한국외국어대학교 컴퓨터공학부
HANKUK UNIVERSITY OF FOREIGN STUDIES

개미 군체 알고리즘(ACO)을 이용한 TSP 구현

Solving the TSP(Traveling Salesman Problem) using ACO(Ant Colony Optimization)

황가은 (Ga-Eun Hwang) [1]

[1] 한국외국어대학교, 컴퓨터공학부; idhge12@gmail.com

한글 요약: NP-Hard 문제라고 불리는 외판원 문제(TSP)를 전역최적화를 위한 근사해법인 메타 휴리스틱 중 개미 군체 알고리즘(ACO)와 그의 변형 알고리즘인 EAS, RAS, MMAS 을 사용하여 코드로 구현하고 해를 구한다.

핵심어: NP-Hard, 외판원 문제, 개미 군체 알고리즘, 메타 휴리스틱

영문 요약: Traveling Salesman Problem(TSP), which is a well-known the NP-Hard problem, is implemented using Ant Colony Optimization(ACO) and its variant algorithms, EAS(Elitist Ant System), RAS(Rank Based Ant System), and MMAS(Min Max Ant System) which are approximate solutions for the global optimization.

Keywords: NP-Hard, TSP, ACO, Meta Heuristics, EAS, RAS, MMAS

본 논문은 2023 학년 2 월에 제출된 한국외국어대학교 컴퓨터공학부 졸업논문이다. 2023.xx.xx

지도교수: 신찬수

서명: _____

1. 서론 - TSP

NP-Hard 문제로 분류되는 외판원 문제(TSP)는 여러 지점들이 존재하고 한 지점에서 다른 지점으로 이동하는 비용을 구하여, 모든 지점들을 단 한 번만 방문하여 원래 시작했던 지점으로 돌아오는 최소 비용의 이동 순서를 구하는 것이다. 가야할 지점이 많을수록 한 지점 당 갈 수 있는 지점이 증가하므로 시간은 지수 배로 증가가 된다. 전역해를 찾기까지의 시간이 매우 오래 걸리므로 전역 최적화 기법인 메타 휴리스틱을 사용하여 전역 최적해를 구한다. 즉, 완벽한 정답은 아닐지라도 정해진 시간 혹은 반복 횟수에 따라 그에 가까운 최적의 해를 구한다는 의미이다. 메타 휴리스틱 기법 중 하나인 유전 알고리즘(GA)를 이용하여 VRP(Vehicle Route Problem)의 전역최적해를 찾는 코드를 구현한 경험이 있어 다른 알고리즘인 개미 군체 알고리즘(ACO)과 그의 변형 알고리즘(EAS, RAS, MMAS)을 이용하여 TSP 를 구현하게 되었다.

개미 군체 알고리즘(ACO)는 먹이를 찾는 개미의 집단 행동으로부터 영감을 받아 1992 년 Marco Dorigo 의 논문[1]에서 제안되었다. 해당 논문에서는 TSP 에 어떻게 ACO 를 적용하는지에 대한 방법과 응용분야에 대해 서술되어 있다. 개미는 지표면 아래에서 주로 생활하기 때문에 시각적인 부분에서

발달하지 못했다. 그 대신 다른 곤충보다 화학 신호가 많이 발달하였고 더듬이로 냄새의 강도나 방향에 대한 정보를 전할 수 있다. 지표에 페로몬을 남겨 다른 개미들이 따라올 수 있게끔 한다. 개미들이 많이 지나다니는 경로일수록 페로몬의 농도가 짙으며 경로를 찾는 행위가 반복될수록 최선의 경로를 발견하게 된다. 이러한 자연현상을 코드로 옮겨 주어진 유한 개의 지점에 대해 최적의 경로를 구한다.

EAS, RAS, MMAS 는 ACO 의 변형 알고리즘이다. EAS 는 페로몬을 추가하는 방법에 엘리티스트 전략을 도입한 것이다. 엘리티스트 전략은 가장 좋은 해에 추가적으로 페로몬을 분비함으로써 해당 해가 포함된 해를 생산하도록 유도한다. RAS 는 순위 기반 개미 시스템이며 EAS 처럼 페로몬을 추가하는 방법에 변화를 준다. RAS 에서 페로몬 추가는 현재의 반복에서 찾은 N 개의 가장 좋은 경로들의 순위를 매겨 순위에 따라 페로몬을 추가를 한다. MMAS 는 최대 최소 개미 시스템이다. MMAS 는 반복 최선해와 전역 최선해를 가진다. 또한 최대 페로몬 양과 최소 페로몬 양을 가지고 반복 최선해 갱신인지 전역 최선해 갱신인지에 따라 추가하는 페로몬 양이 다르다.

해당 논문은 코드의 구성, 자료구조를 주로 언급하여 어떻게 코드로 구현했는지 중점으로 나열한다.

2. 연구 방법 및 결과

코드는 라이브러리 사용이 편리한 Python 으로 구현했다. 문제 해결에 필요한 지점(Node), 경로(Path), 개미(Ant) 클래스와 이들을 다 묶어 클래스에 접근하기 쉽도록 자연현상이라는 개념으로 ACO 클래스를 만들었다.

전체적인 코드의 개요는 다음과 같다.

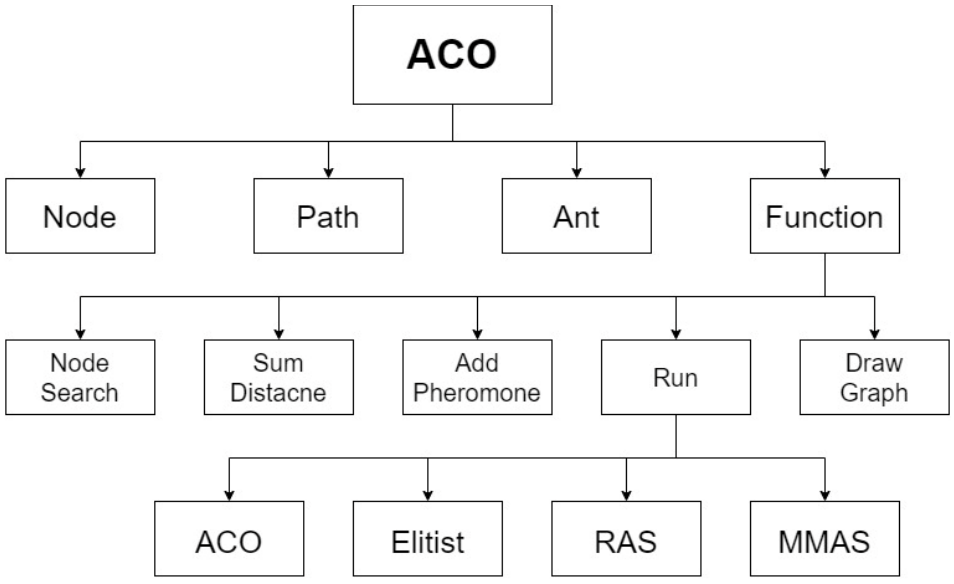


그림 1. 코드의 개요

2.1. ACO

1. Node

```
class Node:
    def __init__(self, lat_long, num):
        self.position = lat_long
        self.pos_number = num
```

노드는 위치와 지점 번호(pos_number) 속성을 가진다. 위치는 다음 코드처럼

nodes_1 = [(random.uniform(-90, 90), random.uniform(-180, 180)) for i in range(0, N)]

랜덤으로 위도와 경도를 튜플 형태로 N 개를 생성한다. 지점 번호를 만들어 놓은 이유는 나중에 인덱스를 효율적으로 이용하기 위함과 데이터를 확인할 때 직관적으로 보기 위해서이다.

