제 10 회 학사학위 졸업논문
지도교수 박 진 완

1000개 이상 도시 외판원문제(TSP) 해결을 위한 알고리즘과 OpenGL을 이용한 PATH 시각화

(Algorithms for solving more than 1000 city traveling salesperson problem (TSP)
and
PATH visualization using OpenGL)

이 지 훈

School of Integrative Engineering
in Chung-Ang University

2018 . 12

제 10 회 학사학위 졸업논문
지도교수 박 진 완

1000개 이상 도시 외판원문제(TSP) 해결을 위한 알고리즘과 OpenGL을 이용한 PATH 시각화

(Algorithms for solving more than 1000 city traveling salesperson problem (TSP)
and
PATH visualization using OpenGL)

이 논문을 학사학위 졸업논문으로 제출합니다.

중앙대학교 공과대학
융합공학부
이 지 훈

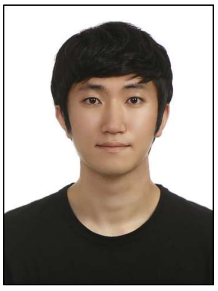
2018년 12월

중앙대학교 공과대학 융합공학과 이 지 훈 의 공학사 학위 취득을 위한 요구 사항 중 그 일부인 본 졸업논문을 인정함.

2018년 12월

지도교수 : 박 진 완 (인)

약 력



이지훈은 1993년 인천시 계양구 효성동에서 이경근씨와 박주선씨의 장남으로 태어났다. 2012년 인천고등학교를 졸업하고, 2017년도에 중앙대학교 공과대학 융합공학부로 입학하여 현재 4학년에 재학 중이며 2019년 02월에 졸업할 예정이다.

감사의 글

2017년 편입학 시험에 합격하여 자랑스러운 중앙대학교의 학생이 된 것이 불과 얼마 전의 일인 것 같은데 벌써 2년이라는 시간이 흘러서 졸업을 앞두고 있습니다. 중앙대학교에 와서 많은 가르침을 주신 교수님들과 교직원 분들에게 감사의 말씀을 드립니다. 제가 중앙대학교에서 2년간 배웠던 지식들은 앞으로 저의 삶에 정말 큰 영향을 줄 것 이며, 저는 그것들을 바탕으로 사회에 도약하여 학교의 이름을 빛낼 수 있도록 노력할 것입니다. 먼저 사회에 나가서 빛내고 있는 선배님들과 함께 나아갈 동기, 후배들에게 부끄럽지 않게 행동하겠습니다.

지금까지 살아온 시간보다 앞으로 살 수 있는 시간이 많고, 그렇기에 미래에 대한 고민도 많지만 그럴 때 마다 옆에서 이끌어주시고 뒤에서 밀어주시는 부모님에게 정말 감사드립니다. 옆에서 때로는 먼저가기도 하고 때로는 늦기도 하지만 항상 함께 인생이라는 긴 여정을 함께하는 나의 자랑스러운 친구들에게도 고마움을 전합니다.

초등학교 6년, 중학교 3년, 고등학교 3년, 그리고 대학교 4년 지금까지 총 16년의 교육을 통해 배운 나의 지식과 그동안의 쌓은 역량을 내년부터는 사회에 나가 펼쳐야하기에 설렘과 동시에 한편으로는 불안하기도 하지만 그동안 제가 해온 것들이 틀리지 않았을 거라 믿기에 기대되는 마음이 더 큰 것 같습니다.

Computer Science에서 학사학위를 받으면서, "현대 컴퓨터의 아버지" 찰스 배비지(Charles Babbage)와 "폰 노이만구조"를 고안한 역사상 가장 뛰어난 천재 '존 폰 노이만(John von Neumann)'에게 경의를 표합니다.

목 차

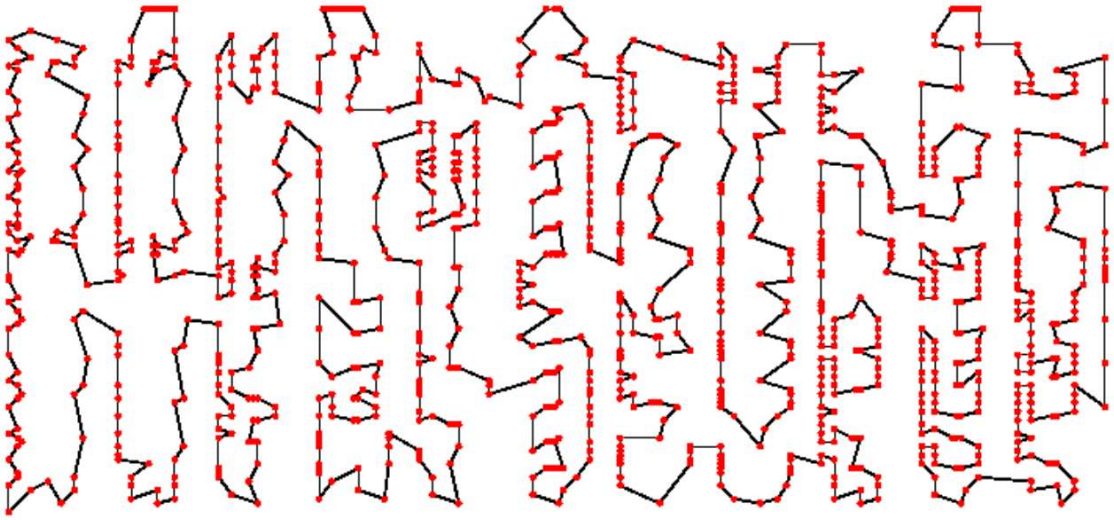
1000개 이상 도시 외판원문제(TSP) 해결을 위한 알고리즘과 OpenGL을 이용한 PATH 시각화

(Algorithms for solving more than 1000 city traveling salesperson problem (TSP) and
PATH visualization using OpenGL)

1. Introduction

외판원순환 문제(Traveling Salesperson Problem, TSP)는 n개의 도시를 최소 비용으로 이동하는 경로를 찾는 NP-complete문제이다. 하나의 도시를 시작으로 나머지 (n-1)개의 도시를 한 번씩 거쳐서 원래 출발한 도시로 돌아오는 해밀턴 사이클(Hamiltonian cycle)을 이룬다. 도시들 간의 모든 거리를 다 계산해서 최적화된 해를 찾으려고 한다면, (n-1)! 의 시간복잡도를 갖는다. 그렇기 때문에 도시의 수가 증가할수록 시간복잡도가 기하급수적으로 커지게 된다. 만약 20개의 도시가 있다면 10^{16} 가지의 가능한 경로의 수를 갖는다. 도시의 수가 많아질수록 최단경로를 찾기는 비용이 너무 커진다. 그렇기에 최단경로를 찾기 위한 많은 방법이 고안됐다. 도시의 수가 많아지면 계산비용이 너무 커지기 때문에 고안된 방법들은 적은비용으로 최적의 경로를 찾기 위해 Optimal이 아닌 Heuristic 알고리즘이다.

경로를 예측하는 여러 가지 알고리즘들을 간단하게 생각해볼 수 있다. DFS, BFS, Backtracking, branch and bound, A*과 같은 Classical한 Exhaustive Search들이 대표적이다. 이와 같은 방법들은 심플하고 가능한 경로들을 만들어내지만 도시의 수에 따라 매우 큰 시간복잡도를 갖게 된다. 이를 개선하기 위해 Greedy 알고리즘을 이용하면 매우 간단하고, 비용이 저렴한 알고리즘을 구현할 수 있다. 예를 들어, Nearest-neighbor search를 이용한 접근방식은 누구나 쉽게 생각할 수 있다. 그 외에도 Hill-climbing, Simulated Annealing, k-opt 등과 같은 방법을 살펴보고 몇 가지 알고리즘을 구현해 볼 것이다. 경로 데이터는 "Solving TSPs"에 있는 optimal path가 존재하는 'XIT1083'를 사용한다. length는 3558이다.



(a)XIT1083

그림1. Optimal path가 연결된 XIT1083(3558).

2. Algorithms

2.1. 최근접 이웃(Nearest-Neighbor, NN) 알고리즘

가장 쉽게 구현할 수 있는 알고리즘 중 하나는 최근접 이웃을 이용한 알고리즘 이다. 알고리즘의 이름에서 알 수 있듯이, 처음 출발 도시로부터 방문하지 않은 도시 중 거리가 가장 짧은 도시를 연결하면서 경로를 생성하는 알고리즘이다. 랜덤하게 하나의 도시를 선택해서 그 도시로부터 가장 인접한 도시를 연결하고 도착한 도시에서 아직 연결되지 않은 도시들 중 가장 인접한 도시를 연결해서 찾아가는 알고리즘 이다.

