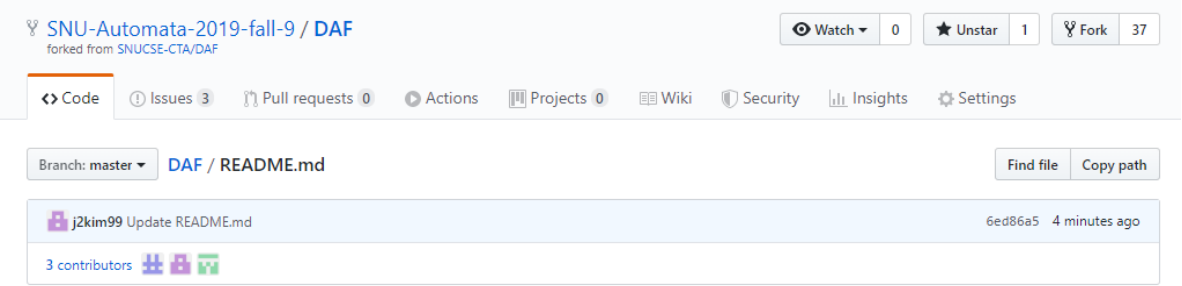
오토마타 이론 HW3 보고서

9조 (오선재, 김종진, 황인휘)



본 과제에서는 적절한 알고리즘을 통해 쿼리 그래프를 DAG로 바꾸어 DAF의 성능을 극대화해야 한다. DAF 알고리즘에서는 DAG의 패스트리로 weak-embedding을 만족하는 후보들을 고려해 탐색하기 때문에 이 후보를 줄이는 것이 중요하다. 이에 우리는 먼저 다음의 몇 가지 아이디어를 고려했다.

1) 엣지 길이 합의 최소화

DAG의 엣지 길이를 i번 노드에서 j번 노드로 가는 엣지는 j-i라고 정의하자. 직관적으로 엣지 길이가 짧을수록 트리를 깊게 하는데 기여도가 클 것이다. 그러면 엣지 길이 합이 최소일 때 좁고 깊은 트리가 만들어져 후보 선정이 까다로워질 것이다. 즉, 거짓 후보가 적어질 것이라 생각했다. 그러나 rooted DAG의 형태를 만족하면서 이러한 트리를 구성하기 까다로워 기각되었다.

2) 엣지 길이 합의 최대화

반대로 엣지 길이 합을 최대로 하면 트리가 얕아져 탐색 과정에서 이득을 볼 수도 있을 것이라 생각했다. 그러나 이도 마찬가지로 rooted DAG의 형태를 만족하면서 이러한 트리를 구성하기 까다로워 기각되었다.

3) 패스트리 중복 정점 최소화

거짓 후보가 생기는 이유의 핵심은 패스트리에 쿼리 그래프의 정점이 중복되어 들어간다는 것이다. 이 중복을 최소화하면 성능을 높일 수 있을 것이라 생각했다. 이를 구현하는 알고리즘은 새 정점이 추가될 때마다 그 앞과 뒤에 새 정점이 추가하는 중복 정점 수를 줄이는 휴리스틱과 최대한 각 점에 들어오는 엣지 수를 줄이기 위해 차수가 높은 정점을 앞쪽에 배열하는 내림차순 정렬의 두 방법을 고려해 보았다.

첫 방식은 패스트리에서 각 정점이 중복되어 나타나는 횟수는 원래 DAG에서 각 정점으로 갈 수 있는 경로 개수임을 생각하면, 차수가 균등하게 분포된 그래프를 생각했을 때 대략적으로 위상정렬시 각 점의 중복되어 나타나는 횟수가 개략적으로 지수함수 형태일 것이라는 추측을 이용하였다. 따라서 새 정점이 추가될 때마다 기존 DAG에서 각 점의 중복 횟수를 구하고 이웃들의 중복 횟수의 로그를 활용하여 중간 위치를 찾아 해당 위치에 넣는 알고리즘을 구상했다. 하지만 이 조건을 만족하면서 DAG 트리가 되도록 하는데 시간이 너무 오래 걸려 기각되었다.

두 번째 방식은 세부적으로 단순히 차수 기준으로 내림차순 정렬하는 것과 차수가 가장 큰

정점을 선택할 때마다 그 정점과 이어진 엣지를 모두 지우고 남은 그래프에서 최대 차수인 정점을 선택하는걸 반복하는 두 방법이 있다. 둘 중 성능상 뒤에 것이 나아 이를 채택했으며 세부 구현은 뒤에 자세히 기술하였다.

4) 패스트리 깊이 최대화

후보를 최대한 까다롭게 하기 위해 패스트리의 깊이가 깊은 것이 유리할 것이다.. 따라서 최장경로를 찾아 1~I에 배열하고 남은 정점의 최장경로를 그 뒤에 배열하는 것을 반복하여 좋은 DAG를 찾을 수 있을 것이라 생각했다. 그런데 최장경로를 찾는 것은 너무 오래 걸리므로 (NP 문제이기에) 근사적으로 DFS로 구한 경로 둘을 이어 붙여 이를 최장경로라고 근사했다. 또한 rooted DAG 형태를 만족하도록 하기 위해 첫 최장경로 이후 나중 최장경로는 그 경로에서 기존 그래프에서 도달 가능한 점 중 가장 앞의 점을 찾아 거기서부터 이어주었다. 이 방식으로 rooted DAG를 빠르게 구할 수는 있었지만 성능상 3)에 밀려 기각되었다.

위의 아이디어 중 실제로 구현에 성공한 아이디어는 3에서 2가지, 4에서 1가지이다. 이를 각각 down1, down2, longestPath라 했을 때 성능은 다음과 같이 나타났다.

	Down1	Down2	longestPath
평균실행시간	110	98	573
평균 재귀 호출	97177	34635	265724
해결한 쿼리	86	87	79

위 3 구현 중 가장 결과가 좋았던 down2의 세부 구현은 다음과 같다.

먼저 기존의 dag.cpp 코드는 main.cpp를 와 따로 실행해야 했기 때문에 두 코드를 합쳐 주어서 컴파일 및 실행이 용이하게 바꾸었다. 그 후, 본격적으로 down2 알고리즘을 구현하기 위해 buildDAG() 함수 부분을 바꾸었다. 먼저, 논문의 root 잡는 방법 자체는 직관적으로 옳은 방법으로 생각되었기에 바꾸지 않았다. 그 후, Degree를 내림차순으로 정렬하기 위한 degree 배열을 따로 잡았다. 그 후 매 시행마다 degree가 최대인 점을 잡고, 그 점에서 연결된 간선들을 지워 주면서 연결된 점들의 degree를 1씩 줄여주는 작업을 반복하였다.

단, 이렇게 시행할 경우 루트에서 도달하지 못하는 정점이 선택될 가능성이 생기기 때문에 rooted dag가 아닐 가능성이 있다. 따라서 추가적인 작업을 하였다. 기존 Visited 배열 말고 checked 배열을 추가로 잡아서 현재까지 순서를 정한 정점들에서 도달 가능한지 판별하였다. 그 후 degree 최대인 점을 판별할 때는 visited가 0이면서 checked가 1인 정점들만 고려하면 된다.