2. Path

```
class Path:
    def __init__(self,   path_distance, initial_pheromone):
        self.path_distance = path_distance
        self.pheromone = initial_pheromone
```

경로(Path) 클래스는 지점과 지점 사이의 거리인 path_distance 속성과 경로가 가지고 있는 페로몬 정보를 가진다.

3. Ant

```
class Ant:
    def __init__(self, num_nodes, paths, ID):
        self.num_nodes = num_nodes
        self.paths = paths
        self.visited_node = []
        self.unvisited_node = []
        self.visited_dist = 0
        self.id = ID
        self.to_add_pheromone = 0
```

개미 클래스는 각 개미들이 경로를 순회하기 위해서 각자가 방문한 노드, 방문하지 않은 노드를 기록하기 위해서 visited_node, unvisited_node 리스트 속성을 가진다. 방문한 노드들의 거리를

저장하기 위해 visited_dist 를 만들었으며 여러 마리의 개미들을 직관적으로 보기 위해서 ID 속성을 부여해주었다. 또한 개미가 페로몬을 분비하므로 분비할 페로몬의 양을 계산하기 위해 to_add_pheromone 속성이 존재한다.

4. Function

이들을 크게 묶고 관리하기 위해서 만든 ACO 클래스의 함수는 다음과 같다.

I. 생성자

ACO 클래스는 자연현상이자 TSP 를 나타낸다. 이 클래스를 정의할 때 초기에 필요한 시작 지점의 숫자, 군체 크기, 페로몬이 쌓이는 양, 초기 페로몬, 반복 횟수, 자연 증발 페로몬 양, 지점 리스트를 인자로 받는다.

인자로 받은 변수들을 자신의 클래스 속성으로 만들고 속성이 될 노드 리스트는 다음과 같이 직관적으로 보기 편하도록 만들었다.

```
self.nodes = [self.Node(nodes[i], i) for i in range(len(nodes))]
```

이렇게 만들어줄 경우 튜플 안에 지점의 위치 정보와 구분하기 위한 숫자 정보가 같이 들어가 직관적으로 볼 수 있다는 장점이 있다.

각 지점 간의 거리를 계산할 때 자기 자신으로 가는 경로는 없음(None)을 가정했다.

```
self.paths = [[None] * self.num_nodes for _ in range(self.num_nodes)]
```

Paths 리스트 안에는 Path 클래스가 지점과 지점사이의 거리를 haversine 으로 계산하여 초기 페로몬 값과 함께 정의한다.

```
self.ants = [self.Ant(self.num_nodes, self.paths, i) for i in range(self.colony_size)]
```

개미는 미리 정의해둔 군체 사이즈만큼 생성하며 여러 마리의 개미들을 구분하고 관리하기 위해서 ID 속성을 포함하고 있는데 이 ID 에 i 가 들어간다.

n 번의 반복 중에서 모든 개미들이 경로 탐색을 마치고 돌아오는 n 번의 주기마다 가장 짧은 거리와 노드 순서를 기록하기 위해 best_global_distance, best_global_tour 를 정의했다.

II. 지점 탐색

지점 탐색은 모든 개미들의 visited_node 와 unvisited_node 를 갱신시킨다. 개미의 visited_node 에 지점이 없다면 start_node(시작 지점)을 모든 개미에게 넣어주고 다음 반복을 진행한다. 지점이 N 개라면 모든 개미의 visited_node 에 N+1 만큼의 노드가 들어가도록 한다. N+1 인 이유는 시작 지점과 마지막 지점이 들어가기 때문에 총 5 개의 지점이 있다면 0, 1, 2, 3, 4, 0 인 형식이 된다.

다음에 갈 지점을 탐색하는 과정에서 unvisited_node 와 visited_node 를 참고한다. 개미가 마지막으로 간 지점에서 다음에 갈 수 있는 지점을 탐색해야 한다. 그러기 위해서 다음과 같이 페로몬 리스트를 만든다.

```
pheromone_list = [k.pheromone for k in self.ants[i].paths[self.ants[i].visited_node[-1].pos_number] if k != None]
```

각 개미는 각자의 경로 정보를 가지고 있다. 개미가 가진 경로 중에서 마지막으로 방문한 노드에서 갈 수 있는 경로를 찾아 해당 경로의 페로몬 정보를 모아 리스트로 만든다. 이때 자기 자신으로 가는 경로는 None 으로 처리했으므로 None 이 들어가지 않도록 해야한다. 페로몬 리스트를 만든 이유는 각 경로가 가진 페로몬을 이용하여 확률적 랜덤 뽑기를 하기 위함이다. 이 페로몬 리스트를 가지고 확률 리스트를 만들고 확률에 비례하여 랜덤으로 다음 지점을 뽑는다.

```
prob = [i / sum(pheromone_list) for i in pheromone_list]

next_node = np.random.choice(self.ants[i].unvisited_node, size = 1, p = prob)[0]
```

prob 은 확률의 특성 상 합이 1 이 되어야 한다. 그러므로 기존의 Pheromone 값을 pheromone 의 합으로 나눴다. Unvisited_node 에서 prob 의 확률에 따라 1 개의 지점을 뽑는다. 그 다음에는 next_node 를 visited_node 에 추가해준다. 루프를 돌 때마다 unvisited_node 는 초기화를 해주므로 따로 unvisited_node 에서 제거하는 과정은 거치지 않았다. 또한 next_node 이전의 마지막 지점에 대한 모든 경로를 None 으로 설정하여 닫아준다. 이미 갔던 지점을 가지 않고 닫힌 지점에서 갈 수 있는 경로도 차단을 하여 오류가 나지 않도록 하기 위함이다.

III. 개미가 다녀간 노드 간의 거리 합

ACO 뿐 아니라 ACO 의 변형 알고리즘인 EAS(Elitist Ant System), RAS(Rank Based Ant Colony), MMAS(Min Max Ant Colony)에 대해서도 구현을 같이 했기 때문에 지점 간의 거리 합을 계산하는 방법이 조금씩 다르다.

