

Graph Matching Challenge Report

2015-12376 김도훈, 2015-12100 지경준

June 5, 2021

1 Environment and How to run it

과제 repository의 README대로 실행하면 된다.

```
mkdir build
cd build
cmake ..
make
./main/program <data graph file> <query graph file> <candidate set file>
```

2 How Our Program Performs Backtracking

Recursion 방식으로 Backtrack을 구현하였다. pseudo code는 Algorithm 1과 같다. Line5의 *root* <- **first node of DAG**, Line 14의 **Select next *u* in *Query***, 그리고 Line 17의 **Sort candidates according to Vertex ID**부분이 matching order의 핵심이고 다음 섹션에서 자세히 설명한다. 실제 구현에서는 아래 pseudo code에 추가로 다음 path로 선택될 수 있는 vertex *u* (in query graph) 의 개수와 현재 partialEmbedding에 해당하는 Extendable Vertex의 수가 다르면 더이상 진행하지 않도록 최적화하였다.

Algorithm 1 Backtrack Pseudo Code

```
1: if  $M$ 's size ==  $Query$ 's size then
2:   Print  $M$ 
3: else if  $M$  is empty then
4:    $M$ 's size == 0
5:    $root \leftarrow$  first node of DAG
6:   Compute Extendablenodes
7:   for each candidate of  $root$  do
8:      $M[root] \leftarrow candidate$ 
9:     Mark candidate as visited
10:    Backtrack( $M$ )
11:    Mark candidate as unvisited
12:   end for
13: else
14:   Select next  $u$  in  $Query$ 
15:   Compute Extendablenodes
16:    $M' \leftarrow M$ 
17:   Sort candidates according to Vertex ID
18:   for each  $v$  in candidates do
19:     if  $v$  is unvisited then
20:        $M'[u] \leftarrow v$ 
21:       Mark  $v$  as visited
22:       Backtrack( $M'$ )
23:       Mark  $v$  as unvisited
24:     end if
25:   end for
26: end if
```

3 How Our Program Chooses a Matching Order

3.1 How to Choose Root

Algorithm 1의 Line5 `root <- first node of DAG`에 해당하는 부분이다. 코드에는 `getTopologicVector()`, `getNextTopologicElem()` 라는 이름으로 구현되어 있다. Algorithm 2 가 `getTopologicVector()`의 pseudo code이다. `getNextTopologicElem()` 함수가 어떻게 정렬할지를 결정하는데 노드의 label Frequency와 Candidate Set의 size로 정렬하였다.

Algorithm 2 BuildDAG

```
1: Select  $r$  whose label frequency is minimum
2: Push  $r$  to topologicVector
3: while Unvisited vertex exists do
4:    $S \leftarrow$  Vertices in unvisited connected to visited
5:    $u \leftarrow$  getNextTopologicElem( $S$ )
6:   Mark  $u$  as visited
7:   Push  $u$  to topologicVector
8: end while
9: Return topologicVector
```

3.2 How to Choose Next Node in Query Graph

Algorithm 1의 Line 14, `Select next u in Query`에 해당하는 부분이다. partialEmbedding에 node v 의 parent가 들어있고 그 parent에 matching된 node가 v 와 neighbor인지를 판단하여 candidate에서 v 를 걸러준다. 이렇게 걸러진 candidate들의 수가 가장 작은 node u (in query graph)를 선택한다. Algorithm 3참고.

Algorithm 3 FilterCandidate

```
1: for extendableVertex in Extendable do
2:   for  $v$  in AllCandidate do
3:     if  $v$  and  $M[v's\ parent]$  are not neighbor then
4:       Filter  $v$  from Candidate
5:     end if
6:   end for
7: end for
```

Test Case	number of found	Execution Time(sec)
hprd_n1	970	0.083769
hprd_n3	908545	9.04493
hprd_n5	32833	1.03159

Table 1: Some of Test Results

3.3 How to Choose Next Node in Candidate

Algorithm 1의 Line 17의 **Sort candidates according to VertexID**부분이다. *Candidate*들을 Vertex ID로 내림차순 정렬하였다. 간단히 내림차순 정렬이므로 pseudo code는 생략한다.

4 Test Result

hprd_n 케이스에 대해서 테스트를 해본 결과 table 1와 같은 결과가 나왔다. 빠른 print를 위해 `std::ios_base::sync_with_stdio(false);`를 추가하여 iostream과 stdio 간의 동기화를 막았다. 출력시 `printf`와 `std::cout`을 번갈아 사용하는 것이 아니라 `std::cout`만 사용하였기 때문에 출력 결과에 영향은 없었다. 나머지 케이스는 출력 결과가 매우 많아 총 소요시간을 측정해보지는 못했지만, shell script를 통해 `yeast_s3`, `yeast_s5`, `yeast_8`을 제외하고는 모두 1초 이내에 결과를 출력하는 것을 확인할 수 있었다.

5 Appendix

Figure 1는 Final Version 도달하기 전까지 시도해본 방법 중 일부이다. label frequency, degree, path size ordering 등을 시도해보았다.

	First Version	wo shuffle	top freq/deg	data Degree decision	data freq u	datafreq u no vertice weight	further freq	weight to height	query label freq	query label freq root change	Final Version
hprd_n1	o	o	o	o	o	o	o	o	o	o	o
hprd_n3	o	o	o	o	o	o	o	o	o	o	o
hprd_n5	o	o	o	o	o	o	o	o	o	o	o
hprd_n8	o	o	o	o	o	o	o	o	o	o	o
hprd_s1	o	o	o	o	o	o	o	o	o	o	o
hprd_s3	o	o	o	o	o	o	o	o	o	o	o
hprd_s5	o	o	o	o	o	o	o	o	o	o	o
hprd_s8	o	o	o	x	o	o	o	o	o	o	o
human_n1	o	o	o	o	o	o	o	o	o	o	o
human_n3	x	o	o	o	o	x	o	o	o	o	o
human_n5	o	o	o	o	o	o	o	o	o	o	o
human_n8	o	o	x	o	o	o	o	o	o	o	o
human_s1	o	o	o	o	o	o	o	o	o	o	o
human_s3	o	x	x	x	x	o	x	x	x	x	o
human_s5	o	o	o	o	o	o	o	o	o	o	o
human_s8	x	x	x	x	x	x	x	x	x	x	o
				2	2	2					
yeast_n1	x	o	o	o	o	o	o	o	o	o	o
yeast_n3	x	x	x	x	x	x	x	x	x	x	o
yeast_n5	o	o	o	x	o	o	o	o	o	o	o
yeast_n8	o	x	o	x	x	o	x	x	x	x	o
yeast_s1	x	x	x	x	x	x	x	x	o	o	o
yeast_s3	x	x	x	x	x	x	x	x	x	x	x
yeast_s5	x	x	x	x	x	x	x	x	x	x	x
yeast_s8	x	x	x	x	x	x	x	x	x	x	x
# of X	8	8	8	10	8	7	8	8	7	7	3

Figure 1: