인공지능 과제2 - 결과 보고서

학번	20190850	이름	이수진
학년	3	제출일	2021-04-24
제목	탐색 기법을 활용한 TSP 문제의 최적해 찾기		

[결과 요약]

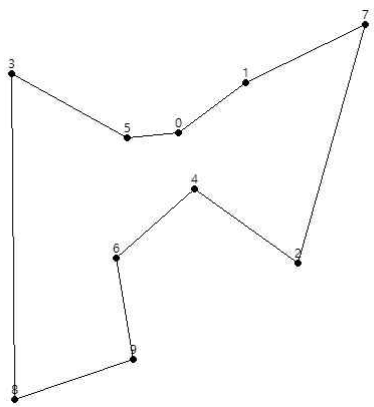
기술 내용		기술 여부	배점	비고
적용 알고리즘에 대한 설명		O	0.5	적용 기법의 개수 : 6
Exhaustive Search		O	1	200개 도시 최소값 :49978, 평균값 :49978
Steepest-ascent Hill-Climbing Search		O	1	200개 도시 최소값 :13909, 평균값 :14232.67
First-choice Hill-Climbing Search		O	1	200개 도시 최소값 :10578, 평균값 :11034.67
Simulated Annealing		O	1	200개 도시 최소값 :7400, 평균값 :7812.333
Genetic Algorithm		O	1	200개 도시 최소값 :20171, 평균값 :20522
추가 알고리즘	Random Restart Steepest-ascent Hill-Climbing Search	O	2	200개 도시 최소값 :13068, 평균값 : 17187
		X	0	-
결론 및 느낀 점		O	0.5	-
합계		8	8	최대 8점

※ 추가로 구현한 알고리즘이 3개 이상인 경우 “추가 알고리즘” 란에 칸을 나누어 추가. 추가 알고리즘의 경우 1개는 2점 부여, 2개는 3점 부여, 3개는 4점 부여 등. 단, 총점의 최대 점수는 8점임

※ 위의 [결과 요약]을 제외하고 본 서식을 수정하여 사용해도 됨

1. TSP (Traveling Salesman Problem)

- n 개의 도시가 주어진 경우 첫 번째 도시부터 모든 도시를 한 번씩만 경유하여 첫 번째 도시로 되돌아오되 전체 경유 거리를 최소화하는 문제



2. 적용 알고리즘에 대한 설명

1) Exhaustive Search

- 모든 경우에 대해 검사한다. 시간이 충분하면 최적의 해를 도출할 수 있다.

2) Steepest-ascent Hill-Climbing Search

- 현재 해로부터 모든 도시 쌍들을 교체하여 현재해보다 같거나 더 좋은 해가 있으면 이동하는 Hill-Climbing Search 알고리즘이다. 현재 해와 다음해 사이에 거리가 같다면 지정된 실행 기간까지 실행될 수 있다.

3) First-choice Hill-Climbing Search

- 현재 해로부터 이웃해 하나를 만들어 더 좋거나 같으면 이동하며, 지정된 실행 시간까지 실행된다.

4) Simulated Annealing

- Hill-Climbing Search와 같이 하나의 이웃 해를 만들어 더 좋으면 이동한다. 단, 이웃해가 현재해보다 좋지 않더라도 확률적으로 이동할 수 있다.

5) Genetic Algorithm

- 해들의 집합인 population을 기반으로 selection, crossover, mutation을 통해 해들을 진화시켜 나가는 방법이다.

6) Random Restart Steepest-ascent Hill-Climbing Search

- 초기 상태를 무작위로 생성해가며 Steepest-ascent Hill-Climbing Search를 수행한다. 지정된 실행 시간까지 실행된다.

3. 실험 환경

- OS : Microsoft Windows 10 Home
- 시스템 종류 : x64 기반 PC
- 프로세서 : Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz, 2208Mhz, 2코어 4논리 프로세서
- RAM : 12.0GB

