	1	0	84
10	84	8	7	6	5	4	3	2	1	0

그림 3.1 교점 수 10개, 3-opt, 하나의 초기 순환로에 대한
모든 이웃 순환로에서 호의 사용빈도

4. 제안하는 알고리듬

4.1 제안하는 알고리듬의 기본개념

제안하는 알고리듬은 비용이 적고 최적 순환로에 포함될 가능성이 있다고 판단되는 호들을 선정하여 초기 순환로에 삽입을 시도하는 알고리듬이다. 따라서, 모든 호를 초기 순환로에 삽입을 시도하지 않으므로 빠른 시간 내에 국지탐색 해를 구한다.

어떤 네트워크의 최적 해밀턴 순환로는 각각의 교점에서 나가는 호들의 비용이 (α · 100)%안에 포함되는 호들만으로 구성된다. 따라서, 제안하는 알고리듬에서 비용이 적고 최적 순환로에 포함될 수 있다고 판단되는 호들이란 의미는 각각의 교점에서 나가는 호의 비용이 적은 호들을 의미한다.

4.2 알고리듬 절차

알고리듬 절차는 크게 두 부분으로 나눌 수 있다. 첫째는 삽입할 호를 선별하는 과정이고, 둘째는 선별한 호를 삽입하는 과정이다. 그림4.1에서 삽입할 호를 선별하는 과정은 다시 비용행렬의 삭감과정(reduction of cost matrix)[1, 13]과 호의 정렬과정으로 구분한다.

비용행렬의 삭감이란 외판원문제를 행렬로 표현한 후 행의 최소비용을 그 행의 모든 비용으로부터 삭감하는 연산을 모든 행에 대해 반복하고 같은 방법을 열에도 적용하는 과정이다. 열에 최소비용을 삭감할 경우 행의 최소비용을 삭감할 때 이미 삭감된 비용이 있은 열은 삭감하지 않는다.

만약 외판원문제에서 삭감한 비용행렬의 최소비용이 되는 호만으로 해밀턴 순환로를 구성할 수 있으면 이때의 순환로가 최적해이다. 본 연구에서 비용을 삭감하는 이유는 비용이 적은 순으로 (α · 100)%의 호를 선정할 때 모든 호들에 대해 선정될 기회를 공평하게 주기 위해서이다. 예를 들어, 교점이 1에서 4까지 있다고 보자. 교점 1에서 교점 2로의 비용은 21, 교점 3으로의 비용은 22, 교점 4로의 비용은 23이고 교점 2에서 교점 1로의 비용은 1, 교점 3으로의 비용은 2, 교점 4로의 비용은 3일 때 비용이 적은 순으로 (α · 100)%의 호를 선정할 경우 교점 2에서 나가는 호들만이 선정될 수 있다. 또한 교점 1에서 나가는 호들은 하나도 선정되지 않을 수 있다.

삭감한 비용행렬을 크기 순으로 정렬하기 위해 정렬 알고리듬을 사용한다. 전체 호들 중 앞 순위부터 ($\alpha \cdot 100$)% 해당하는 호들에 대해서 삽입을 시도한다. 만약 앞 순위부터 ($\alpha \cdot 100$)% 해당하는 호들을 모두 삽입 시도하여도 순환로의 비용이 적어지지 않을 경우(해가 개선되지 않을 경우) 알고리듬을 끝낸다.

