TSP_GA 알고리즘 분석 보고서

20202058 신대니

1 개요

유전 알고리즘을 직접 구현해 TSP 최적화를 진행해 보았다.

2 구현 언어 및 방법

1. 모든 구현은 C++언어로 진행 하였다.

2. 도시들은 2차원 벡터에 저장했으며, 도시들의 거리를 구할 때마다 dist함수를 호출하는 것 보다 W[1000][1000] 행렬에 저장해 두는 것이 성능에 더 도움될 것이라 생각해 도시들 저장 후에 바로 W행렬을 구했다.

3. Path - 정답 후보가 될 염색체들은 Path객체에 저장했다. Path객체에는 경로 정보를 담고있는 벡터 x, 해당 경로의 비용, 해당 경로 중 가장 유망해 보이는 부분의 시작index와 끝 index, 그리고 그 부분의 비용, 각 Edge의 평균의 정보를 가지고 있다. Path::Cal_cost() 함수에서 경로의 총 비용을 구하며 가장 유망한 부분(크기가 S로 고정 되어있는 경로들 중 cost가 가장 작은 부분)의 비용도 함께 구했다. 유망한 부분의 길이는 전처리기를 통해 S를 유동적으로 바꿔가며 실험을 진행했다. 이 부분은 교차 부분에서 더욱 자세하게 설명할 예정이다.

4. 경로 x는 해답이 순환되어 배열 상으로 다르지만 논리적으론 같은 경우를 대비하여 강의 자료에 소개된 것 처럼 x[i] = j 일 경우 i에서 j로 간다는 뜻으로 경로를 저장했다.

5. 염색체들은 정렬이 자주 필요할 것 같아 정렬이 용이한 list로 관리했다. 총 N개의 염색체를 가지고 유전 알고리즘을 진행했으며, 한 세대가 끝나도 N개를 계속 유지하도록 관리했다. N은 전처리기를 통해 유동적으로 결과를 확인하며 실험을 진행했다.

6. 초기에 염색체를 만드는 과정에서 랜덤으로 한 vertex(city)를 잡아 greedy 알고리즘을 통해 중복을 제외하고 가장 가까운 vertex로 연결해 최적화 했다. 위와같은 방법으로 N개의 염색체를 만들었다

7. 선택 – 선택은 정렬된 N개의 염색체 중 상위로 좋은 것 들 부터 골라 유전을 진행했다. (1, 2), (3, 4) … 순으로 선택해 마지막 (N-1, N)까지 선택될 수 있도록 했따.(따라서 N은 항상 짝수로 진행했다)

8. 교차 – 교차 부분에서 유망한 부분은 S가 쓰인다. 선택된 두개의 Path중 유망한 부분의 비용이 작은 Path를 a로, 다른 Path를 b로 crossover 함수를 호출한다. Crossover함수는 두 Path로 교차를 진행 후, 만들어진 자식 Path를 반환한다. 교차를 진행할 때는 유망한 부분의 cost가 작은 Path a의 유망한 부분을 자식에게 그대로 물려준다 그 후에 연결할 때에는 a와 b의 앞으로 유망한 정도를 간단하게 판단해 더 유망하다고 생각되는 쪽의 index를 넘겨준다. (기준 index로 부터 3개의 Edge를 거쳐 비용이 더욱 짧은쪽을 유망한 것으로 판단한다) 만약 a와 b모두 기준 index다음의 index가 이미 연결이 되었다면 기준 index로부터 가장 가까운 vertex로 연결한다. 모든 vertex들이 연결이 끝났다면 자식 Path를 list에 push 한다.

9. 변이 – TSP를 풀기 위한 유전알고리즘에선 부모의 좋은부분을 가져오는 쪽에 신경을 쓰고 최대한 변이를 사용하지 않으려했으나, 교차만으로는 값의 향상을 일으키기 힘들어 실험 분석 이후에 약간의 변이를 추가했다. 변이는 아래 실험 결과 및 분석에서 그림과 함께 자세하게 설명할 예정이다.

10. 대치 – 교차, 변이가 끝났다면, list안에는 1.5*N개의 염색체들이 있을 것이다. 이 염색채들 중 중복된 것들을 삭제하고 정렬 후에 다시 N개의 염색체가 남을 때 까지 뒤에서(cost가 큰 순서로)삭제한다.