4. 실험 결과 및 분석

1) Exhaustive Search

- 3분 제한 및 1회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

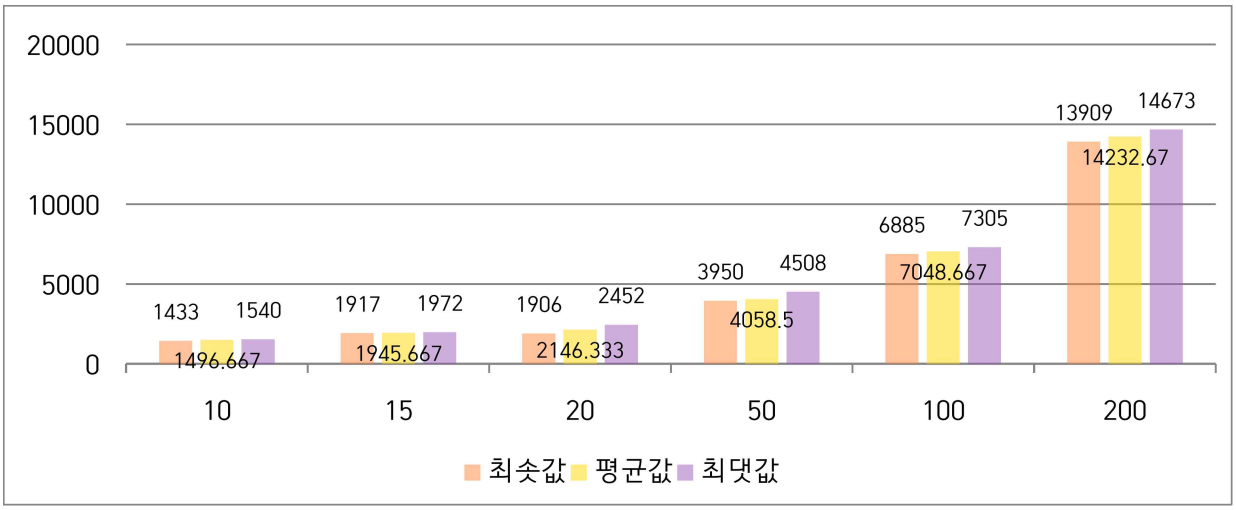
도시 수(개) / 실행 횟수(회)	10	15	20	50	100	200
1	1425	2366	3400	11319	24871	49978

- 3분 제한에서 15개부터 200개까지는 시간 초과가 나타남.
- 도시의 개수가 20개일 때부터 비교적 경유 거리가 커짐.

2) Steepest-ascent Hill-Climbing Search

■ 3분 제한 및 3회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

실행 횟수(회) \ 도시 수(개)	10	15	20	50	100	200
1	1540	1972	2452	4167	6956	13909
2	1433	1917	2081	3950	7305	14116
3	1517	1948	1906	4508	6885	14673

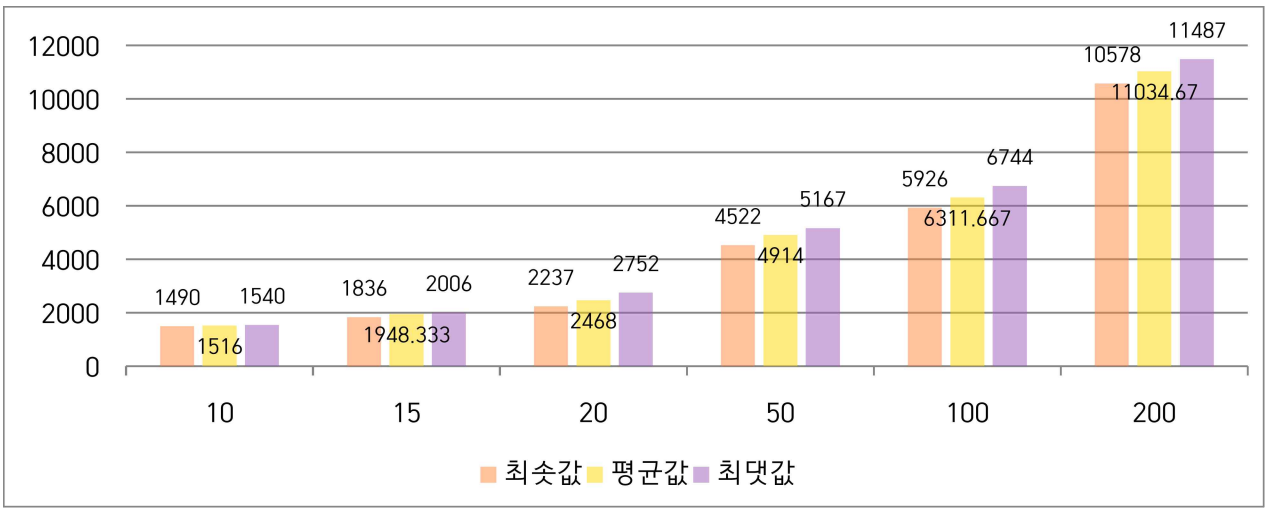


- 도시의 개수가 커질수록 평균값이 최솟값, 최댓값과 큰 차이를 보임.
- 50개부터 드물게, 200개부터는 모두 시간 초과됨.