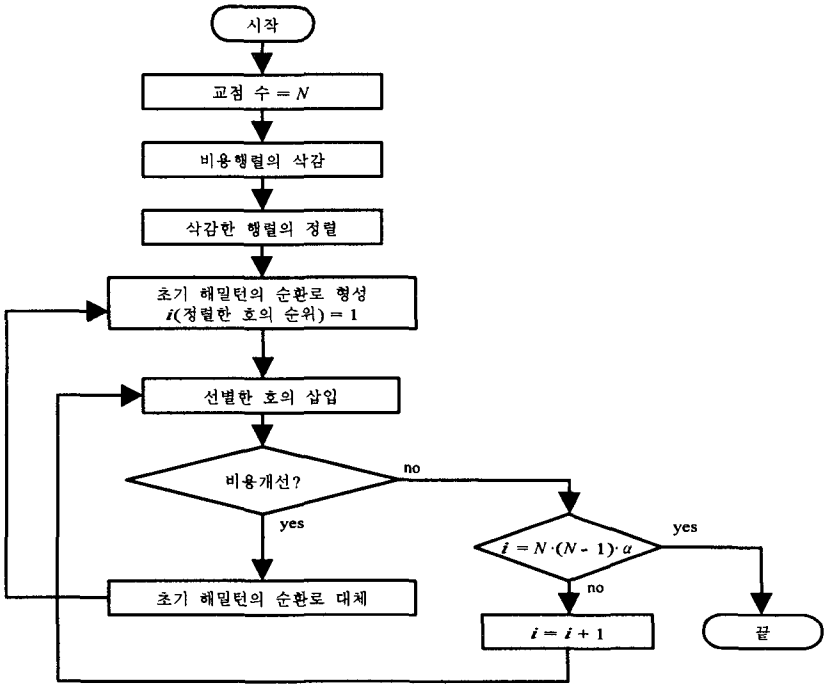


그림 4.1 제안하는 알고리듬의 절차

4.3 선별한 호의 삽입 과정

삽입하는 과정은 3-opt의 기본과정과 같다. 삽입후보인 3개의 호들 중 하나의 호를 선정하면 나머지 2개의 호가 정해져 최종적으로 3개의 호를 결정한다. 3-opt의 기본과정에 따라 제거하려는 다른 3개의 호를 결정한다. 이 과정에서 삽입후보 호들 중 나머지 2개의 호는 교점 수에 따라 여러 가지의 경우가 발생할 수 있다. 제안하는 알고리듬에서는 모든 경우에 대해 삽입과정을 진행한다. 따라서 하나의 호에 대해 삽입을 시도하는 경우 교점 수에 따라 많은 이웃 순환로를 만들 수 있다.

삽입하는 과정은 크게 순방향의 경우와 역방향의 경우로 구분한다. 예를 들어, 초기 해밀턴 순환로가 1→2→3→4→5→1과 같을 경우 호 (2, 5)를 삽입하는 과정은 순방향이고 호 (4, 2)를 삽입하는 과정은 역방향이다. 만약 호 (2, 3)을 삽입하는 경우는 이미 순환로에 포함되어 있는 호이므로 이를 무시하고 정렬한 호들 중 다음 순위의 호에 대해 삽입과정을 진행한다.

3-opt에서 제거하려는 3개 호의 시작교점의 위치가 i, j, k라면 삽입후보 호는 각각 호 (v_i, v_{j+1}), 호 (v_j, v_{k+1}), 호 (v_k, v_{i+1})와 같다. 예를 들어, 위의 예제에서 호 (2, 5)를 삽입할 경우는 호 (v_i, v_{j+1})과 호 (v_j, v_{k+1})에 해당한다. 4.4절 수치예제를 통해 호의 삽입과정을 자세히 알아본다.