11. 이후 list안에서 cost가 가장 작은 Path를 정답으로 채택한다.

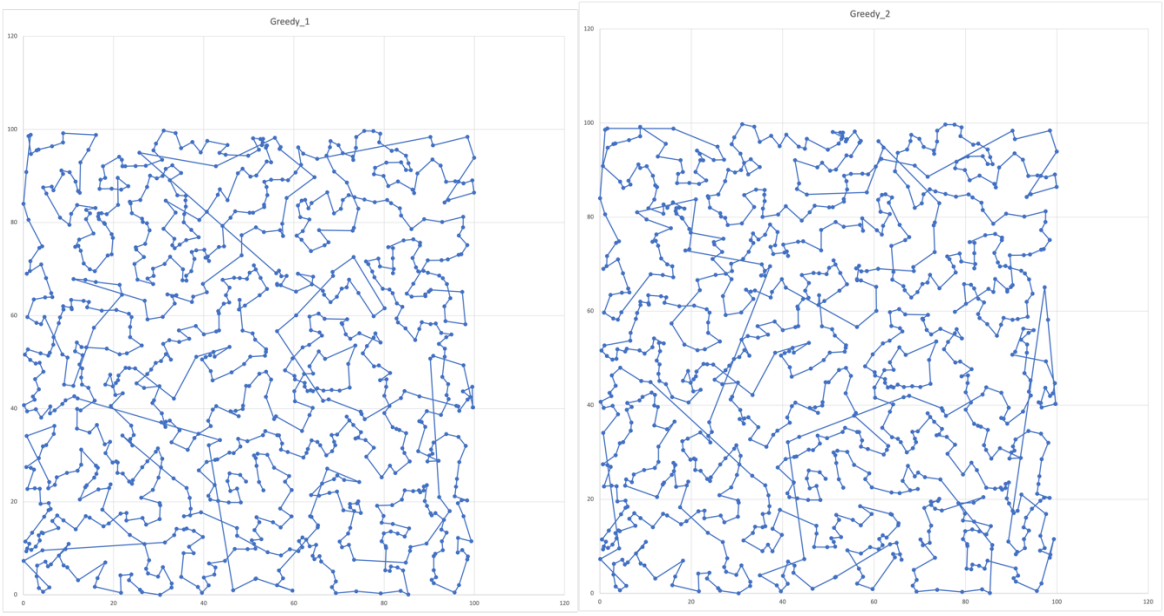
3 실험 결과 및 분석

0. 모든 실험에서 염색체의 갯수 N은 20으로 고정 후에 실험을 진행했지만, 유망한 부분의 길이인 S는 계속해서 바꿔가며 실험을 진행했다.

1. Tree 구조

1) 초기에 염색체를 최적화 하는 방법으로 Greedy알고리즘을 채택함으로써 트리 탐석 기법을 사용했다.

2. 염색체 초기설정



앞서 말한 greedy 방법으로 최적화 한 염색체들 중 2가지의 경우를 가져왔다. 랜덤한 vertex로 부터 시작해 가장 가까운 vertex로 중복을 확인하며 연결했다. 중복을 확인한 방법으로는 경로를 저장할 vector를 (1000, -1)로 초기화 한 후 연결하고 싶은 vertex가 비어 있는지(==-1)를 확인해 가며 비어있을 경우에만 연결을 했다.

3. 초기 구현

1) 초기의 교차 작업은 greedy로 최적화 했던 염색체인 만큼 첫째, 부모의 좋은 경로를 최대한 많이 가져오며 둘째, 우선순위가 뒤로 밀려 길게 연결된 부분을 바꾸기 위한 알고리즘을 구상했다. 첫번째 조건을 해결하기 위해 떠올린 아이디어로 염색체의 유망한 부분을 계산해 이 부분을 그대로 물려주는 방식을 택했다. 이렇게 하면 교차 작업에서 소요시간을 어느정도 줄일 수도 있다고 생각해 초기에는 S의 값을 250으로 설정 후 교차연산을 진행 했다. 두번째 조건은 위에서 확인할 수 있듯이 우선순위가 뒤로 밀리게 되면 주변 vertex들이 전부 선택되어 가장 가까운 도시와 연결했음에도 굉장히 길게 연결되어있는 부분이 몇개 있는 것을 확인할 수 있었다. 그래서 떠올린 아이디어는 vertex를 연결할 때 마다 a와 b의 이후 3개에서 5개정도 edge를 확인해 더욱 짧은 것을 골라 연결했다. 같은 greedy로 염색체를 만들었지만 출발점이 랜덤이라 길게 연결되어있는 부분이 염색체 마다 다르기 때문에 기준 vertex에서 이후에 긴 부분이 나온다면 다른 염색체를 우선으로 연결하도록 했다. A경로와 b경로의 다음 vertex모두 이미 연결이 되어있는 상황이면 greedy로 가장 가까운 vertex를 고르도록 했다.