2.2. 언덕오르기(HILL-CLIMBING) 알고리즘

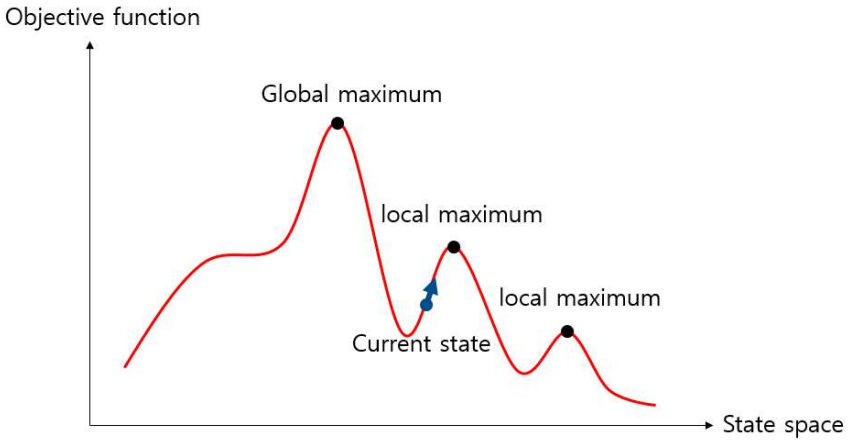


그림2. 언덕오르기 알고리즘의 상태그래프

언덕오르기 알고리즘은 기울기 하강법(gradient descent)이라고도 불린다. 기울기에 비례해서 단계적으로 최댓값 혹은 최솟값에 도달하는 알고리즘이다. 하지만 이 방법에는 치명적인 약점이 있다. 바로 local optima에 빠질 수밖에 없는 문제를 갖고 있다. 예를 들어서, 산에 오르는데 안개가 심하게 있어서 앞이 보이지 않는 상황이다. 그런 경우 계속 오르다가 다음 발을 내디디는 순간 오르막이던 산이 내리막으로 바뀌었다. 그런 경우 발을 내딛기 바로 직전의 위치를 산의 정상이라고 생각할 것이다. 하지만 그 위치는 정말 산의 정상일수도 있고 그저 하나의 작은 봉우리일 수도 있다. 위의 그래프에서 보면 시작하는 위치에 따라서 local maximum에 도달할 수도, global maximum에 도달할 수도 있다. 그렇기에 시작 위치를 어떤 방법으로 잡아야 할까? 하는 문제에 도달한다. 시작점을 잡기위한 많은 방법이 있지만 local optimum에 도달하면 랜덤하게 시작점의 위치를 정해서 다시 시작하는 방법

도 생각할 수 있는 방법 중 한가지이다.

2.2.1. K-opt algorithm

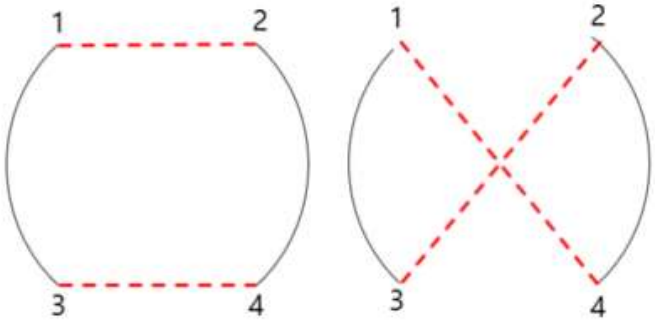


그림3. 2-opt의 경로

k-opt방식은 local search 알고리즘이다. 2-opt, 3-opt 등 앞의 k에 따라 경로를 비교하는 개수가 달라진다. <그림3>과 같이 1부터 4까지 4개의 도시가 있다. 이들은 하나의 경로를 이루어야하기 때문에 (1,3)+(2,4) 이런 식으로 연결 될 수 없다. 그러면 가능한 경우는 {(1,2)+(3,4)} 또는 {(1,4)+(2,3)} 두 가지의 경우가 있다. 두 가지 경우의 거리 값을 비교해서 더 가까운 경로를 연결하면서 점점 거리를 줄여가는 방식이 2-opt 이다.

3-opt는 6도시의 경로를 비교하기 때문에 8가지의 경우의 수가 생긴다. <그림4>를 보면 2-opt로 계산 가능한 경우가 4가지가 존재하고 새로운 4가지의 경로가 추가된다. 2-opt의 시간복잡도는 $O(n^2)$, 3-opt는 $O(n^3)$ 그리고 k-opt의 시간 복잡도는 $O(n^k)$를 갖는다.