4.4 수치예제

그림4.2에서 초기 해밀턴 순환로에 호 (2, 4)를 삽입할 경우 교점2의 위치가 1이고 교점4의 위치가 3이므로 순방향이다. 따라서 삽입할 수 있는 방법은 호 (v_i, v_{j+1})과 호 (v_j, v_{k+1})에 해당한다. 호 (v_i, v_{j+1})의 경우 교점 2의 위치 1은 i값이 되고 교점 4의 위치 3은 j+1값이 된다. 따라서 i값은 1, j값은 2, k값은 3 또는 4가 될 수 있다. 이 경우 두 개의 이웃 순환로를 형성할 수 있다. i, j, k값이 각각 1, 2, 3인 경우 3-opt의 기본과정에 따라 제거하려는 호는 각각 호 (2, 3), (3, 4), (4, 5)이며 삽입후보 호는 각각 호 (2, 4), (4, 3), (3, 5)이다. i, j, k값이 각각 1, 2, 4인 경우 제거하려는 호는 각각 호 (2, 3), (3, 4), (5, 1)이고, 삽입후보 호는 각각 호 (2, 4), (5, 3), (3, 1)이다. 따라서, 호 (v_i, v_{j+1})인 경우 2개의 이웃 순환로를 만들 수 있다. 호 (v_j, v_{k+1})인 경우 교점 1의 위치 1은 j값이 되고 교점 4의 위치 3은 k+1값이 된다. 따라서 i값은 0인 경우밖에 없으며 j값은 1이고 k값은 2이다. 이 경우 i, j, k값이 각각 0, 1, 2이므로 제거하려는 호는 각각 호 (1, 2), (2, 3), (3, 4)이고 삽입후보 호는 각각 호 (1, 3), (3, 2), (2, 4)이다.

호 (4, 3)을 삽입할 경우 교점 4의 위치는 3이고 교점 3의 위치는 2이므로 역방향이다. 역방향인 경우는 호 (v_k, v_{i+1})인 경우에 해당한다. 교점 4의 위치 3은 k값이 되고 교점 3의 위치 2는 i+1값이 된다. 따라서 i값은 1이고 j값은 2인 경우밖에 없으며, k값은 3이 된다. i, j, k값이 각각 1, 2, 3이므로 제거하려는 호는 각각 호 (2, 3), (3, 4), (4, 5)이고 삽입후보 호는 각각 호 (2, 4), (4, 3), (3, 5)이다. 이 경우 하나의 이웃 순환로를 만들 수 있지만 삽입하는 호에 따라 여러 개의 이웃 순환로를 만들 수 있다.

그림4.2에서 호 (4, 1)를 삽입할 경우 역방향에 해당한다. 이 경우는 역방향이며 삽입할 호의 도착교점의 위치가 0인 경우이다. 호 (v_k, v_{i+1})에 해당하지만 도착교점 1의 위치가 0이므로 i값은 -1이 되어 문제가 발생한다. 이런 경우 호 (4, 1)를 삽입하기 위해서 삽입할 호의 출발교점인 4의 위치를 j로 하고 마지막 교점인 5의 위치를 k로 하면 호 (v_j, v_{k+1})을 적용하여 호 (4. 1)을 삽입할 수 있다. 이때 i값은 0부터 (j - 1)값을 취할 수 있다.

- 초기 해밀턴의 순환로(위치값)
  1(0)→2(1)→3(2)→4(3)→5(4)→1(5)
- 호 (2, 4)를 삽입할 경우 →순방향 알고리듬 적용
  - 호 (v_i , v_{j+1})일 경우
    이웃 순환로 : 1→2→4→3→5→1 (i, j, k) = (1, 2, 3)
    이웃 순환로 : 1→2→4→5→3→1 (i, j, k) = (1, 2, 4)
  - 호 (v_j , v_{k+1})일 경우
    이웃 순환로 : 1→3→2→4→5→1 (i, j, k) = (0, 1, 2)
- 호 (4, 3)을 삽입할 경우 →역방향 알고리듬 적용
  - 호 (v_k , v_{i+1})일 경우
    이웃 순환로 : 1→2→4→3→5→1 (i, j, k) = (1, 2, 3)

그림 4.2 호의 삽입과정

4.5 제안하는 알고리듬의 계산량

교점 수가 N이므로 호의 수는 N^2-N이다. 모든 호들을 정렬하는 데 퀵정렬을 사용하면 계산량은 $O(N\ logN)$이다[2]. 전체 호 중 (α · 100)%에 해당하는 호들에 대해 삽입을 시도하기 때문에 이 때의 계산량은 $(N^2-N)\cdot\alpha$ 이다. 하나의 호를 삽입하는 과정에서 평균적으로

N가지의 경우가 발생할 수 있다. 따라서 모든 경우를 고려한 전체의 계산량은 $((N^3-N^2)\cdot\alpha) + N\cdot logN$ 이다.