3) First-choice Hill-Climbing Search

■ 3분 제한 및 3회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

실행 횟수(회) \ 도시 수(개)	10	15	20	50	100	200
1	1490	1836	2752	4522	6255	11309
2	1540	2003	2415	5167	6744	10578
3	1518	2006	2237	5053	5926	11487

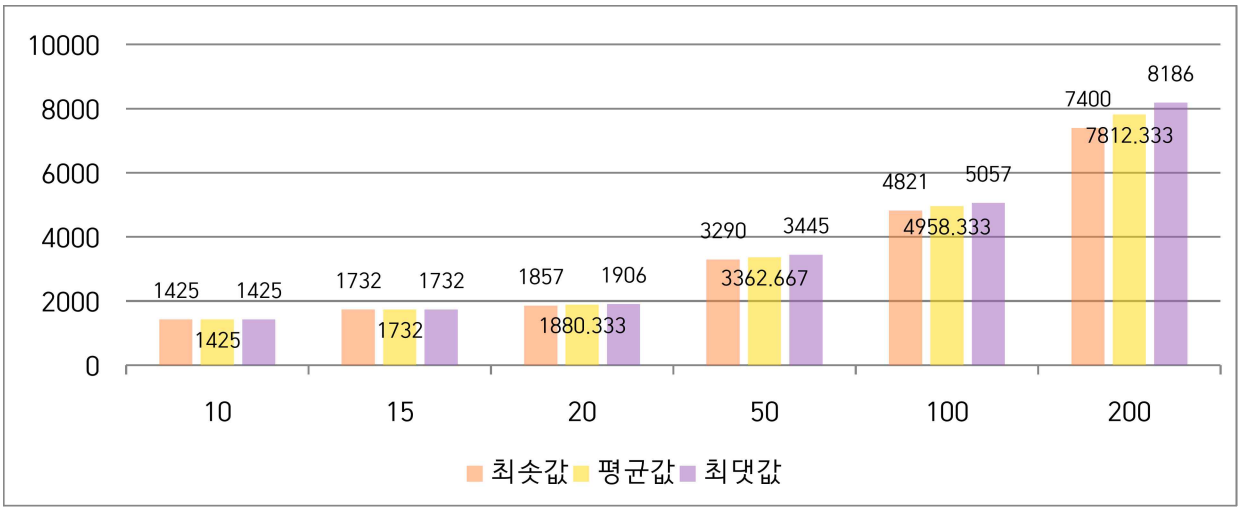


- 도시가 적을 때 (2)와 비교해 비슷하거나 조금 더 큼.
- 도시가 100개 이상일 때, (2)와 비교해 눈에 띄게 경유 거리가 짧아짐.
- 제한 시간(3분)까지 실행하므로 모든 과정에서 시간 초과가 나타남

4) Simulated Annealing

■ 3분 제한 및 3회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

실행 횟수(회) \ 도시 수(개)	10	15	20	50	100	200
1	1425	1732	1878	3353	5057	7851
2	1425	1732	1857	3290	4997	8186
3	1425	1732	1906	3445	4821	7400

