<Graph Pattern Matching Challenge 보고서>

2016-14460 김윤수, 2016-10475 강병우

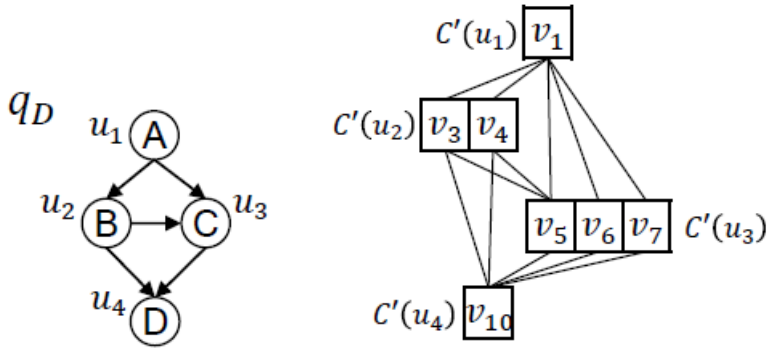
1. Backtracking 수행 방법

수업시간에 소개된 recursive하게 partial embedding을 늘리는 방식을 채택하였다. Backtracker class 안에 있는 backtrack 함수는 partial_embedding, 아직 partial embedding에 포함되지 않은 query vertex들을 담은 unvisited_query_vertices, 그리고 recursion의 깊이를 알려주는 recursion_depth을 변수로 받는다.

주어진 Partial embedding에 대해서 unvisited_query_vertices 중에 parent(2번에서 설명할 개념)가 모두 partial embedding에 속한 vertex(extendable_vertices)를 구한다. 그 중 matching order에 따라 다음 연결할 query vertex u가 정해지면, u의 candidate set 중에 partial embedding에 없고, u의 parent의 partial embedding에 모두 연결된 extendable candidate v_i들을 구한다. 그 후 unvisited_query_vertices에서 u를 지워주며, for loop을 사용하여 각각의 extendable candidate v_i을 u에 mapping한 partial embedding을 생성해주고, recursion_depth를 1 늘린 변수로 backtracking을 recursive하게 불러준다.

Unvisited_query_vertices에 더 이상 vertex가 남아있지 않은 경우는 모든 query vertex가 data vertex와 mapping 된 경우이므로 이 때 embedding을 출력해준다. 그렇지 않은 이상 backtrack을 recursive하게 계속 불러준다.

<예시>



- Query vertex = {u1, u2, u3, u4}
- u1이 v1와 mapping 된 partial embedding에 대해서 u2가 matching order에 따라 다음에 연결될 query vertex인 경우
- u2의 CS: {v3, v4} 중 u2의 parent u1에 mapping된 v1과 연결된 extendable candidate는

{v3, v4}

- unvisited query vertices에서 u2을 빼준 뒤 recursion depth를 1 늘려주고
- partial_embedding에 (u2, v3)을 추가한 채로 backtrack을 한번 불러주고
- partial_embedding에 (u2, v4)을 추가한 채로 backtrack을 한번 불러준다.
- Unvisited query vertices에 더 이상 vertex가 없을 때, partial embedding을 출력한다.

이렇게 구현한 결과 어떤 query graph에 대해서는 1분내에 모든 embedding을 찾는 반면 어떤 query graph에 대해서는 embedding을 1000초가 지나도 찾지 못하는 것을 발견하였다. 이는 embedding이 없는 candidate에 대해서도 우리 프로그램이 recursive call을 진행하기 때문이라고 판단하여 간단한 pruning을 구현하였다.

Recursive call을 한 횟수를 세면서 이 call의 횟수가 특정 수 N을 넘었는데도 출력된 embedding이 없는 경우, 진행되고 있는 backtrack을 그냥 return하여 recursion depth를 줄이는 방법을 사용하였다. 또한 이런 상황이 1번 발생하면 1단계 올라가고, 2번 발생하면 2단계, 3번 발생하면 3단계 올라가게끔 하였다. 이 때 N은 휴리스틱을 이용해 (search tree에서 앞으로 찾아야 하는 path의 개수의 upper bound)$^{0.16}$으로 adaptive하게 두었으며, 최소값을 1만, 최대값을 100만으로 설정하였다.