기본적으로 visited_node 에 따라서 해당 거리를 더하는 것은 동일하다. RAS 의 경우에는 순위를 매기기 위해서 ras_tour_dist 라는 리스트 속성을 만들어 페로몬의 순서대로 정렬을 시켰다. 각 개미들의 경로를 돌고 거리를 더하면서 현재 설정된 global_best_distance 보다 짧다면 최선해를 갱신한다.

IV. 페로몬 분비

페로몬 분비에 대해서는 EAS 와 RAS 를 고려한다. MMAS 같은 경우는 따로 함수를 만들었다. 분비할 페로몬 양은 다음과 같다.

```
self.ants[i].to_add_pheromone = (self.pheromone_deposit_weight /
self.ants[i].visited_dist) * 1000
```

방문한 노드와 그 다음 노드의 페로몬에 to_add_pheromone 만큼 페로몬을 추가하고 RAS 같은 경우에는 순위를 매긴 리스트를 사용하여 해당 위치의 인덱스에 0.5 를 곱한 페로몬을 다시 추가해주었다.

```
self.paths[self.ants[i].visited_node[j].pos_number][self.ants[i].vis-
ited_node[j+1].pos_number].pheromone +=  (self.ras_tour_dist.i
ndex((self.ants[i], self.paths[self.ants[i].visited_node[j].pos_num-
ber][self.ants[i].visited_node[j+1].pos_number])) * 0.5)
```

EAS 의 경우에는 최선해에 다시 페로몬을 추가했다.

MMAS 는 거리를 계산하는 것과 페로몬을 분비하는 것을 합친 함수를 만들었다. 최소 페로몬과 최대 페로몬이 존재하므로 정의를 해주었고 각 개미 마다 갱신하는 반복최선해가 있고 루프 마다 갱신하는 전역최선해가 있다. 두 개의 최선해를 혼합하여 사용한다. 그 이유는 EAS 와 RAS 에서는 지난 반복에서 찾은 최선해 주위의 탐색을 강화하는 전략을 사용한다. 이러한 최선해를 강화한 이후의 탐색은 조기에 해가 정체되는 문제를 야기시킨다. 좋은 최선해를 찾기 위해서는 좋은 최선해를 이용하되 조기 정체를 피할 수 있어야 하기 위함이다. 반복 최선해의 경우에는 전역 최선해보다 더 적은 페로몬을 추가한다. 2 번 페로몬을 추가하는 과정에서 특정 해의 페로몬 양이 과도하게 많아지면 다양한 탐색이 불가능해지고 조기 정체를 야기하기 때문에 이전에 설정한 max_pheromone 보다 많아진다면 max_pheromone 으로 변경해준다. 또한 자연에서 페로몬은 계속 증발하기 때문에 min_pheromone 보다 낮다면 min_pheromone 으로 변경해준다.

V. 실행 함수

각 알고리즘마다 실행함수를 생성했다. 기본적인 양식은 같으며 사용하는 함수와 인자가 조금씩 다르다.

전체적인 구조는 다음과 같다. 지점 탐색, 거리 계산, 페로몬 추가 함수를 정해진 횟수만큼 반복하여 전역 최선해를 구한다. 또한 해당 알고리즘으로 계산하여 나온 결과를 표시한다. 결과에는 시작 지점과 초기 경로 순서, 최선해의 경로 순서, 초기 거리의 총합, 최선해의 거리 총합, 초기 거리 대비 효율 감소량, 페로몬의 총합이 포함되어 있다.

VI. 그래프 표현

ACO, EAS, RAS, MMAS 알고리즘으로 나온 전역 최선해를 그래프로 시각화 한다. 지점들을 그래프 상에 나타내고 그 지점들을 순서에 맞게 잇는다. 알아보기 쉽게 하기 위해 시작 지점이자 끝 지점의 색을 다르게 했다.

3. 결과 분석

예를 들어 10 개의 지점에 대해 25 마리의 개미가 100 번 반복하여 경로를 탐색한 결과와 사용된 테스트 데이터의 정보(지점의 위치, 지점사이의 거리 정보)는 다음과 같다.

인덱스	0	1	2	3	4	5	6	7	8	9
위도	-24.872	-66.735	16.654	64.109	59.254	-73.551	-51.668	-83.522	-15.885	57.991
경도	-100.69	76.115	-11.346	-55.778	73.228	38.590	47.221	-131.69	69.113	-22.081

표 1. 사용된 데이터 정보

Km	0	1	2	3	4	5	6	7	8	9
0	None	9825.27	10714.07	10631.98	16162.64	8668.27	10946.35	6630.07	15350.28	11694.57
1	9825.27	None	11594.61	17827.91	14011.49	1569.98	2307.45	3240.52	5677.48	16010.48
2	10714.07	11594.61	None	6247.13	8115.56	10647.46	9465.04	12213.56	9533.85	4681.51
3	10631.98	17827.91	6247.13	None	5643.32	16759.19	15572.99	16887.43	13244.38	1909.54
4	16162.64	14011.49	8115.56	5643.32	None	14994.01	12554.65	17234.62	8363.59	5034.35
5	8668.27	1569.98	10647.46	16759.19	14994.01	None	2467.17	2541.76	6692.92	15312.69
6	10946.35	2307.45	9465.04	15572.99	12554.65	2467.17	None	4982.48	4426.15	13710.06
7	6630.07	3240.52	12213.56	16887.43	17234.62	2541.76	4982.48	None	8915.74	16636.16
8	15350.28	5677.48	9533.85	13244.38	8363.59	6692.92	4426.15	8915.74	None	11569.68
9	11694.57	16010.48	4681.51	1909.54	5034.35	15312.69	13710.06	16636.16	11569.68	None

표 2. 지점 간 거리 정보

0 지점에서 1 지점으로 가는 거리는 1 지점에서 0 지점으로 가는 거리와 같다.

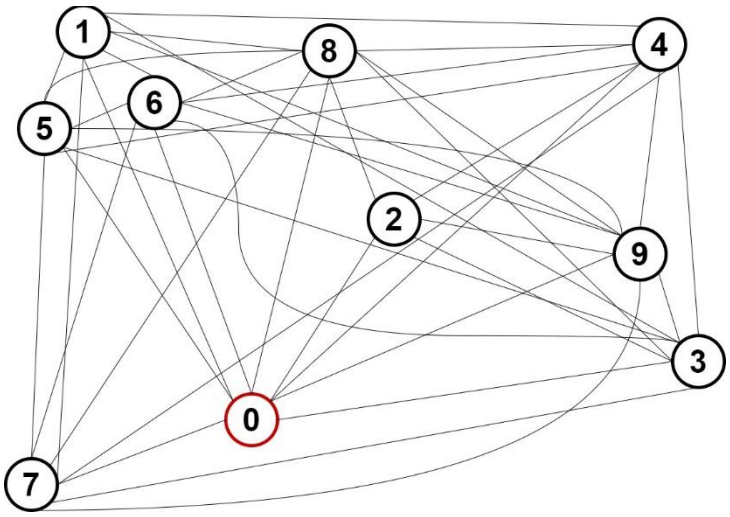


그림 2. 사용된 데이터 위치 정보

```
<- ACO 시작 -> 100회 반복
--------------------------------------------------------------------------------

<결과>

시작 지점: 0
순서 : 0 -> 2 -> 3 -> 9 -> 4 -> 8 -> 1 -> 5 -> 6 -> 7 -> 0
초기 순서 : 0 -> 8 -> 1 -> 7 -> 4 -> 9 -> 2 -> 5 -> 3 -> 6 -> 0
거리 총합 : 53595.848 km
초기 거리 총합 : 105144.801 km
초기 거리 대비 효율: 49% 감소
페로몬 총합 : 143.336

--------------------------------------------------------------------------------
```