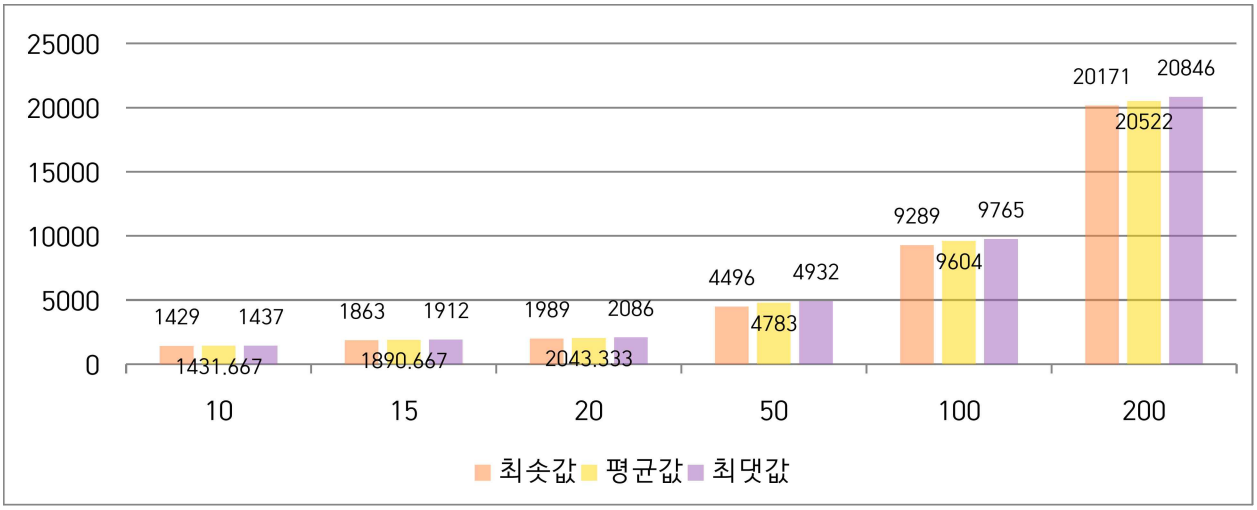


- Tmax는 10000.0, Tmin은 1.0, steps는 1000000으로 설정.
- (1) ~ (6)의 실험 중 가장 짧은 거리가 나타남.
- 15개 미만으로 거의 항상 같은 결과를 나타내고, 그 후에도 최솟값, 평균값, 최댓값의 차이가 크지 않음
- 개수가 커질수록 시간이 걸렸으나, 모든 실험에서 짧은 시간 안에 결과를 창출했음.

5) Genetic Algorithm

■ 3분 제한 및 3회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

실행 횟수(회) \ 도시 수(개)	10	15	20	50	100	200
1	1429	1912	2055	4921	9765	20549
2	1429	1897	2086	4932	9289	20171
3	1437	1863	1989	4496	9758	20846

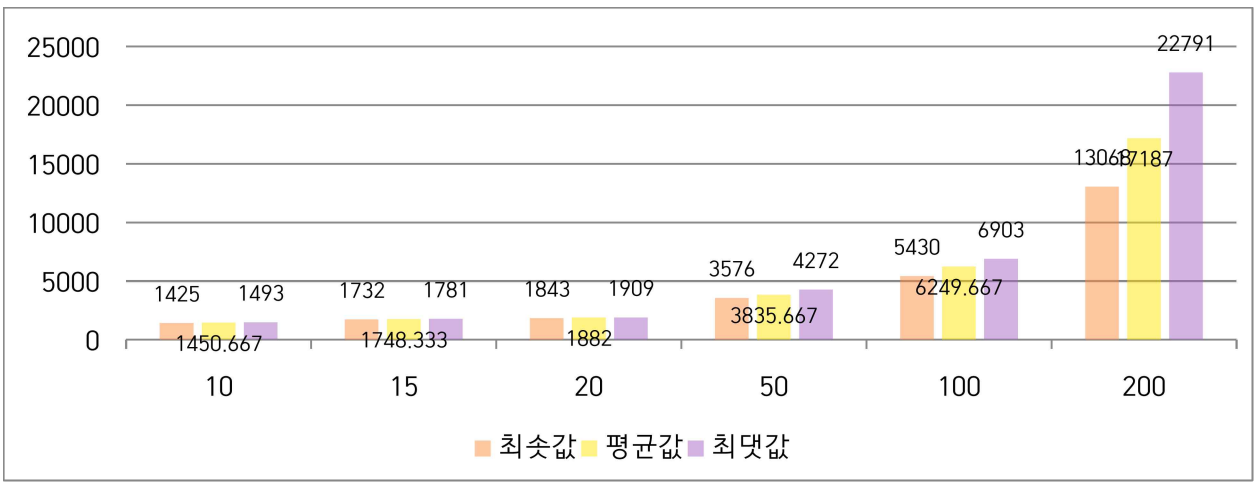


- 50개 이하일 때는 짧은 거리와 더불어 결과가 비슷하나, 100개 이상부터 비교적 거리가 커짐.
- 제한 시간 3분을 초과하지는 않으나, 도시가 많을수록 시간이 오래 소유됨.