5. 실험결과와 분석

실험을 위하여 4장에서 제시한 알고리듬 절차를 Visual C++ 6.0으로 프로그램하여 Pentium 166MHz CPU, 메모리 32M를 장착한 IBM호환 PC에서 수행하였다.

5.1 실험환경

알고리듬의 평가기준은 알고리듬의 수행시간과 해의 정확도로 설정하였다. 실험을 위한 문제는 랜덤한 문제와 TSPLIB[6]의 문제를 사용하였다. 초기해는 랜덤하게 설정하여 최근거리인접점 알고리듬을 적용하였다.

우선 랜덤한 문제로는 노드(교점) 수를 50, 100, 150, 200으로 설정하였으며 데이터는 랜덤하게 추출하였다. 랜덤한 문제는 각 노드 수 별로 5개를 설정하여 실험했다. α 값이 0.1일 경우 빠른 시간에 적절한 해를 구해주므로 본 실험에서는 α 값을 0.1로 설정하였다. α 값에 따른 실험결과는 5.2.3항에 나타나 있다. 그리고 5.2.5항에서는 비용행렬의 삭감여부에 따른 해의 정확도를 비교하였다.

표 5.1 실험계획

구분	노드 수	문제번호	α
랜덤한 문제	50	A1, B1, C1, D1, E1	0.1
	100	A2, B2, C2, D2, E2	0.1
	150	A3, B3, C3, D3, E3	0.1
	200	A4, B4, C4, D4, E4	0.1
TSPLIB 문제	17	br17	0.1
	45	ftv44	0.1
	70	ft70	0.1
	71	ftv70	0.1
	100	kro124p	0.1
	170	ftv170	0.1

5.2 실험결과

5.2.1 해의 비교

그림5.1에서 제안하는 알고리듬의 해가 3-opt의 해보다 좋다고 볼 수 있다. 노드 수만큼

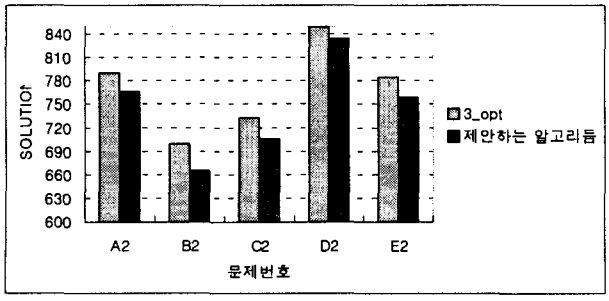


그림5.1 노드 수 100개인 5개의 문제에 대한 해의 비교

의 해를 비교할 경우 제안하는 알고리듬이 3-opt보다 좋은 해를 갖는 비율은 실험결과 약 70-80%이다. 따라서 제안하는 알고리듬은 일반적으로 좋은 해를 준다.

5.2.2 실행시간의 비교

그림5.2의 실행시간에서는 많은 차이가 있다. 제안하는 알고리듬이 3-opt보다 약 8~12배 실행시간이 빠르게 나타난다. 제안하는 알고리듬은 비용이 적은 호들만을 선별하여 삽입을 시도하기 때문에 빠른 시간 내에 국지탐색을 할 수 있다.