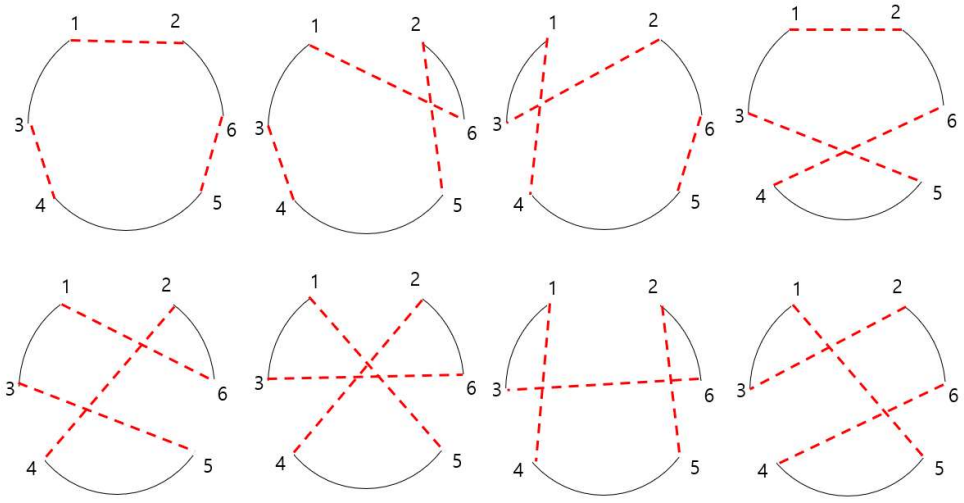


그림4. 3-opt의 경로

2.3. 담금질 기법(Simulated Annealing, SA)

담금질 기법은 금속이 냉각되면서 점점 안정화 되어 가는 과정을 모방한 global optimization을 수행하는 알고리즘이다. 해가 local optima에 빠지는 것을 방지하기 위해 고안된 알고리즘이다.

$$P = \frac{1}{1 + e^{\frac{eval(Vc) - eval(Vn)}{T}}} \quad (1)$$

식(1)의 결과로 얻은 P 값을 가지고 경로를 확률적으로 연결하게 된다. 온도변수 T(Temperature)에 변화를 주기 때문에 P값은 T에 종속적이게 되고, 새로운 경로는 현재 값보다 나빠질 가능성도 있다. T값을 서서히 증가 시키다가 매우 큰 값이 되면 분자의 값에 상관없이 P값은 0.5에 수렴하게 되고 랜덤하게 연결한다. 만약 T가 1에 근접하면 HILL-CLIMBING과 같다. 계속해서 경로를 찾다보면 값이 안정적으로 감소한다.

3. Visualization

OpenGL/glut를 이용해서 구현했다. 데이터 파일을 읽어 와서 x, y좌표를 저장하고, 입력받은 데이터의 크기에 따라 map의 크기를 유동적으로 조정하기 위해 max_x, max_y를 glOrtho의 right, top 값의 인자로 활용한다. 입력받은 x, y좌표를 이번에는 GL_POINTS를 사용해서 화면에 좌표를 찍는다.

```
glBegin(GL_POINTS);
for (int i = 1; i <= len; i++)
{
    glVertex2f(v[i].first, v[i].second);
}
glEnd();
```

알고리즘의 결과로 생성된 경로의 순서가 저장된 파일을 읽어서 path를 저장하고, GL_LINE_STRIP을 사용하여 저장했던 경로를 결과의 순서대로 좌표 위에 연결한다.

```
glBegin(GL_LINE_STRIP);
for (int i = 0; i <=len; i++)
{
    glVertex2f(v[path[i]].first, v[path[i]].second);
}
glEnd();
```