6) Random Restart Steepest-ascent Hill-Climbing Search

■ 3분 제한 및 3회 실행(가로는 도시의 수, 세로는 실행 횟수를 나타냄)

실행 횟수(회) \ 도시 수(개)	10	15	20	50	100	200
1	1425	1732	1843	3576	6416	22791
2	1493	1781	1909	4272	5430	15702
3	1434	1732	1894	3659	6903	13068

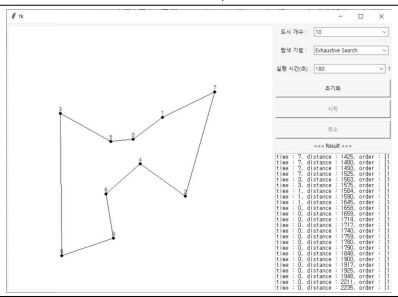
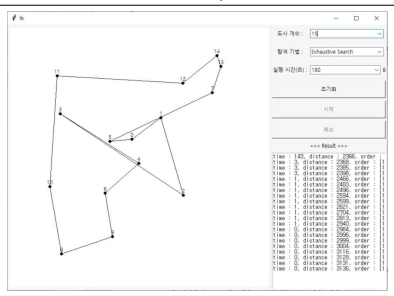
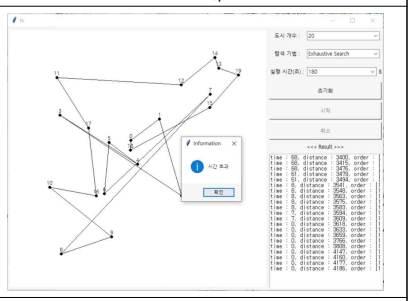
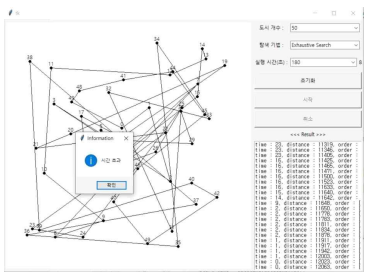
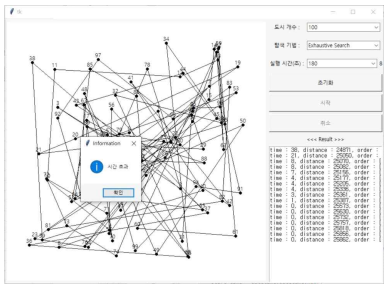
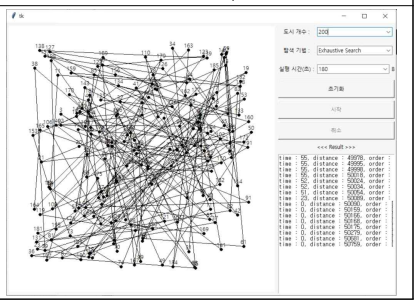


- (2)와 비교 시 100개 이하에서는 대부분 좋은 결과를 보임
- 200개일 때 최솟값과 최댓값의 차이가 매우 큼. 이는 알고리즘상 처음 초기 해에서 hill-climbing이 실행될 때 3분을 넘기는 경우 마지막 해를 결과로 보여주기 때문.
- (2)와 마찬가지로 50개부터 시간 초과가 보이고, 100개 이상일 때 모두 시간 초과됨.

5. 실행 결과 화면(예시)

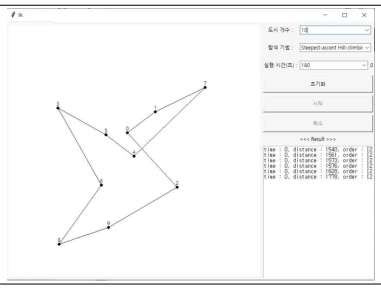
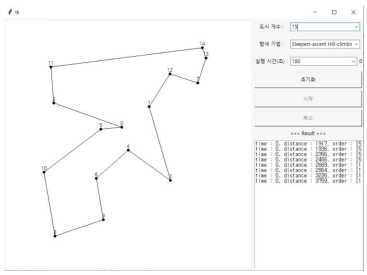
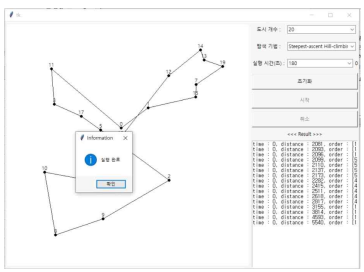
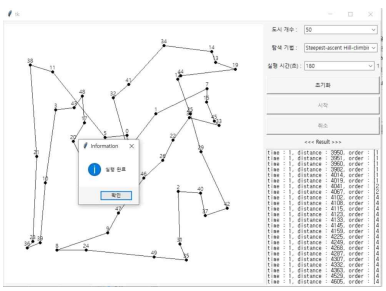
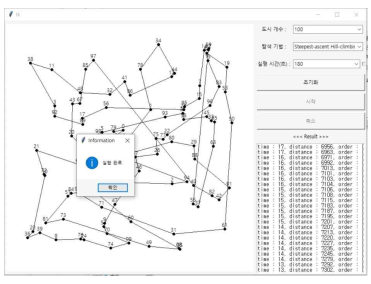
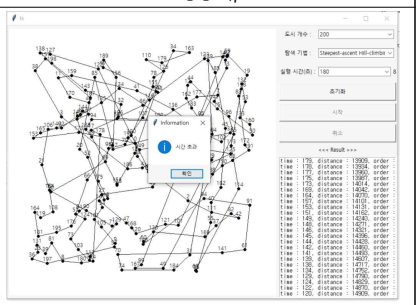
1) Exhaustive Search

■ 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면이다.