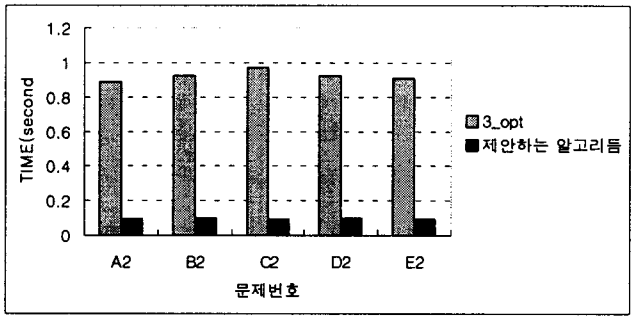


그림5.2 노드 수 100개인 5개의 문제에 대한 실행시간의 비교

5.2.3 α 값의 변화에 따른 비교

표 5.2는 TSPLIB 문제를 사용하여 α 값의 변화에 따른 제안하는 알고리듬의 실행시간과 해를 비교한 결과인데 α 값에 따라 변함을 알 수 있다. α 값이 0.05일 때는 실행기간은 가장 빠르게 나타나지만 해는 나쁘다. 일반적으로 α 값이 0.1 또는 0.15일 경우 해가 더 이상 개선되지 않는다. α 값이 0.15이상일 때 해가 개선된다 해도 개선정도는 미미하다. 따라서 α 값이 0.1 또는 0.15일 때 효율적이라고 할 수 있다.

표 5.2 α 값의 변화에 따른 실행시간과 해의 비교

문제번호	실행시간				해			
/ α	0.05	0.1	0.15	0.2	0.05	0.1	0.15	0.2
br17	0	0	0	0	39	39	39	39
ftv44	0	0.006	0.007	0.009	1697	1692	1690	1690
ft70	0.05	0.085	0.098	0.10	45010	44783	44758	44757
ftv70	0.014	0.022	0.027	0.034	2104	2066	2066	2066
kro124p	0.052	0.075	0.097	0.115	38607	38555	38552	38552
ftv170	0.24	0.33	0.45	0.52	3082	3075	3075	3075

5.2.4 3-Opt와 제안하는 알고리듬 비교

표 5.3은 TSPLIB 문제를 사용하여 3-opt와 제안하는 알고리듬에 대해 실행시간과 해를 비교한 결과이다. 실험에서 α 값은 0.1로 설정하였다.

표 5.3에서 실행시간은 제안하는 알고리듬이 3-opt보다 약 10~30배 빠르게 나타나고 있다. 또한 제안하는 알고리듬과 3-opt의 실행시간의 차이는 문제에 따라 다르게 나타난다. 문제가 다를 경우 해는 제안하는 알고리듬이 적을 때도 있고 3-opt가 적을 때도 있다. 따라서, 해의 정확도에서는 어느 알고리듬이 다른 알고리듬보다 좋다고 말하기는 어렵다.

표 5.3 3-opt와 제안하는 알고리듬의 비교

문제번호 \ 구분	시간		해		
	3-opt	제안하는 알고리듬	3-opt	제안하는 알고리듬	최적 알고리듬
br17	0.001	0.0	39	39	39
ftv44	0.06	0.006	1672	1692	1613
ft70	0.33	0.08	44737	44783	38673
ftv70	0.27	0.02	2060	2066	1950
kro124p	3.16	0.075	38978	38555	36230
ftv170	13.38	0.35	3085	3075	2755

5.2.5 비용행렬의 삭감여부에 따른 해의 비교

표 5.4는 제안하는 알고리듬에서 비용행렬을 삭감했을 때와 그렇지 않을 경우에 대해 해를 비교한 결과로써 삭감에 대한 효과를 보여준다. α 값은 0.1로 설정하였다.

삭감에 대한 효과를 보여준다. ftv170만을 제외하고 비용행렬을 삭감했을 때가 그렇지 않을 경우보다 해가 더 좋게 나타난다. 따라서, 문제별로 결과가 다르게 나타날 수 있지만 일반적으로 비용행렬을 삭감했을 경우가 그렇지 않을 경우보다 해가 더 좋아진다.