그림 3. ACO

```
<- ACO ELITIST 시작 -> 100회 반복
--------------------------------------------------------------------------------

<결과>

시작 지점: 0
순서 : 0 -> 7 -> 5 -> 1 -> 6 -> 8 -> 4 -> 9 -> 3 -> 2 -> 0
초기 순서 : 0 -> 2 -> 7 -> 6 -> 5 -> 1 -> 8 -> 4 -> 9 -> 3 -> 0
거리 총합 : 49744.147 km
초기 거리 총합 : 63564.194 km
초기 거리 대비 효율: 22% 감소
페로몬 총합 : 1070.108

--------------------------------------------------------------------------------
```

그림 4. EAS

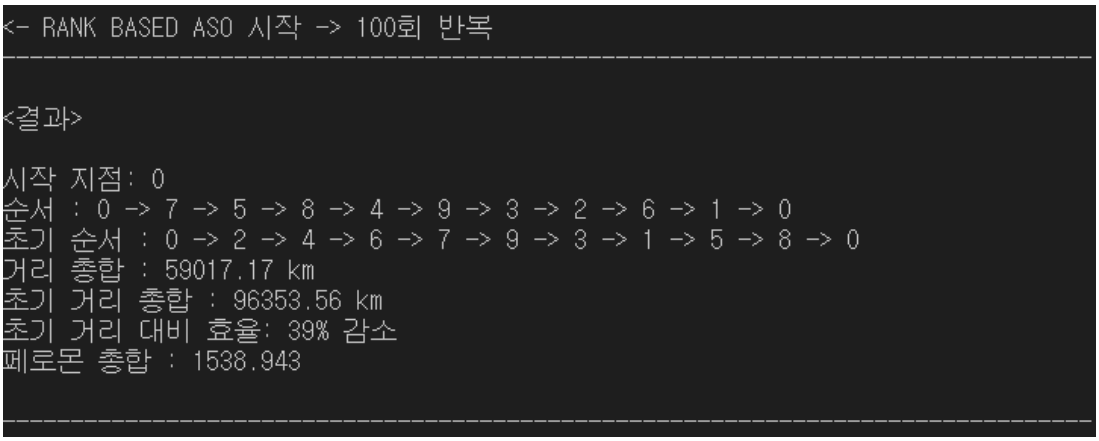


그림 5. RAS

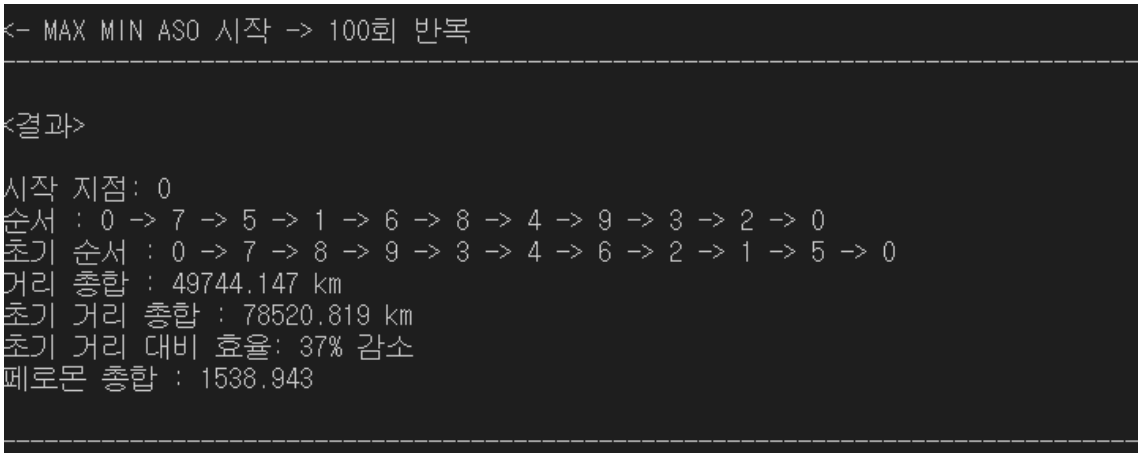


그림 6. MMAS

그림 7. 성능비교

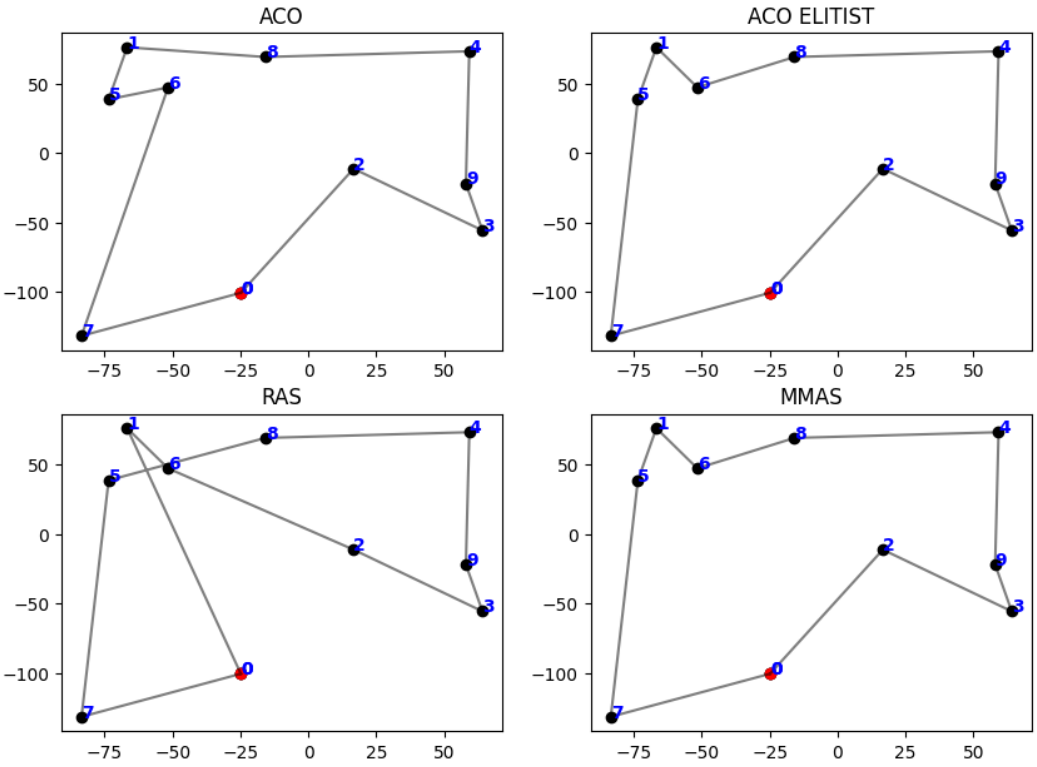


그림 8. 그래프 표현

해당 실행 결과를 표를 정리하면 다음과 같다.