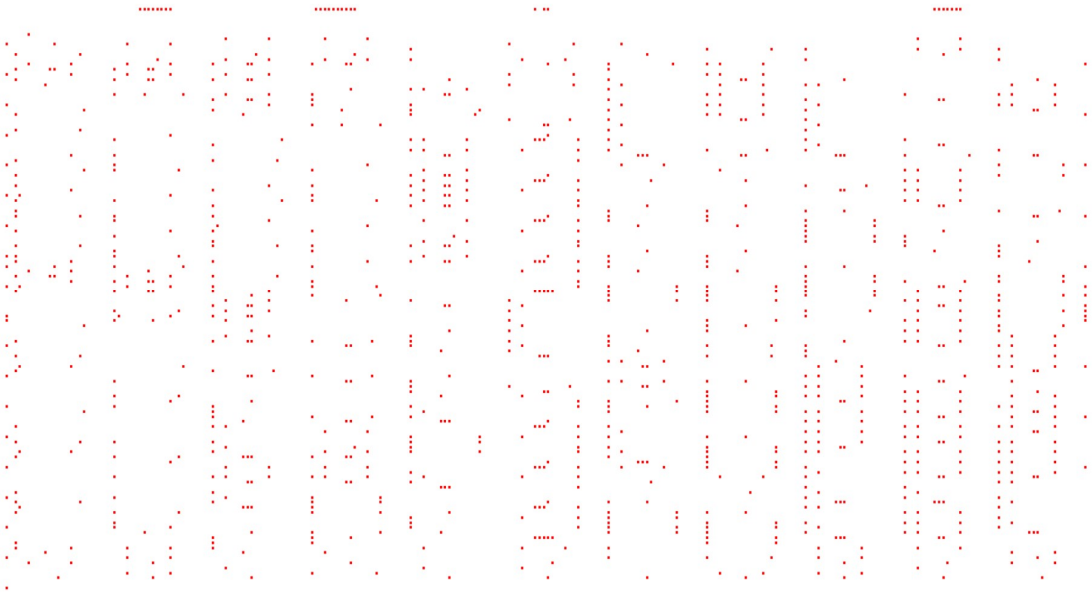


그림5. OpenGL/glut로 구현한 XIT1083좌표

4. Result

4.1. NN

1. 랜덤하게 시작점을 잡는다.
2. 그 점으로부터 가장 가까운 점을 연결한다.
3. 연결한 점으로부터 가장 가까운 아직 연결되지 않은 점을 연결한다.
4. 2, 3번을 반복한다.

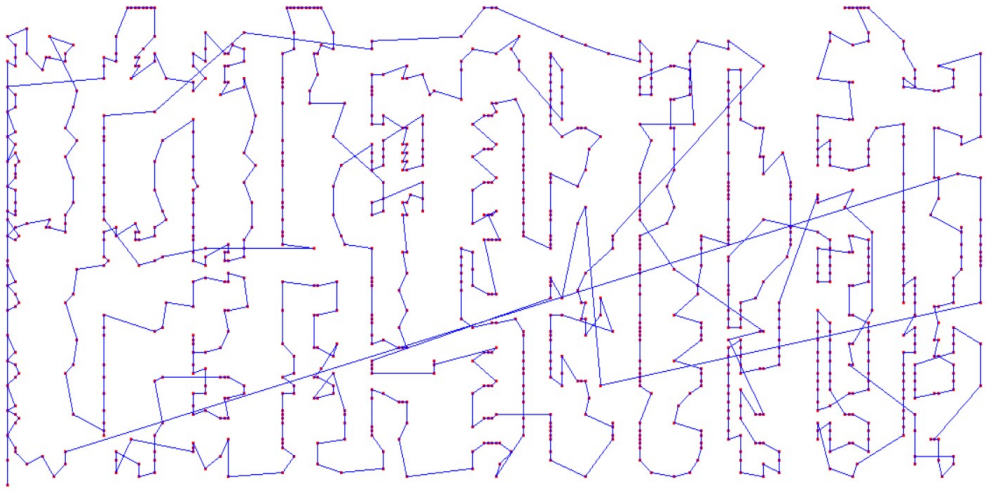


그림6. NN으로 구한 XIT1083 경로(4708).

한 점으로부터 아직 연결되지 않은 가까운 점을 search하고 연결하기 때문에 계산 비용이 많이 발생한다. 그렇기 때문에 처음에 각 점들 사이의 거리를 구하고 그 거리들을 오름차순으로 sort하고, 아직 연결되지 않은 점들을 짧은 순서대로 연결해주는 것으로 매 점마다 search하는 비용을 개선했다.

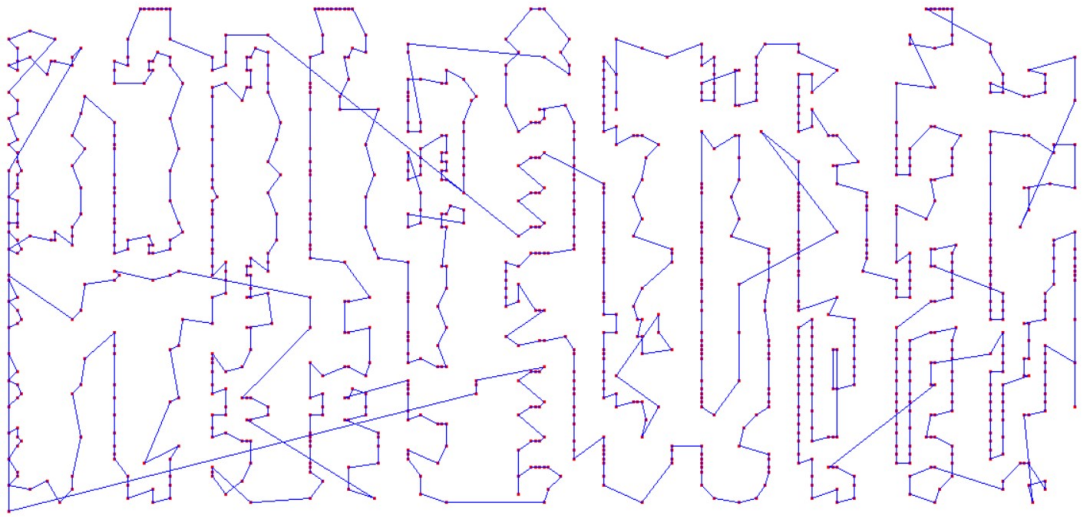


그림7. 개선된 NN으로 구한 XIT1083의 경로(4280).