10개	15개	20개
		
50개	100개	200개
		

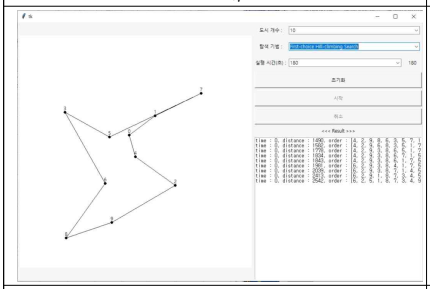
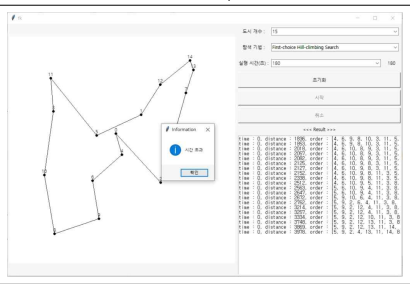
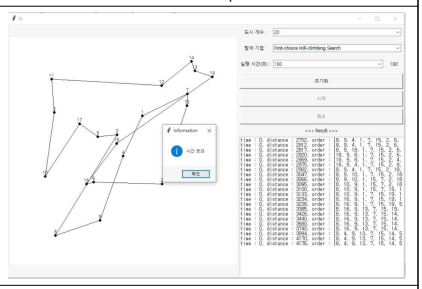
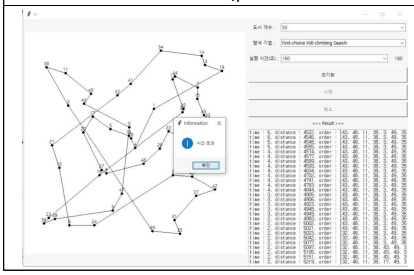
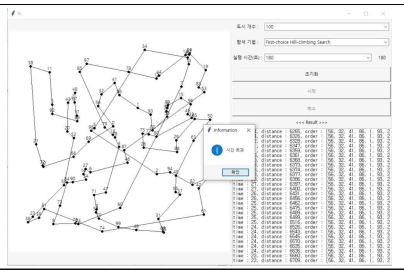
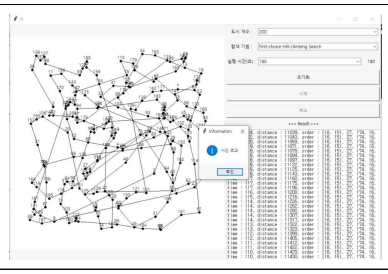
2) Steepest-ascent Hill-Climbing Search

- 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면 중 하나이다.

10개	15개	20개
		
50개	100개	200개
		

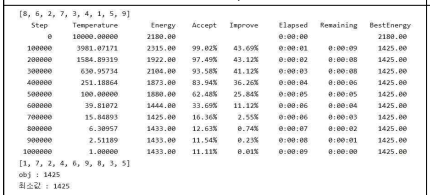
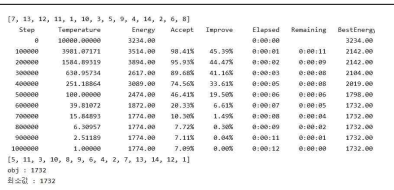
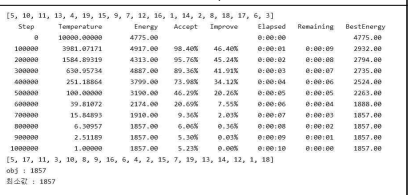
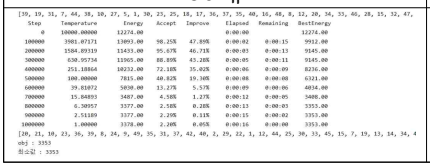
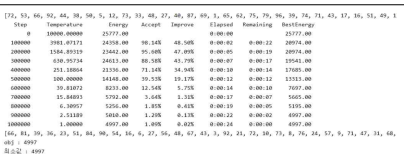
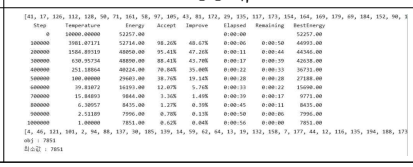
3) First-choice Hill-Climbing Search

- 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면 중 하나이다.