알고리즘	수행시간 순서	경로가 짧은 순서	초기 거리 대비 효율 감소율
ACO	1	3	49%
EAS	2	1	22%
RAS	3	4	39%
MMAS	4	2	37%

표 3. 알고리즘 별 결과 비교

개미 군체 사이즈가 20, 반복 횟수는 100, 시작 지점과 끝 지점은 0, 증발하는 페로몬 양 5, 초기 페로몬 1, 페로몬이 쌓이는 정도 10 의 조건으로 10 번 반복하여 통계한 결과는 다음과 같다.

알고리즘	평균 수행시간(sec)	수행 시간 순위 평균	거리 순위 평균	감소량 평균
ACO	1.9306	2.5454	2.5454	38.8333
EAS	1.9621	2.1818	2	35.4166
RAS	1.9953	3	2.6363	35.75
MMAS	1.9398	2.7272	2.8181	24.4166

표 4. 10 번 반복 후 알고리즘 별 결과 비교

평균 수행시간을 보면 4 개의 알고리즘이 큰 차이가 나지 않는다. 그 중에서도 ACO, MMAS, EAS, RAS 순서로 수행 시간이 빠르다. 수행 시간을 기준으로 순위를 매겼을 때 순위 평균을 보자면 EAS, ACO, MMAS, RAS 순으로 낮은 순위를 가질 확률이 높다. 거리를 기준으로 순위를 매겼을 때 순위 평균을 보자면 EAS, ACO, RAS, MMAS 순으로 상대적으로 더 긴 거리를 가질 확률이 높다. 감소량 평균이 가장 차이가 많이 나는 부분이다. MMAS 의 감소량 평균은 24.4166 인 반면 나머지 ACO, EAS, RAS 는 38.8333, 35.4166, 35.75 로 확연한 차이를 보인다. MMAS 가 다른 알고리즘에 비해 감소량 평균이 현저히 낮은 이유는 다음과 같이 설명할 수 있다. MMAS 에서는 반복최선해와 전역최선해 2 개를 사용한다. 페로몬 추가도 공통적으로 하지만 반복최선해의 경우에는 전역최선해보다 낮은 양의 페로몬을 추가한다. 그리고 탐색 과정 중 조기 정체를 막기 위해 최소 페로몬, 최대 페로몬 양을 이용하여 경로의 페로몬 양을 일정한 간격 아래 있도록 조정했다. 이는 다양한 해를 탐색하기 위한 목적도 있다. CPU 로 돌아가는 환경이므로 N 이 작다. 그러므로 작은 N 의 범위에서 2 개의 최선해를 사용하고 더 다양한 해를 탐색하기 위해 페로몬의 범위를 조정했기 때문에 페로몬을 추가하고 증발되는 과정에서 좋지 않은 해가 확실하게 걸러지지 않게 된다. 다른 알고리즘 같은 경우는 같은 N 의 범위 안에서도 MMAS 만큼 다양한 해의 탐색을 목적으로 하지 않고 전역최선해의 이용 강화를 통해 해를 찾는다. 또한 페로몬 양을 일정한 범위 아래에서 조정하지 않으므로 품질이 좋지 않은 해가 확실하게 걸러진다. 이런 이유로 좁은 범위에서 전역 최선해의 갱신 품질이 좋지 않아 감소량이 다른 알고리즘에 비해 낮다.

만약 N 의 크기가 100, 200 이상으로 컸다면 ACO, EAS, RAS 와 같은 전역 최선해를 강화하여 해를 찾는 알고리즘은 조기 정체 때문에 지역최선해를 찾을 가능성이 높고 그에 반해 MMAS 는 local optima 에 빠지지 않아 전역 최선해를 찾을 가능성이 높다.

그래프 모형을 확인했을 때 지점의 개수가 10 개 이전으로는 4 개의 모형이 거의 비슷했으나 지점의 개의 개수가 15 개가 넘어간 후부터는 조금씩 차이를 보인다. 더 다양한 비교를 위해서 지점의 개수를 5 개부터 100 개까지 위의 군체 25 마리, 반복 횟수 100 회인 경우와 군체 50 마리, 반복 횟수 200 회인 경우로 테스트를 하여 수행 시간과 초기에 설정된 거리에 대비하여 어느정도 감소율이 있는지에 대해 표로 정리하고 시각화 했다.

	5	10	15	20	25	30	35	40	50	70	90	100
ACO	0.4806	1.471	2.903	5.034	7.332	10.302	13.71	17.632	32.138	55.439	88.105	116.528
EAS	0.468	1.508	2.849	5.012	7.256	10.18	13.627	17.6	32.517	53.387	91.278	116.567
RAS	0.473	1.645	2.845	5.739	7.322	10.383	13.742	17.82	31.629	53.754	94.04	112.357
MMAS	0.462	1.686	2.828	5.698	7.329	10.244	13.624	19.259	28.568	54.05	100.484	112.763

표 5. 군체 25 마리, 반복 횟수 100 회의 수행 시간

	5	10	15	20	25	30	35	40	50	70	90	100
ACO	0%	49%	42%	32%	34%	24%	25%	4%	11%	22%	12%	9%
EAS	6%	22%	33%	28%	12%	21%	15%	28%	19%	16%	21%	19%
RAS	2%	39%	24%	23%	12%	25%	19%	19%	4%	13%	14%	12%
MMAS	0%	37%	27%	21%	27%	16%	18%	11%	9%	9%	5%	6%