<예시>

- Query vertex u1에 연결될 수 있는 data vertex가 v1, v2, u2에 연결될 수 있는 data vertex가 v3, v4이라고 할 때
- backtrack(partial_embedding={(u1,v1),(u2,v3)}, unvisited=(u3, u4), recursion depth=2)을 불렀는데 아직 찾은 embedding이 없고 recursive call의 횟수가 N을 처음 넘겼으면 그냥 return하고 backtrack(partial_embedding={(u1,v1),(u2,v4)}, unvisited=(u3, u4), recursion depth=2)을 부른다.
- backtrack(partial_embedding={(u1,v1),(u2,v3)}, unvisited=(u3, u4), recursion depth=2)을 불렀는데 아직 찾은 embedding이 없고 recursive call의 횟수가 N을 두번째로 넘겼으면 2번 그냥 return하고 backtrack(partial_embedding={(u1,v2),}, unvisited=(u2, u3, u4), recursion depth=1)을 부른다.

이런 조치를 통해 한 곳에 알고리즘이 막히는 것을 방지할 수 있었으며 기존에는 제 시간에 못 찾던 embedding을 찾을 수 있게 되었다.

2. Matching Order 구현 방법

기본적인 아이디어는 수업 ppt에서 얻었다. 우선 주어진 query graph을 DAG로 바꿔주었다. 이 때 DAG의 root는 가장 많은 vertex으로 뻗어나가면서 본인과 mapping이 가능한 candidate의 개수는 가장 적은 vertex으로 골라주었다. ($\frac{|CS(u)|}{deg_q(u)}$가 최소인 u) 그후 BFS 알고리즘을 사용하여 edge의 방향을 설정해주었다. Edge (u,v)에 대해 u에 먼저 도달하고 v에 나중에 도달한다면 u에서 v로 향하게끔 설정해주었다. 실제 코드에서는 u의 child가 v이고, v의 parent가 u라고 설정해주는 식으로 구현하였다.

그 후 backtracking을 할 때마다 다음과 같은 과정을 거쳐 다음 matching order을 찾았다. 이는 수업시간에 소개된 Adaptive Matching Order 중 Candidate Size Order의 아이디어를 이용한 것이다.

1. 현재 query DAG에서 아직 partial embedding에 들어가 있지 않은 vertex 중 parent가 모두 partial embedding에 있는 extendable_vertices을 찾는다. (Partial Embedding에 아무것도 들어가 있지 않을 때는 extendable_vertices는 DAG의 root뿐이다.)
2. extendable_vertices 내의 각각 vertex u의 Candidate Set 중 partial embedding에 없고 u의 parent에 mapping 된 data vertices와 모두 연결된 data vertices, extendable_candidates를 찾는다.
3. extendable_vertices 중 가장 extendable_candidate의 크기가 작은 u가 다음 matching order에 해당한다.

우리 프로그램은 모든 extendable_candidate에 대해 backtrack을 recursive call하기에 extendable_candidate의 크기가 가장 작은 extendable vertex을 뽑으면 recursive call의 횟수를 줄일 수 있고 이는 결국 embedding을 더 빨리 찾게 해준다. 맨 처음 query graph를 DAG로 바꿀 때 root를 $\frac{|CS(u)|}{deg_q(u)}$ 가 최소가 되는 u로 잡은 것도 root에서의 recursive call을 줄이면서 동시에 다음 matching order을 고를 때 최대한 많은 선택지 중에서 가장 extendable_candidate의 수가 작은 extendable_vertex을 고르기 위함이다. (root의 degree가 크면 root 다음 mapping할 수 있는 extendable vertex가 많기 때문이다)

3. Backtracker Class에 대한 간단한 설명

프로그램은 Python의 Class을 사용하여 작성하였다. Backtracker라는 Class를 하나 설정하여 주어진 path에서 데이터를 읽어오는 등 필요한 기능들을 정의해주었으며, 마지막 matching order와 같은 부분은 BacktrackerV1~10 이라는 자식 Class을 생성하여 따로 구현해주었다. 아래는 Backtracker Class에 있는 함수 중 몇 가지 중요한 함수에 대한 설명이다.