2) 위에서 언급한 것처럼 초기에는 변이를 넣지 않았다.

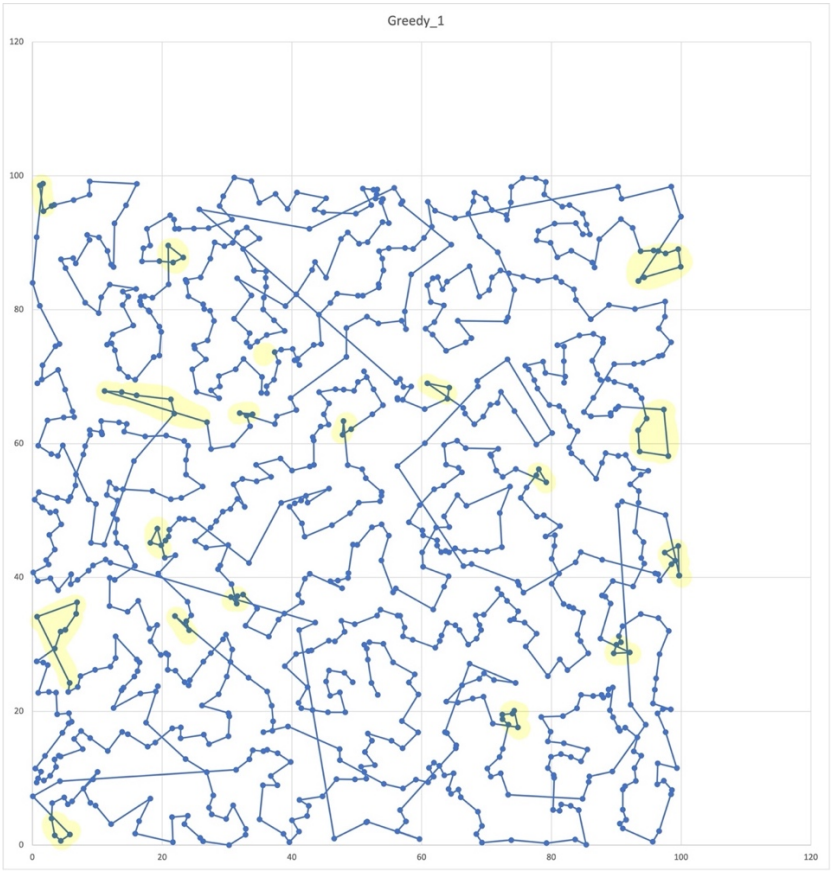
3) 대치는 부모세대의 염색체와 자식세대의 염색체 중 중복 되는 것들을 삭제한 뒤, 최종 비용이 큰 녀석들을 우선으로 삭제했다.

4) 분석 – 이 버전에서의 결과는 굉장히 좋지 않았다. 좋지 않은 정도가 아니라 초기 설정 이후로 값의 변화가 거의, 혹은 전혀 일어나지 않았다. 확인결과 교차연산 직후의 자식세대 염색체들은 대부분 부모세대의 염색체보다 큰 값을 가지고있어 대치과정에서 도태되며, 일정 수준 이후엔 모든 자식들이 도태되고 계속해서 같은 염색체들만 계속 살아남는 것으로 확인되었다.

4. 변이 추가

1) 위와 같은 문제점을 보고 그대로 유전하는 부분인 S의 값이 너무 크다고 생각해 S의 값을 100으로 줄였으며, 또한 변이의 필요성을 느꼈다. 변이 통해 값이 조금이나마 개선이 된다면 염색체들 사이의 순위에 영향을 줄 수 있을 것이고, 유망하다 판단된 부분에도 영향을 줄 수 있을 것이라 생각했기 때문이다.

2) 변이를 위해 떠올린 아이디어는 아래와 같다.