<그림1>과 비교했을 때, 두 NN모두 중간에 긴 선과 꼬인 선들이 여러 곳에서 보인다. 기본 NN의 거리는 4708, 개선한 NN은 4280이다. optimal path와 오차율은 각각 **24.42%**, **16.87%**를 보인다. Sorting을 통해서 성능을 상당부분 개선가능 하다.

4.2. k-opt

1. k-opt를 이용해서 거리가 짧아지는 방향으로 계속해서 경로를 개선한다.
2. 거리가 더 줄어들면 교환한다.
3. 1, 2번을 반복한다.

4.2.1. 2-opt

2-opt는 <그림3>에서처럼 원래의 경로와 두 edge가 바뀐 두 가지 경우 중 더 짧은 거리를 선택해서 연결해준다. 비교할 대상은 랜덤하게 선택한다.

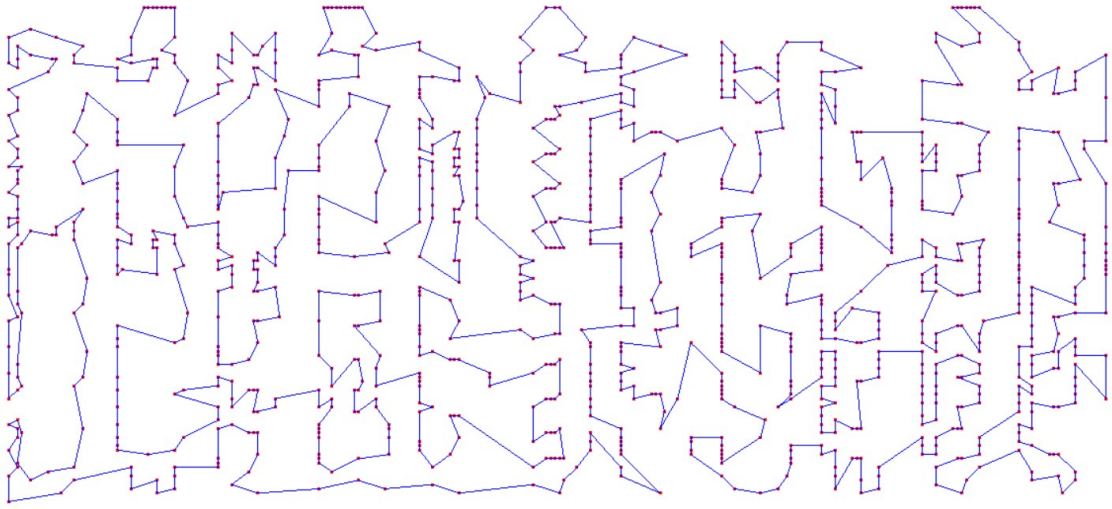


그림8. 2-opt로 구한 XIT1083의 경로(3878).

2-opt는 2초 만에 3878라는 경로를 도출했다. optimal path와의 오차율은 **8.26%**이다. <그림6>과 <그림7>에서 보였던 교차되고 긴 선들이 보이지 않는다. 2-opt을 이용하면 확연히 경로가 개선된다.

4.2.2. 3-opt

3-opt는 <그림4>에서 보였듯이 8가지의 경우를 계산해야해서 2-opt보다 계산비용이 더 많이 발생한다. 가능한 경우 중 현재보다 짧은 경우가 있으면 edge를 교환한다.