표 5.4 비용행렬의 삭감여부에 따른 해의 비교

문제번호 \ 구분	비용행렬 삭감할 경우	비용행렬을 삭감하지 않을 경우
br17	39	39
ftv44	1692	1701
ft70	44783	44931
ftv70	2066	2093
kro124p	38555	38787
ftv170	3075	3065

5.2.6 제안하는 알고리듬의 해의 개선모습

제안하는 알고리듬이 빠른 시간 내에 국지탐색을 하는 이유는 선별한 호 즉 적은 비용의 호들만을 삽입을 시도함으로써 이웃 순환로로 개선되는 과정에서 3-opt에 비해 적은 개선횟수로 많은 비용을 개선하기 때문이다. 즉, 제안하는 알고리듬은 빠른 시간 내에 초기 순환로를 비용이 적은 호들로 구성된 순환로로 만든다. 따라서 3-opt에 비해 개선횟수는 적지만 최종 순환로의 비용은 일반적으로 좋게 나타나고 있다. 그림 5.3은 제안하는 알고리듬과 3-opt가 해를 개선해 가는 모습을 보이고 있다. 그림 5.3에서 사용한 문제는 A2이며 하나의 국지탐색 예를 보이고 있다. 만일 가로축에 시간을 포함하면 많은 시간 차이를 두고 해가 개선되는 모습이 나타난다.

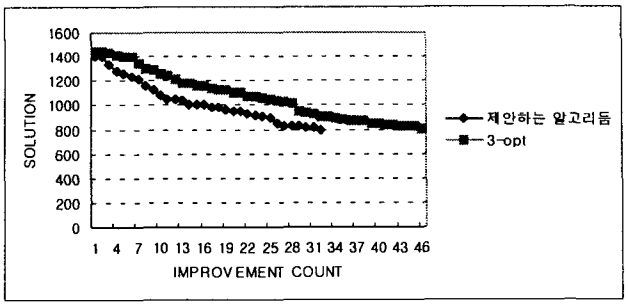


그림 5.3 제안하는 알고리듬과 3-opt의 개선모습

5.2.7 β를 포함한 실험결과

각각의 교점에서 나가는 호들의 비용을 크기 순으로 정렬하여 (β · 100)%에 해당하는 호들에 대하여 삽입을 시도할 수 있다(여기서 β는 0에서 1사이의 값임). 따라서 β는 모든 호에 대해 적용하는 α와는 달리 각 교점에서 나가는 호들에 대해 적용하는 파라미터이다. 표 5.5는 α만 적용할 경우와 α와 β를 동시에 적용할 경우의 실험결과이다. α는 0.1, β는 0.07로 설정하여 실험하였다.

표 5.5 α만 적용할 경우와 α, β둘 다 적용할 경우의 실험결과 (α=0.1, β=0.07)

문제번호 \ 구분	해의 비교		실행시간의 비교 (단위 : 초)	
	α적용	α, β적용	α적용	α, β적용
A1	315	314	0.01	0.01
A2	766	766	0.09	0.09
A3	1148	1148	0.344	0.346
A4	1550	1550	0.79	0.79
br17	39	39	0.0	0.0
ftv44	1692	1666	0.006	0.007
ft70	44783	44749	0.08	0.116
ftv70	2066	2065	0.02	0.03
kro124p	38555	38543	0.075	0.088
ftv170	3075	3075	0.35	0.42

표 5.5에서 랜덤한 문제인 경우 α만 적용할 경우와 α, β 둘 다 적용할 경우는 해와 실행시간에서 차이를 발견하기 힘들다. 그 이유는 문제가 랜덤할 경우 α만 적용할 경우 삽입을 위한 호들과 α, β 둘 다 적용할 경우의 삽입을 위한 호들 사이에는 별 차이가 없기 때문이다. TSPLIB문제에 대해서는 해에서 개선을 보이고 있다. 시간이 더 걸려도 해의 개선을 요한다면 α, β 둘 다 적용하는 것이 효과적이다.

6. 결론

본 연구를 통해 제안하는 알고리듬이 3-opt에 비해 빠른 시간 내에 국지탐색 해를 구함을 알 수 있다. 이는 3-opt에서는 고려하지 않은 호의 선별과정을 통해 해를 개선시킬 수 있는 호들만을 삽입해 봄으로써 3-opt에 비해 해가 악화되지 않으면서 빠른 시간 내에 국지탐색 해를 구해 준다. 선별과정을 통해 선택된 호들 즉, 비용이 적은 호들에 대해 우선적으로 삽입을 시도함으로 많은 개선이 초기의 짧은 시간에 계속적으로 이루어진다.