3) 위 그림은 초기에 greedy로 구한 초기 염색체이다. 색칠한 부분을 살펴보면 불필요하게 한 바퀴 꼬여 있는 것을 볼 수 있는데, 변이로 여기 꼬여있는 부분을 풀어 내는것을 목표로 잡았다.

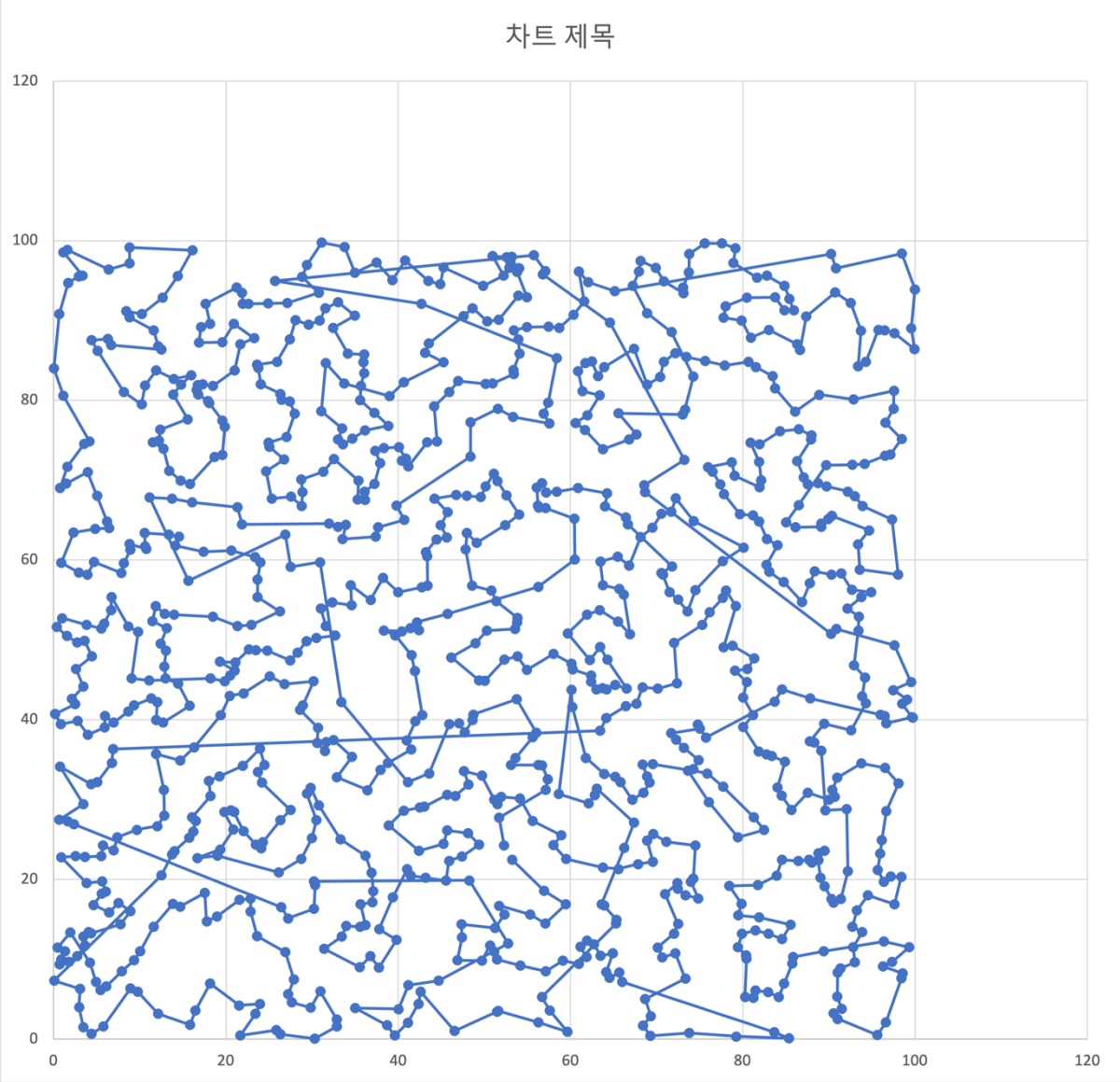
4) 변이는 부모세대와 교차로 만들어진 자식 염색체들 모두에게 적용했다. Path에 저장된 객체중 랜덤으로 한 인덱스를 기준으로 정해 기준 인덱스로 부터 뒤의 6개의 인덱스만 섞어준 뒤 원래 경로의 cost보다 섞은 새로운 경로의 cost가 낮다면 섞은 경로로 업데이트 해 주는 방식을 택했다.