10개	15개	20개
		
50개	100개	200개
		

4) Simulated Annealing

- 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면 중 하나이다.

10개	15개	20개
		
50개	100개	200개
		

5) Genetic Algorithm

- 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면 중 하나이다.

10개	15개	20개
Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 9 List: [5, 3, 8, 9, 6, 4, 2, 7, 1] 최단 거리 : 1429.0828040100596	Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 14 List: [4, 1, 12, 14, 13, 7, 2, 9, 8, 10, 3, 11, 5, 6] 최단 거리 : 1897.2922564921455	Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 19 List: [10, 8, 9, 16, 6, 5, 17, 3, 11, 18, 4, 2, 15, 7, 19, 13, 14, 12, 1] 최단 거리 : 2055.0130635841065
50개	100개	200개
Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 49 List: [27, 16, 6, 5, 48, 32, 41, 7, 19, 45, 33, 30, 25, 15, 44, 12, 34, 14, 13, 1, 18, 47, 9, 21, 20, 46, 2, 최단 거리 : 4921.929991769585	Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 99 List: [28, 1, 93, 89, 67, 43, 69, 87, 75, 77, 80, 47, 57, 54, 6, 58, 44, 7, 96, 88, 26, 46, 31, 35, 22, 50, 최단 거리 : 9289.307719206858	Order initializer for like the Traveling Salesman Problem Slot [Mutator] (Count: 1) Name: G1DListMutatorSwap - Weight: 0.50 Doc: The mutator of G1DList, Swap Mutator .. note:: this mutator is :term:`Data Type Independent` Slot [Crossover] (Count: 1) Name: G1DListCrossoverPMX - Weight: 0.50 Doc: The PMX Crossover for G1DList (partially matched crossover) - G1DList List size: 199 List: [10, 59, 14, 142, 101, 121, 2, 166, 162, 123, 122, 189, 116, 22, 75, 174, 66, 52, 89, 194, 67, 43, 152 최단 거리 : 20549.171688185143

6) Random Restart Steepest-ascent Hill-Climbing Search

■ 왼쪽에서 오른쪽, 위에서 아래 순서대로 도시 수 10개, 15개, 20개, 50개, 100개, 200개 실행 화면 중 하나이다.

10개	15개	20개
시작 (init) time : 0, distance : 2530, order : [9, 5, 6, 1, 4, 3, 2, 7, 8] time : 0, distance : 1535, order : [6, 8, 9, 2, 4, 7, 1, 3, 5] time : 0, distance : 1518, order : [5, 6, 9, 8, 3, 1, 7, 2, 4] time : 0, distance : 1468, order : [5, 3, 8, 9, 6, 2, 7, 1, 4] time : 0, distance : 1433, order : [4, 5, 3, 6, 8, 9, 2, 7, 1] time : 0, distance : 1425, order : [5, 3, 8, 9, 6, 4, 2, 7, 1] 실행 종료	시작 (init) time : 0, distance : 3619, order : [3, 1, 9, 8, 5, 7, 2, 4, 13, 12, 11, 6, 10, 14] time : 0, distance : 1881, order : [11, 3, 10, 8, 9, 2, 7, 13, 14, 12, 1, 4, 6, 5] time : 0, distance : 1872, order : [6, 9, 8, 10, 3, 11, 5, 4, 2, 1, 12, 14, 13, 7] time : 0, distance : 1811, order : [4, 2, 1, 12, 7, 13, 14, 11, 3, 10, 8, 9, 6, 5] time : 0, distance : 1732, order : [1, 12, 14, 13, 7, 2, 4, 6, 9, 8, 10, 3, 11, 5] 실행 종료	시작 (init) time : 0, distance : 5151, order : [7, 9, 12, 18, 19, 11, 10, 14, 15, 1, 4, 3, 16, 2, time : 0, distance : 2011, order : [12, 14, 13, 19, 7, 15, 1, 5, 18, 4, 2, 6, 16, 9, 8, 10, time : 0, distance : 1968, order : [18, 4, 2, 9, 8, 6, 16, 10, 3, 11, 17, 5, 1, 15, 7, 19, : time : 0, distance : 1954, order : [4, 6, 16, 5, 17, 11, 3, 10, 8, 9, 2, 15, 7, 19, 13, 14, time : 0, distance : 1912, order : [4, 6, 16, 10, 8, 9, 2, 15, 7, 19, 13, 14, 12, 1, 11, 3, time : 0, distance : 1843, order : [18, 5, 17, 11, 3, 10, 8, 9, 16, 6, 4, 2, 15, 7, 19, 13, 실행 종료
50개	100개	200개
시작 (init) time : 0, distance : 12684, order : [48, 4, 3, 15, 43, 36, 8, 1, 20, 27, 26, 28, 35, 12, 32, 10, 38, 46, 5, 3 time : 179, distance : 4273, order : [5, 17, 20, 21, 10, 23, 36, 39, 8, 24, 49, 35, 31, 9, 43, 3, 38, 11, 48, 32, 2 시간 초과	시작 (init) time : 0, distance : 25392, order : [20, 40, 5, 56, 22, 65, 25, 83, 29, 3, 9, 73, 2, 69, 49, 72, 97, 6, 42, 90, time : 179, distance : 6416, order : [79, 5, 18, 28, 4, 87, 29, 63, 88, 2, 60, 99, 74, 70, 57, 9, 47, 46, 26, 80, 65, 3 시간 초과	시작 (init) time : 0, distance : 53342, order : [93, 76, 57, 151, 23, 197, 85, 62, 71, 123, 96, time : 179, distance : 22791, order : [51, 76, 57, 70, 23, 197, 59, 62, 163, 123, 96, 102, 시간 초과

