알고리즘 과제 2

2016-16908 심영인

2017-14384 김영인

1. How to run it

해당 알고리즘을 구현하기 위해 프로그래밍 언어는 C++을 사용했고, IDE는 CLion을 이용했다. Readme에 안내된 대로, terminal 창에 build directory를 만들고 여기로 이동해서 cmake ..와 make를 순서대로 입력한 뒤, ./main/program <data graph file> <query graph file> <candidate set file>를 입력하면 embedding을 찾고, 이것이 출력된다.

2. Explain how our program chooses a matching order & how performs back-tracking

1) 본 알고리즘에서 핵심 역할을 수행하는 변수는 (vector) q이다. 이 vector의 요소는 nodeQ의 정보를 가지고 있는데, nodeQ는 해당 node를 가리키는 prev node와 해당 node가 가리키는 next node에 대한 정보를 포함하고 있다. 또한, nextNodes와 prevNodes는 각각 next와 prev에 있는 vertex 정보를 포함한 nodeQ 인스턴스를 가지고 있다. 이와 같은 자료형을 가지고 query graph를 dag 형식으로 만들었다. 이 과정에서 BFS 방식을 이용했는데, 첫번째로 방문할 root node의 경우, factor(v)≡|Candidate set(v)|/(Degree(v)) (v∈query) 와 같이 정의된 factor를 고려하여 이 값이 가장 작은 v를 root node로 선정했다. 이후, 이렇게 만든 dag를 topological sort (q_sort)했고, 이 과정에서 dag 순서는 임의로 지정한다.

2) 다음으로 candidate 정보를 이용해서 q tree를 만든다. q의 모든 원소를 순회하면서 CS 정보를 추출했으며, 이것을 candidates에 저장했고 이것을 내장된 sort함수로 오름차순으로 sort하였다. 또한, nodeQ 타입의 cur 변수를 만들어 여기에는 CS의 next node와 prev node 정보를 저장했다. 여기서 정의한 candidate과 cur은 cost function을 만들 때 사용한다.

3) 매 단계마다 특정 vertex의 Extendable candidates를 고려해줘야 하는 DAF와 달리, 본 알고리즘은 dynamic programming을 이용하여 cost function을 구하고, 이것을 통해 path size order를 구했기 때문에 cost가 낮은 순서대로 matching ordering을 진행한다.

* 각 matching 단계에서는 아래와 같은 방법으로 embedding 조건을 확인한다.

첫번째로, Injective 조건을 확인하기 위해 embedding으로 선정된 node와 이미 embedding으로 확정된 node 사이에 동일한 node가 있는지 확인했고, 만약에 있으면 해당 노드는 embedding이 될 수 없으므로 다음 Candidate node로 넘어가도록 구현했다. (dataVertexVisitInfo를 통해 확인)
두번째로, query 내 두 node 사이에 edge가 있으면, 그들의 embedding에 대응되는 node 사이에

도 edge가 있는지 확인하고자 했으며, 이를 위해, 이미 방문한 query 내 모든 node(visited)들에 대하여 이들의 embedding에 대응되는 각 node와 candidate 사이에 edge가 있는지를 확인했다. 이렇게 2가지 조건을 모두 통과한 candidate만 최종적으로 embedding에 들어갈 수 있게끔 구현했다. (data graph, candidate set, query 정보를 이용해서 확인)