표 6. 군체 25 마리, 반복 횟수 100 회의 초기 거리 대비 감소율

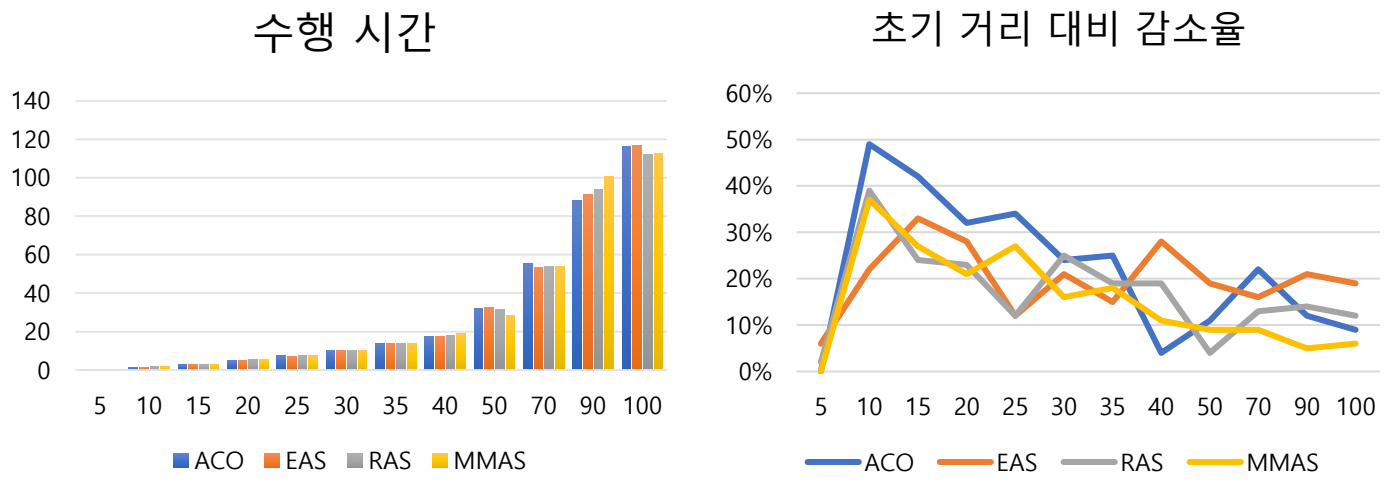


그림 9. 군체 25 마리, 반복 횟수 100 회 시각화

다음은 군체 50 마리, 반복 횟수를 200 회로 늘려 테스트한 결과이다.

	5	10	15	20	25	30	35	40	50	70	90	100
ACO	2.190	7.003	14.346	25.999	56.551	51.180	81.968	107.11	171.72	327.01	496.03	579.81
EAS	2.168	6.964	14.793	25.116	36.463	54.007	83.217	111.50	169.87	329.32	454.50	563.43
RAS	2.206	7.106	14.887	25.621	36.874	52.120	74.934	103.80	149.75	295.71	462.79	569.04
MMAS	2.163	7.164	14.905	28.1557	36.701	54.352	77.95	95.680	153.74	289.93	456.4	564.99

표 7. 군체 50 마리, 반복 횟수 200 회의 수행 시간

	5	10	15	20	25	30	35	40	50	70	90	100
ACO	21%	46%	11%	28%	38%	33%	26%	11%	21%	10%	16%	17%
EAS	1%	18%	34%	21%	27%	33%	26%	28%	25%	16%	18%	21%
RAS	5%	37%	40%	26%	21%	34%	23%	15%	23%	24%	12%	12%
MMAS	0%	32%	21%	26%	15%	17%	17%	12%	12%	8%	11%	7%