5) 1.5*N개의 염색체에 변이를 적용한 후 대치는 동일하게 이루어 졌다.

6) 분석 – 변이를 추가하니 역시 값의 개선은 이루어졌지만, 개선이 이루어진 빈도가 너무 적고 개선 폭도 미미해서 변이를 추가한 것만으로는 원래의 교차에 영향을 주기 힘들어 보였다.

5. 교차 개선

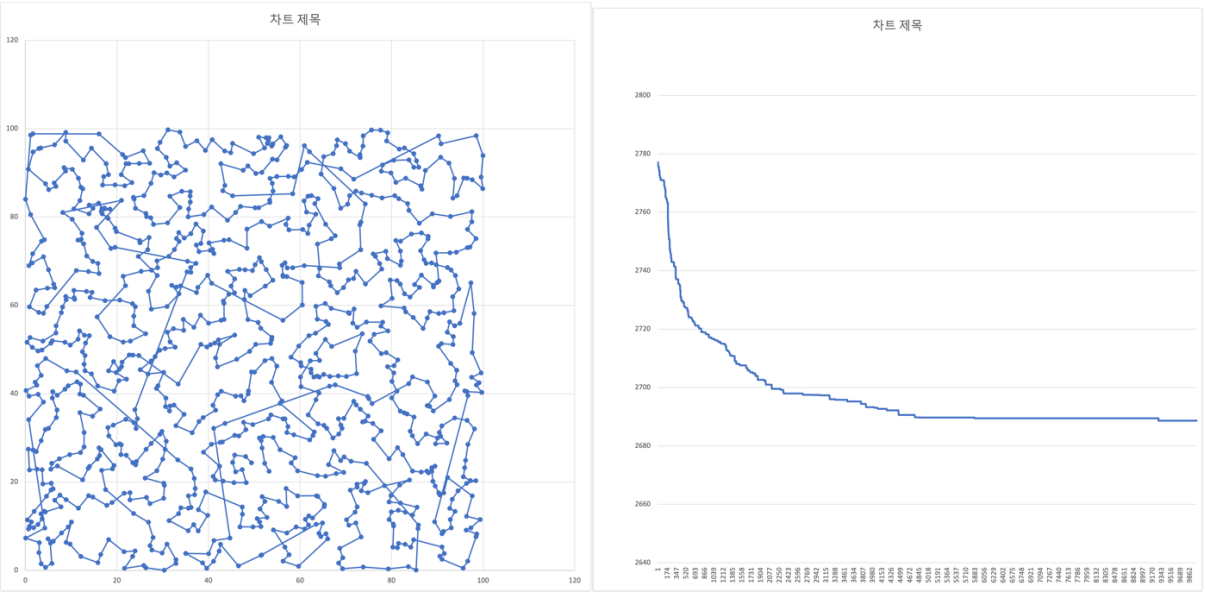
1) 위와 같은 문제점을 보고 다시 한번 S의 값이 너무 크다고 생각해 S의 값을 10으로 대폭 줄였다. 또한, 변이를 추가 한 것만으론 교차에 영향을 주기 힘들어 보였기 때문에 교차나 변이에 다른 방법을 채택해야 최적화를 진행할 수 있을 것이라 판단했다. 아래 사진은 변이를 추가한 후의 한 염색체의 경로이다.



사진을 보면 변이로 꼬인 부분은 거의 풀어진 것을 확인할 수 있었으나 역시 교차가 많이 일어나지 않아 긴 부분은 바뀌지 않고 그대로인 것을 볼 수 있었다. 따라서 변이에 무언가를 추가하는 것 보다는 교차를 고치는 방법이 더욱 최적화 하기 좋다고 판단했다.