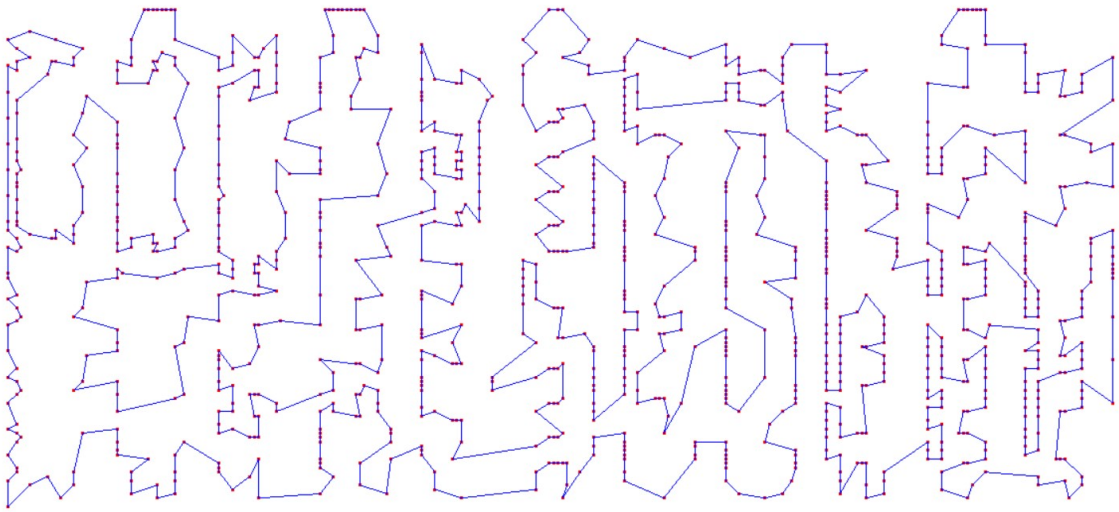


그림9. 3-opt로 구한 XIT1083의 경로(3783).

3-opt로 구한 경로의 길이는 3783이다. optimal path와의 오차율은 **5.95%**이다. 2-opt보다 오차율은 2%이상 감소했지다. 하지만 2-opt는 2초 만에 답을 도출했지만 3-opt는 1484초의 시간이 걸렸다. 계산에 대한 비용이 많이 증가한 만큼 더 많은 경로를 보기 때문에 local optima를 최대한 피하면서 개선할 수 있다.

4.2.3. Random restart(2-opt)

2-opt의 성능을 개선하기 위해 local optima에 도달하면 random restart를 사용했다.

1. 한 점을 랜덤하게 잡는다.
2. 2-opt를 이용해서 거리가 짧아지는 방향으로 계속해서 경로를 개선한다.
3. local optima에 빠지면 랜덤하게 다른점을 잡는다.
4. 2, 3번을 반복한다.

Random restart 2-opt를 이용한 알고리즘에서는 594초의 시간을 소모하고 3806이라는 결과를 얻었다. optimal path와의 오차율은 **6.52%**이다. local optimum에 도달하면 다른 경로를 찾는 방법이 알고리즘 개선에 도움이 된다는 것을 알 수 있다.

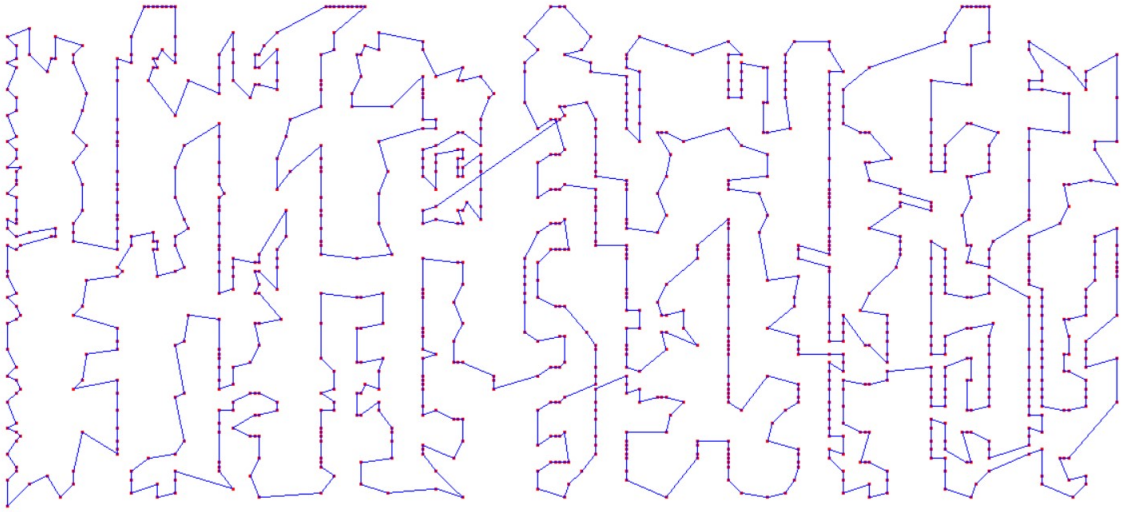


그림10. Random restart(2-opt)로 구현한 XIT1083 경로(3806).

4.4. SA

1. 초기 경로를 설정한다.
2. T의 초기값 설정한다.
3. 경로의 개선을 수행한다.
4. 온도를 점점 낮춘다.
5. 2, 3, 4를 반복 수행한다.

초기경로를 NN으로 연결하고, T값을 초기에 5000으로 설정하고 25번 2-opt를 수행하면 T값을 95%씩 낮춰서 T=1이 될 때까지 감소시키고, T가 1일 때는 더 이상 감소시키지 않고 경로를 개선했다. 1380초의 시간이 걸려서 3791이라는 값을 얻었다. 오차율은 **6.15**%이다.