표 7. 군체 50 마리, 반복 횟수 200 회의 초기 거리 대비 효율

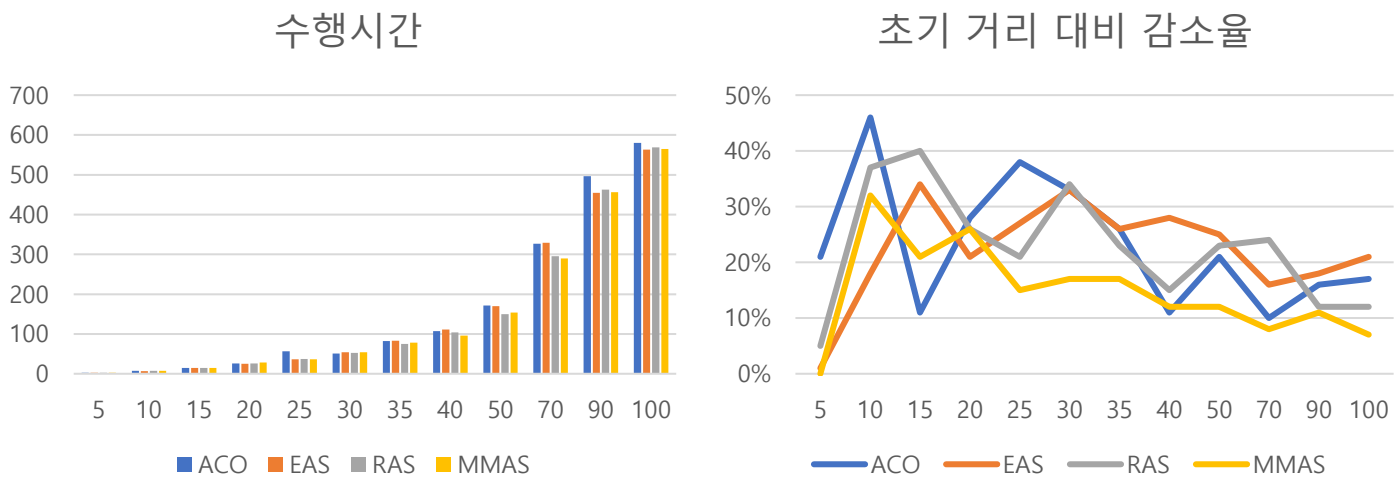


그림 10. 군체 50 마리, 반복 횟수 200 회 시각화

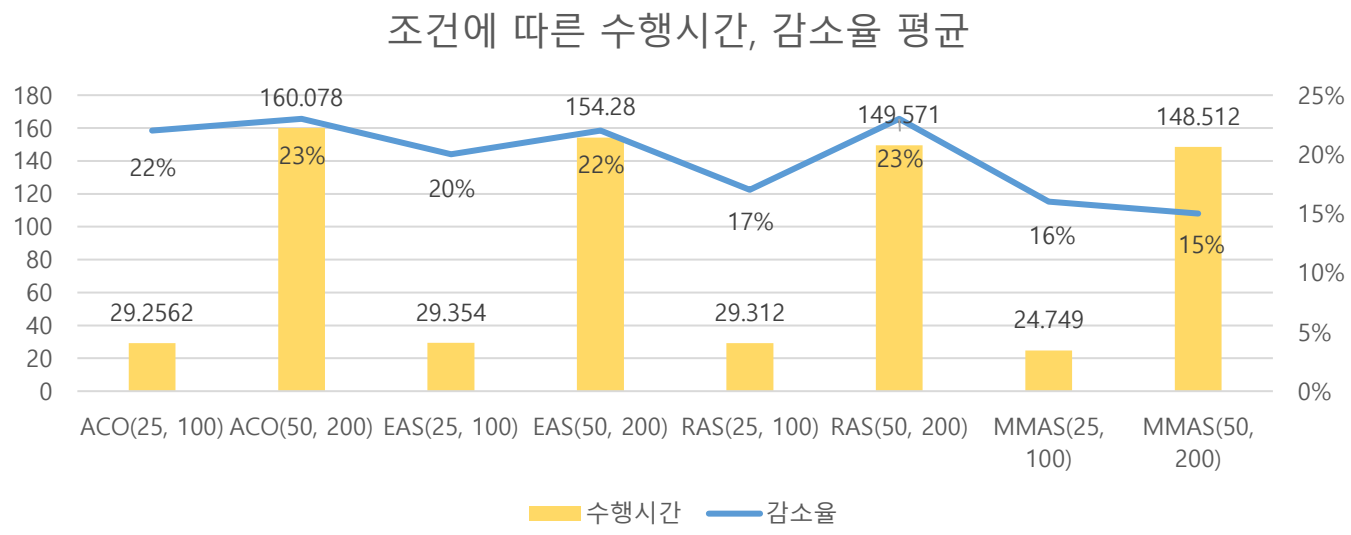


그림 11. 수행시간과 감소율의 평균

위의 그래프로 전체적인 양상을 보도록 하겠다. 먼저 군체 25 마리, 반복횟수가 100 회인 경우에 4 개의 알고리즘의 수행시간의 차이는 거의 없으나 24.749 초인 MMAS 가 그 중에서 가장 빠르다. 감소율은 수행시간과 반대로 MMAS 가 16%로 가장 작은 감소율을 보인다. 군체 50 마리, 반복횟수가 200 화인 경우에는 군체와 반복 횟수가 증가함으로써 수행시간이 평균 28.1678 초에서 153.11 초로 5 배 이상으로 늘어났음을 볼 수 있으나 감소율은 평균 19%에서 21%로 2%의 작은 증가폭을 보인다.

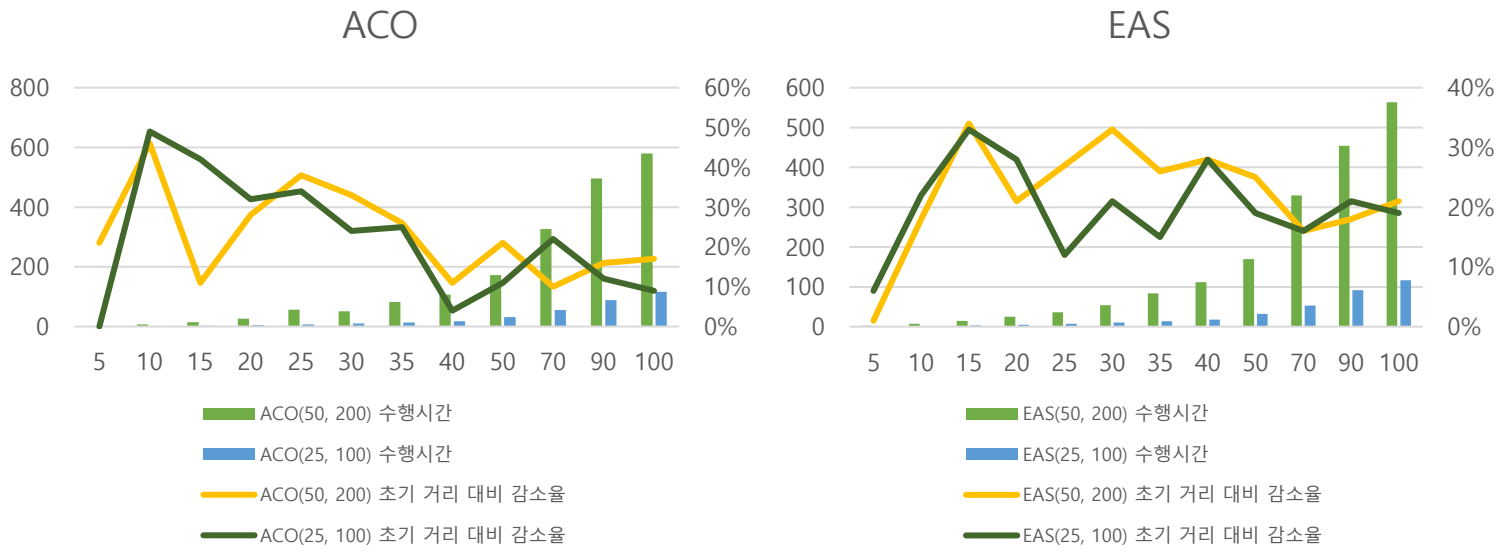
ACO, EAS, RAS, MMAS 를 조건과 상관없이 수행시간과 감소율을 평균으로 나타낸 결과는 다음과 같다.

	수행시간	감소율	수행시간 평균	감소율 평균
ACO(25, 100)	29.2562	22%	94.667	23%
ACO(50, 200)	160.078	23%		
EAS(25, 100)	29.354	20%	91.817	21%
EAS(50, 200)	154.28	22%		
RAS(25, 100)	29.312	17%	89.441	20%
RAS(50, 200)	149.571	23%		
MMAS(25, 100)	24.749	16%	86.63	16%
MMAS(50, 200)	148.512	15%		

표 8. 알고리즘 별 수행시간과 감소율의 평균

알고리즘 종류마다 평균을 내어본 결과 수행시간에 있어서 MMAS 가 86.63 초로 가장 빠르고 94.667 초인 ACO 가 가장 느리다는 것을 알 수 있다. 감소율 평균 또한 MMAS 가 가장 낮고 ACO 가 가장 높다. 평균 수치로 보면 수행시간이 느릴수록 감소율이 높아진다.

조금 더 자세히 살펴보기 위해 조건이 달라짐에 따라 알고리즘의 성능이 어떻게 변화하는지 보기 위해서 그래프로 시각화 했다.