6. 결론 및 느낀 점

■ 결론

3분의 제한을 두고 TSP를 해결할 때, simanneal 라이브러리를 이용한 Simulated Annealing에서 시간, 해의 결과에서 가장 좋은 값을 보인다. 물론, 이는 3분의 제한을 두고 3회 실행했을 때의 결과이므로 실행 횟수를 좀 더 늘리거나, 제한 시간을 늘리면 다른 결과가 나올 수 있다.

■ 느낀 점

TSP 문제를 Search로 구현하고 실험하는 과정을 통해서 보다 주어진 알고리즘 및 구현 알고리즘에 대한 이해도를 높일 수 있었다. 다만 직접 구현했던 Random Restart Hill-Climbing Search는 해를 구하지 못하고 시간이 초과할 경우까지 고려해서 초기 해를 다시 생성하는 코드를 작성할 수도 있었을 것 같은데 그러지 못해서 아쉬웠다.

[소스 코드]

1) Random Restart Steepest-ascent Hill-Climbing Search

- 기존 Steepest-ascent Hill-Climbing 코드를 바탕으로 초기 해를 랜덤으로 생성하도록 변경한 코드이다. 기존 hill-climbing에서 같거나 더 좋은 해가 없으면 해를 창출하는데, 이때 초기 해를 다시 생성한다. 그리고 이전의 hil-climbing 결과와 비교하여 좋으면 출력하고, 그렇지 않으면 출력하지 않는다. 이러한 while()문은 제한 시간인 3분이 될 때까지 반복된다.

```
def Random_SA_HillClimbingSearch() :
    …
    temp = False # 다시 초기해를 생성할지 알려주는 요소
    temp2 = False # 결과를 내지 못하고 시간 초과됐는지 알려주는 요소
    pre_best_distance = 0

    while True:
        if temp :
           # 초기해 랜덤 생성
           cities = [i for i in range(1, city_count)]
           random.shuffle(cities)
           best_distance = GetDistance(cities)

           cur_best_distance = best_distance
           temp = False

        … # 기존 tsp_main.py의 hill-climbing code

        else:  # 같거나 더 좋은 해가 없으면 다시 반복
            # 첫 hill-Climbing 이거나 기존 해보다 결과 해가 좋으면
            if pre_best_distance == 0 or best_distance < pre_best_distance :
               elapsed = int((datetime.datetime.now() - start_time).total_seconds())
               print("time : " + str(elapsed) + ", " + "distance : " + str(
                    best_distance) + ", " + "order : " + str(cities) + "\n")
               pre_best_distance = best_distance
            temp = True
            if not temp2:
               temp2 = True
         if (time.time() - start) > 180:  # 180초 초과
             if not temp2 : # 만약 hill-climbing에서 처음 초기해 후 도중
                            탐색 종료가 되었다면
               # 마지막 해 출력
               elapsed = int((datetime.datetime.now() - start_time).total_seconds())
               print("time : " + str(elapsed) + ", " + "distance : " + str(best_distance)
                     + ", " + "order : " + str(cities) + "\n")
               return -1
             return 0

return 1
```