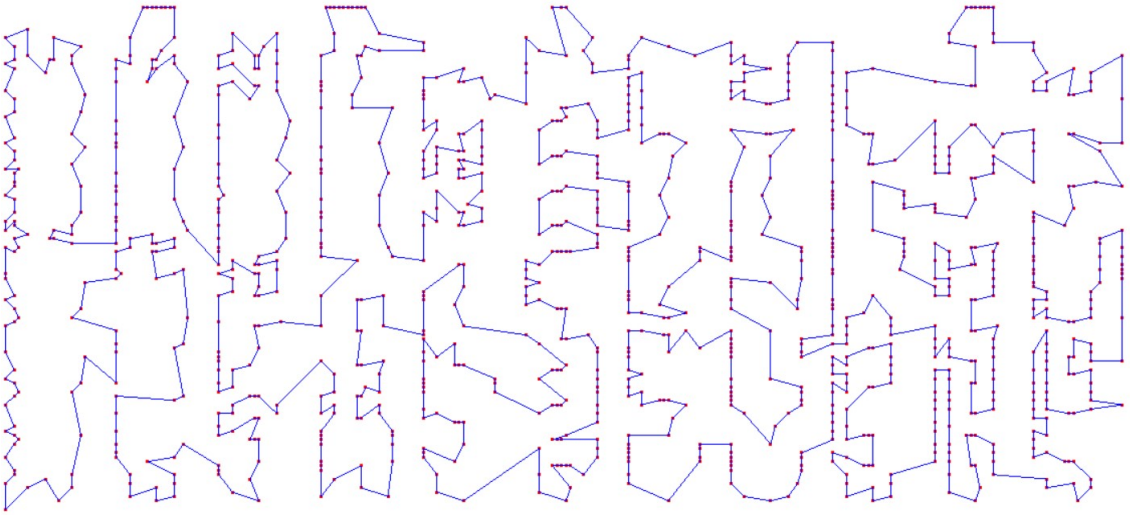


그림11. SA로 구현한 XIT1083 경로(3791).

4.5. Limited time(10s)

외판원이 경로를 찾아야하는데 무한정 시간동안 경로를 개선할 수 없기에 제한된 시간(10초)동안 어떤 알고리즘이 가장 좋은 성능을 보이는지 확인했다.

Algorithm	Length	times(s)
NN	4280	0.011
2-opt	3885	2
3-opt	3880	10
SA	3988	10

표1. 제한된 시간(10초) 동안 알고리즘 성능비교

5. Conclusion

초기 경로를 NN으로 연결하면 16%정도의 오차율을 갖는 경로를 얻을 수 있다. NN으로 구한 경로의 정확도의 개선을 위해서는 많은 시간을 투자해서 local optimum인 부분들을 개선해서 global optimum을 만드는 방법을 이용할 수 있다.

2-opt 방식을 이용하면 8%정도의 오차율을 갖는 경로를 찾을 수 있고, 3-opt방식을 이용하면 오랜 시간이 걸리지만 global optimum과 6%이하의 오차율을 가지는 경로를 찾을 수 있다. 2-opt를 개선하기 위해 random restart방식을 이용하면 6.5%정도의 정확도를 갖는 경로를 구할 수도 있다. SA 알고리즘은 초기 T의 설정을 잘 해줘야 한다.

TSP문제는 도시의 수에 따라 시간복잡도가 매우 커질 수 있기 때문에 결국에 알고리즘의 정확성과 시간을 둘 다 고려해야 하는 문제이다. 그렇기에 만약 짧은 시간 안에 결과를 도출해야 한다면 2-opt방식은 좋은 선택이 될 수 있다.

6. References

[1] Blazinskas, A., & Misevicius, A. (2011). Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. Kaunas University of Technology, Department of Multimedia Engineering, Studentu St, 50-401.

[2] Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. Journal of the operations research society of America, 2(4), 393-410.

[3] Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization, 1, 215-310.

[4] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the

traveling-salesman problem. Operations research, 21(2), 498-516.
[5] Zhan, S. H., Lin, J., Zhang, Z. J., & Zhong, Y. W. (2016). List-based simulated annealing algorithm for traveling salesman problem. Computational intelligence and neuroscience, 2016, 8.
[6] AlSalibi, B. A., Jelodar, M. B., & Venkat, I. (2013). A comparative study between the nearest neighbor and genetic algorithms: a revisit to the traveling salesman problem. International Journal of Computer Science and Electronics Engineering, 1(1), 34-38.
[7] Lim, A., Rodrigues, B., & Zhang, X. (2006). A simulated annealing and hill-climbing algorithm for the traveling tournament problem. European Journal of Operational Research, 174(3), 1459-1478.