해의 정확도를 비교하면 TSPLIP데이터인 경우 제안하는 알고리듬이 3-opt보다 좋다고 하

기는 어렵다. 데이터에 따라 제안하는 알고리듬이 좋은 경우와 그렇지 않는 경우가 발생한다. 그러나, 랜덤한 데이터인 경우 일반적으로 제안하는 알고리듬이 좋은 해를 보이고 있다.

실행시간을 비교할 경우 제안하는 알고리듬은 3-opt에 비해 약 8배 이상 빠르다. 데이터에 따라 결과가 다르게 나타나지만 일반적으로 8~30배 빠르게 나타나고 있다. 이는 α 값을 0.1로 설정하여 비용이 적은 호만을 삽입해 봄으로써 계산량이 그 만큼 적어지기 때문이다.

제안하는 알고리듬은 파라미터인 α 값을 조절함으로써 실행시간과 해를 조절할 수 있다. 지금까지의 실험결과 α 값이 0.1일 경우 효율적으로 빠른 시간 내에 국지탐색 해를 구한다. 일반적으로 α 값이 0.1보다 커지면 해가 더 이상 개선되지 않으며 개선되어도 그 정도는 미미하다. 그러나 α 값이 0.1보다 적을 경우 α 값을 0.1에서 0.15사이의 값으로 설정함으로써 해가 더 좋아질 수 있다.

참 고 문 헌

[1] 강맹규, *네트워크와 알고리듬*, 박영사, 서울, 1991.

[2] 강맹규, *자료구조*, 홍릉과학출판사, 서울, 1995.

[3] 이재승, 신해웅, 강맹규, "Prüfer수를 이용한 외판원문제의 유전해법," *공업경영학회*, 제20권, 제41집, pp. 1-14, 1997.

[4] Bellmore, M., and G. Nemhauser, "The Traveling Salesman Problem: A Survey," *Operations Research*, Vol. 16, pp. 538-558, 1968.

[5] Gendreau, M., A. Hertz, and G. Laporte, "New Insertion and Postoptimization Procedures for the Traveling Salesman Problem," *Operations Research*, Vol. 40, No 6, pp. 1086-1094, 1992.

[6] Reinelt, G., *The Traveling Salesman Computational Solutions for TSP Applications*, Spring-Verlag, Heidelberg, pp. 73-160, 1994.

[7] Golden, B., L. Bodin, T. Doyle, and W. Stewart, Jr., "Approximate Traveling Salesman Algorithms," *Operations Research*, Vol. 28, pp. 694-710, 1980.

[8] Kanellakis, P. C. and C. H. Papadimitriou, "Local Search for the Asymmetric Traveling Salesman Problem," *Operations Research*, Vol. 28, pp. 1086-1099, 1980.

[9] Lawer, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons, N. Y., 1985.

[10] Lin, S., "Computer Solution of Traveling Salesman Problem," *Bell System Tech. Journal*, Vol. 44, pp. 2245-2269, 1965.

[11] Lin, S. and B.W. Kernighan., "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, Vol. 21, pp. 498-516, 1973.

[12] Martin, O., S. W. Otto, and E. W. Felten, "Large-step Markov Chains for the TSP Incorporating Local Search Heuristics," *Operations Research Letters*, Vol. 11, pp. 219-224, 1992.

[13] Phillips, Don T. and A. Garcia-Diaz, *Fundamentals of Network Analysis*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[14] Sakakibara, K., "New Edges not Used in Shortest Tours of TSP," *European Journal of Operational Research*, Vol. 104, pp. 129-138, 1998.

[15] Voudouris, C. and E. Tsang., "Guided Local Search and Its Application to the Traveling Salesman Problem," *European Journal of Operational Research*, Vol. 113, pp. 469-499, 1999.