알고리즘 HW2, 3 레포트

컴퓨터공학부 2018-12967 박재문
컴퓨터공학부 2021-15391 신기준

0. Prerequisites

- master branch의 src 디렉토리 내에서, main.cc 파일이 최종 결과물입니다.
- 프로젝트의 root 디렉토리에서 다음 과정을 거쳐서 실행할 수 있습니다.
- cd src
- g++ main.cc -o main -std=c++17
- ./main (graph file) (query file) (candidate set file): 규정에 명시된 프로그램의 실행형태를 따릅니다.
- 만약 헤더인 <bits/stdc++.h>가 환경에 포함되어 있지 않다면, 다음의 헤더를 포함시켜 주세요.

```
#include <iostream>
#include <vector>
#include <stdio.h>
#include <cstdio>
#include <utility>
#include <limits>
#include <queue>
#include <random>
#include <bitset>
```

1. Preprocessing

1) query graph

주어진 query graph의 고정된 root로부터 bfs 방식으로 graph를 탐색하며, 먼저 탐색된 정점 -> 나중에 탐색된 정점으로 향하는 DAG를 구성한다. 또한 이를 바탕으로 2.1에서 소개된 matching order를 계산하는데 사용되는 Weight의 array를 사전에 계산한다.

2) Candidate Set

먼저, Candidate Set들 중 그 크기가 1인 set이 있다면 올바른 embedding은 이들을 무조건적으로 포함한다. 따라서 나머지 candidate set에 이러한 속성을 가지는 정점이 존재한다면 이들을 candidate set에서 제거한다. 또한 만약 이 제거 과정에서 새롭게 크기가 1인 set이 나타난다면 이 과정을 반복한다.

또한 모든 data graph의 vertex들이 candidate set 내에서 몇 번 포함되었는지 나타내는 값인 “중복도(multiplicity)”를 정의한다. 중복도가 낮은 정점들이 backtracking 과정에서 먼저 탐색되도록 함으로써, 어떤 정점의 중복 사용으로 인해 backtracking이 멈추는 과정을 방지하기 위함이다.

중복도가 정의되었다면, candidate set의 정점들을 random하게 섞은 다음 (중복도 / 3)가 큰 순서대로 stable_sort한다. 그러므로 backtracking은 candidate set의 원소들을 순차적으로 탐색하는 형태로 진행되어야 한다.

2. Matching Order

먼저 레포트에서 사용되는 대부분의 용어는 ppt 슬라이드에 설명된 용어와 mental model을 따른다.

1) Extendable vertex에 대한 Matching Order

Matching Order는 기존에 소개된 Path-size ordering을 따른다. candidate set의 각 원소는 weight라는 값을 가지며, 각 extendable vertex마다 $C_M(u)$에 속한 모든 정점 v에 대해 weight의 합이 가장 작은 원소를 다음 탐색 지점으로 정하는 방법이다. 이 계산 시점은 Partial Embedding M에 원소가 추가되거나 제거될 때이다.

어떤 정점의 weight는 DAG 위에서, 그 정점으로부터 시작해 '부모가 1개인 정점'만을 따라 나갈 수 있는 경로의 개수의 상한값이다. 이 값은 partial embedding M에 의존적이지 않은 값이므로, preprocessing 과정에서 계산된 것이다. 이 값이 작다면 직선형 경로가 가장 적게 등장하는 정점을 먼저 탐색하는 셈이므로, space를 빠르게 탐색할 수 있다. extendable한 query graph의 정점 중에서 weight가 가장 작은 값을 빠르게 계산해 내기 위해 Range Min Query를 계산하는 Segment Tree를 하나 관리한다.

2) Candidate set의 Candidate Order

Candidate set에 존재하는 정점들에 대해, 이들의 순서를 특정한 알고리즘에 따라 섞는다. 결국 구해야 하는 값은 query graph를 전부 덮는 embedding이므로, candidate set 중에서 알맞은 candidate를 먼저 탐색하는 작업을 시행하였다. 이는 앞선 "Preprocess"의 2)번 항목에 해당한다.

3. Pruning

query graph에 '방향성'을 부여한 DAG를 만들고 탐색을 진행할 때, extendable한 정점들 중 임의의 것이 선택될 수 있으므로, 만약 재귀적으로 embedding을 찾지 못한 이유와 연관이 없는 정점에 도달할 경우 pruning이 가능하다.

새로운 extendable이 있는 candidate set에 도달하여 partial embedding을 확장시킬 때, query graph의 정점 개수만큼의 길이를 갖는 배열을 선언하고 다음의 작업을 거친다. 아래에서 query graph의 label이 반드시 0부터 매겨짐을 이용하여, 배열의 index로 사용하는 값이 query graph의 label이다.

1) 어떤 candidate가 중복되어 사용되고 있는 경우, 그 정점이 사용되는 matching의 query graph 상에서의 정점 위치를 index로 하여 배열에 1을 표시한다.
2) 어떤 candidate가 $C_M(u)$에 포함되지 않는 경우, DAG에서 이 정점의 부모들의 위치를 index로 하여 배열에 1을 표시한다.
3) 어떤 candidate가 조건을 만족하여 다음 재귀호출이 진행되는 경우, 현재 계산하는 extendable의 index가 재귀호출 함수의 반환값의 배열에 1로 표시되어 있지 않다면 그 배열 전체를 바로 현재 호출된 재귀함수의 반환값으로 한다.
4) 중간에 강제로 반환되지 않은 채로 함수의 끝에 도달하면, 새로 1을 표시하고 있는 배열을 반환한다.
5) 다만 만약 재귀함수의 하위에서 올바른 embedding이 찾아진 경우에는 배열을 초기화하고 0으로 채워진 배열을 반환한다.

여기서 각 배열의 index가 1이라는 것은, 이 index의 정점의 mapping이 잘못되었을 수

있다는 의미를 내포하게 된다. 1)의 경우 중복된 정점이 사용되고 있으므로 현재의 embedding ~ 중복된 정점 사이에 존재하는 정보는 어떻게 되어있든 무관하며, 바로 중복된 정점이 호출되는 함수에 도달하기까지의 재귀호출을 끊을 수 있다. 2)의 경우 현재의 extendable이 결정된 이유는 그 부모 정점들이 전부 M에 포함되어 있기 때문이므로, 다른 정점들의 상태가 현재 정점에서 embedding을 찾을 수 없는 '이유'에 기여하지 않기 때문이다. 3)은 현재 실행되고 있는 함수가 종료될 조건이며, 4), 5)는 1), 2), 3)의 기능이 제대로 작동하기 위한 함수의 정의라고 볼 수 있다.