2) 이전의 교차연산에서 다음 vertex를 선택하는 기준의 우선순위가
① 유망한 부분을 그대로 유전
② a와 b중 더욱 유망한 vertex를 선택
③ a와 b의 다음 vertex가 모두 이미 선택이 되었다면 가장 가까운 vertex를 선택
순으로 진행 되었다. 여기서 cost의 개선을 이루기 위해선 길게 연결된 부분을 우선적으로 처리해 주어야 한다고 생각해 새로운 조건을 추가하고 우선순위도 약간 변경했다. 변경한 우선순위들은 아래와 같다.
① 유망한 부분을 그대로 유전
② 다음으로 연결할 부분의 cost가 평균보다 2배이상 높으면 선택
③ a와 b중 더욱 유망한 vertex를 선택
④ a와 b의 다음 vertex가 모두 이미 선택이 되었다면 가장 가까운 vertex를 선택
으로 변경해 유전 알고리즘을 진행해 보았다.

3) 분석 – 이번 교차의 변화를 통해 어느정도 의미 있는 성과를 거두었다.



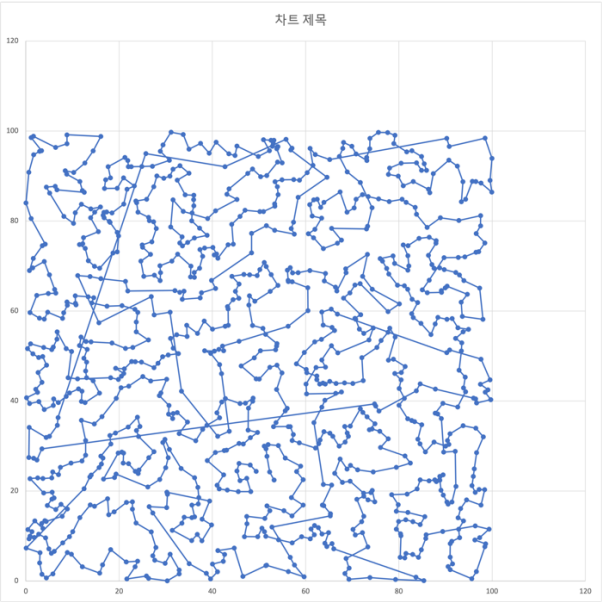
유전 알고리즘을 총 10000회 진행했을 때 얻은 결과이다. 왼쪽은 10000회 이후 염색체 집단에서 cost가 가장 작게 나온 경로이고, 오른쪽은 대치까지 진행한 후 염색체 집단의 가장 작게 나온 cost값의 변화 그래프이다. 최종 경로를 보면 알 수 있지만, 완벽하게 최적화가 된 모습은 아니다. 조금씩 길게 연결되어있는 모습이 보이긴 하지만, 그 전 보단 길이가 짧아졌고, 횟수도 조금 줄어든 것을 확인할 수 있었다. 변화 그래프를 보면 알 수 있듯이 일정 수준을 넘어서면 값의 변화가 급격히 느려지며 정체되는 순간을 발견할 수 있다.

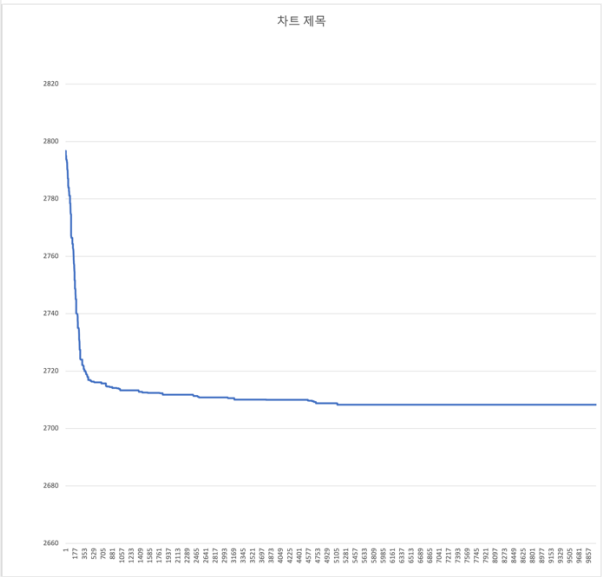
6. 선택 개선

1) 개선이 꾸준히 일어나지 않고 정체 되는 건 크게 두 가지 이유라고 생각했다. 첫째, 교차에서 부모보다 더 나은 자식이 태어나지 않는 점. 둘째, 선택되는 과정에서 같은 부모가 선택되어 같은 자식이 태어나 더이상 변이로도 값의 개선이 힘들어 계속 자식들이 도태

되는 점.