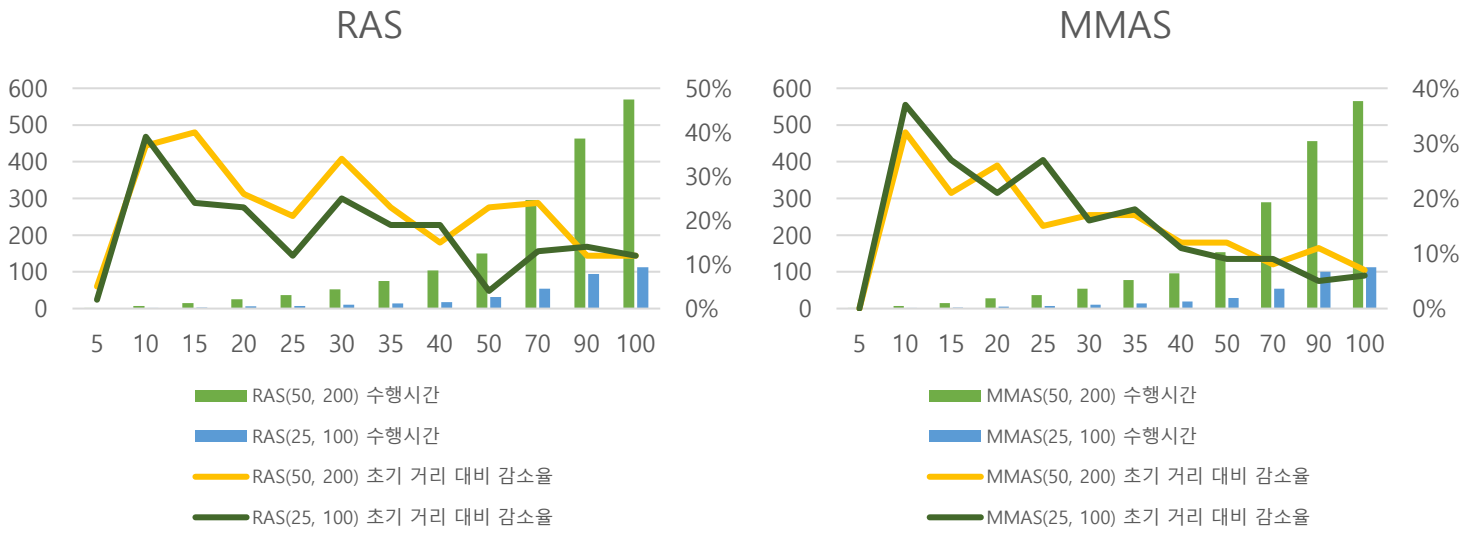


그림 12. 조건에 따른 ACO, EAS, RAS, MMAS 결과

그림 9, 10, 12 를 참고하여 보면 지점의 개수가 낮을 때엔 ACO, EAS, RAS, MMAS 모두 높은 감소율을 보여주었으나 지점의 개수가 많아질수록 초기와 비교하여 낮은 감소율을 보인다. 지점의 개수가 많아질수록 낮은 감소율을 보이는 이유가 증가한 지점에 비례하여 그만큼 경로가 많아졌는데 군체의 수와 반복횟수가 적어서 그러한 결과가 나온 것이라고 생각했으나 군체의 수와 반복횟수를 늘렸음에도 지점 개수가 많아질수록 낮은 감소율을 보이는 것은 동일했다. 해당 실험결과로 감소율은 지점의 개수와 반비례한다는 것을 볼 수 있다. 표 8 과 그림 11, 12 를 참고하여 그래프의 모양이 가장 차이가 나는 알고리즘은 RAS 이다. 군체 25 마리, 반복횟수 100 회인 RAS 의 감소율은 17%이며 군체 50 마리, 반복횟수 200 회인 RAS 의 감소율은 23%로 5%가 차이가 난다. 그와 반대로 가장 차이가 적은 알고리즘은 각각 16%, 15% 인 MMAS 이다.

그렇다면 어떤 알고리즘이 실험결과를 바탕으로 가장 좋은 성능을 지니고 있는지 알아보겠다.

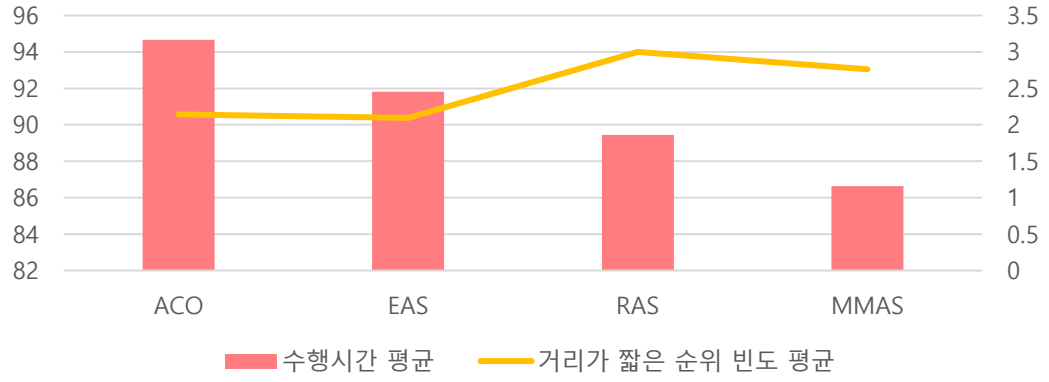


그림 13. ACO, EAS, RAS, MMAS 수행시간, 짧은 거리 순위 빈도 평균

성능을 알아보기 위해 알고리즘의 평균 수행시간과 평균 감소율, 짧은 거리 순위 빈도 수 평균을 참고한다. 표 8, 그림 11, 13 을 통해 계산한다. 먼저 수행시간과 감소율의 관계를 나타내기 위해 $\frac{감소율}{수행시간}$ 을 사용하여 ACO 는 0.242957, EAS 0.228716, RAS 0.223611, MMAS 0.184695 로 나타낼 수 있다. 다음으로 수행시간과 짧은 거리가 나온 평균 빈도의 관계를 나타내기 위해 $\frac{짧은\ 거리\ 순위\ 빈도}{수행시간}$ 을 사용한다. 그 결과로 ACO 0.022627, EAS 0.022817, RAS 0.033542, MMAS 0.031882 로 나타낼 수 있다.

수행시간과 감소율의 관계에서는 ACO 가 가장 높고 수행시간에 대해 짧은 거리가 나올 확률이 높은 알고리즘은 RAS 다. 감소율은 초기의 경로가 어떻게 설정되었는지에 따라 차이가 있다. 그러므로 감소율보다 짧은 거리의 순위 빈도 평균에 더 중점을 둘 필요가 있다고 판단하여 수행시간과 감소율의 관계와 수행시간과 짧은 거리가 나온 평균 빈도 관계에서 계산한 결과 가장 성능이 좋은 알고리즘은 RAS 이다.

4. 결론 및 토론

여러가지의 메타휴리스틱 중 개미 군체 최적화 알고리즘과 그의 변형 알고리즘에 대해 알아보고 이를 이용하여 코드를 구현하고 최적해를 구했다. 그 결과 각 알고리즘의 수행시간, 감소율, 평균적인 순위 등에 대해 알 수 있었으며 기준을 정해 비교함으로써 현재 실험 결과를 바탕으로 한 가장 좋은 성능을 나타내는 알고리즘을 선정했다.

CPU 로 실행이 되는 환경이므로 N 의 크기를 다양하게 설정하여 실험해보지 못했으나 각 알고리즘의 결과를 이론적으로 확인할 수 있었다. 코드로 만들 수 있는 부분은 다 구현했으며 향후 시뮬레이션화를 비롯하여 TSP 를 VRP 문제로 업그레이드하여 수정할 계획이 있다.

참고문헌 - References

1. Marco Dorigo. Ant colony optimization, 2006
2. 김여근. 메타휴리스틱스, 전남대학교출판부, 2017
3. Lee Jacobson. Ant colony optimization For Hackers, https://www.theprojectspot.com/tutorial-post/ant-colony-optimization-for-hackers/10, 2015