2) 따라서 두 번째 문제점을 먼저 해결하기 위해 선택의 방식을 약간 개선했다. 어차피 N개의 염색체가 모두 선택이 될 것이라면 굳이 (1, 2), (3, 4) … (N-1, N) 순으로 선택을 하지 않아도 될 것이라 생각했다. 더욱 활발히 교차가 이루어 지게 하려면 최대한 다양한 염색체들끼리 만나야 한다고 생각해 N개의 염색체 중 랜덤으로 2개를 선택해 교차를 진행하도록 약간 변경했으며, 아래는 변경 후 결과이다.





3) 최종 결과에는 엄청난 변화는 찾아볼 수 없었지만, cost의 변화 그래프에는 큰 변화가 있었다고 할 수 있다. 한계치까지 도달 하는 속도가 굉장히 빨라진 것을 확인할 수 있었다. 예측한대로 교차가 더욱 활발히 일어나 초반에는 최적화가 굉장히 빠르게 일어나지만, 어느 순간부터는 교차로의 개선은 거의 일어나지 않고, 미미하게 조금씩 개선 되는 것을 보아 한계치 이후에는 교차보단 변이에서 최적화가 일어났다고 추측할 수 있었다. 따라서 납득할만한 최종 정답이 아니지만, 한계치를 만났다는 건 내가 구현한 교차에 한계가 있다는 것으로 해석할 수 있었다.

7. 교차의 개선

1) 결론부터 말하자면 이후의 개선은 실패했다. 최종 결과에 개선이 이루어지지 않아 실패했다는 것이 아닌 구현에 실패했다. 생각했던 아이디어를 구현하려 노력했지만, 시간 부족으로 인해 보고서를 먼저 작성하게 되었다. 아래는 구현하려던 아이디어와 보고서를 작성하며 새롭게 떠오른 아이디어이다.

2) 교차에서 계속 한계점를 맞는 근본적인 이유는 길게 연결된 부분을 개선하지 못해서 였다고 생각했으며, 이 부분을 해결하기 위해 Greedy방법으로 연결을 최소화 하려 했다. 교차 연산에서 다음 vertex를 연결하는 기준과 우선순위에서 greedy는 정말 정말 최후의 방법으로 미루고 최대한 greedy 이전의 방법 안에서 끝낼 수 있도록 하는 방법을 떠올리려 노력했다. 그렇게 구상한 방법은 적당한 시기가 되면 다시 시작점으로 돌아오도록 하는

것이다. 길게 연결되는 이유는 vertex들이 거의 연결되었을 경우 남아있는 vertex를 무작정 연결하기 때문에 길게 연결되는 것이라 생각했고, vector상 끝나는 점이 시작하는 점과 너무 다르기 때문에 마지막에 연결할 때를 최적화 해야겠다고 생각했다. 따라서 적당한 때(vertex가 약 100개정도 남았을 때)가 되면 시작점으로 돌아오도록 구현하고 싶었지만, 결국 실패했다.

3) 보고서를 작성하며 떠올린 새로운 방식은 시작점을 기준으로 양쪽을 동시에 연결 해 나가며 Path a, b기준으로 중간점을 향해 양방향 연결을 시도해 보고 싶다. 길게 연결된 부분은 거의 vector에서 끝에 위치해 있을 확률이 높으니 서로 양방향 탐색을 하며 기준점도 정해 둔다면 어느정도 최적화를 기대해 볼 수 있을 것 같다.

4 결론

유전 알고리즘의 선택, 교차, 변이, 대치 모두 구현에는 성공했지만, 최적화는 실패했다. 특히 교차부분에서 결국에는 Greedy 알고리즘을 사용해 특정 한계점에서 길게 연결된 부분을 줄이는 것에 실패했다.

초기 염색체를 최적화 할 때 사용한 Greedy 방법은 완전 무작위 탐색과 비교했을 때 최적화를 위한 탐색이기 때문에 굳이 최적의 해를 무작위로 탐색할 필요도 없을 뿐더러 최적의 해가 곧 TSP의 해가 되기 때문이다.

비록 보고서를 쓰는 이 시점에서 최적화에는 실패했지만, 새롭게 구상한 방법을 구현하고 분석하는 과정을 통해 납득이 갈 만한 최적화를 꼭 이뤄낼 예정이다.