- select_root: query graph와 cs를 받아서 query graph의 vertex u에 대해 $\frac{u의\ CS\ 크기}{u의\ degree}$가 가장 작은 u를 돌려준다.
- build_dag: select_root 함수를 통해 찾은 root를 이용하여 BFS를 진행한다. 그 결과 먼저 방문된 u와 나중에 방문된 v 사이 edge가 있으면 u는 v의 parent가 되고, v는 u의 child가 된다.
- parents_all_in_partial_embedding: query vertex u와 partial_embedding을 받아서 DAG을 만들 때 설정한 u의 parent가 모두 partial embedding에 있는지 확인해준다.
- candidate_connected_to_all_parents: query graph의 vertex인 u, data graph의 vertex인 v, partial embedding을 받아 u의 parent와 연결된 partial embedding의 data vertex들이 v와 연결되어 있는지 확인해준다.
- return_condition: 1분 이상 시간이 넘어가거나 최대 embedding 개수를 넘어가면 프로그램이 종료되게 해준다. 그 외에도 unvisited_query_vertices가 없으면 partial_embedding을 출력해준다.
- find_extendable_vertices: unvisited_query_vertices와 partial_embedding을 받아, 아직 mapping 안된 query vertices 중 어떤 것의 parent가 모두 partial embedding에 있는지 찾아준다.
- find_extendable_candidates: query vertex u와 partial embedding을 받아 u의 CS 중, partial embedding에 없고 u의 parent에 mapping 된 data vertex들과 모두 연결된 candidate들을 찾아준다.
- check_prune: (search tree에서 앞으로 찾아야 하는 path 개수의 upper bound)^0.16을 N이라 뒀을 때, recursion call이 N만큼 이루어졌는데 찾은 embedding이 없다면 recursion call을 몇 단계 위로 올린다.

4. 시도한 방법들

최종 프로그램은 여러 시행착오 끝에 완성된 10번째 버전이다. 다음은 몇 가지 이전 버전들에 대한 간단한 설명이다.

V1: Matching order을 설정할 때 extendable candidate의 개수를 다 계산해주지 않고 그냥 CS의 크기를 비교하였다. 이는 계산 overhead를 줄일 수 있는 방법이지만 matching order가 너무 단순하여 선택되지 않았다.

V2: 현재 V10와 같은 Matching order을 사용하였지만 check_prune 함수가 포함되지 않은 버전이다.

V3: matching order 없이 처음 찾아진 extendable vertex를 선택하는 버전이다.

V4: Matching order로 extendable_candidate의 개수가 가장 큰 vertex을 선택하게끔 해본 버전이다.

V5: extendable_candidate 개수뿐만 아니라 각각 extendable_candidate이 가지는 CS의 크기까지 고려해 Matching order을 선택한 버전이다. 성능개선이 딱히 없어 선택하지 않았다.

V6: 이전에는 extendable candidate들에 대해 backtrack을 하는 순서를 정해주지 않았으므로 extendable_candidate을 추가로 data graph에서 degree를 기준으로 order한 버전이다. 특별한 성능개선은 없었다.

V7: Matching order을 adaptive 하게 계산하는 과정에서 발생하는 overhead를 없애기 위해 candidate size order에 따라 먼저 정렬하고 backtracking을 진행한 버전. 이 또한 성능개선이 특별히 없었다.

V8: partial embedding에 있는 data vertex를 제외한 candidate set의 크기가 가장 작은 query vertex를 다음 vertex로 선택하는 버전이다.

V9: V7에 pruning을 적용한 버전이다.

V10: V2에 pruning을 적용한 버전이다.

5. 프로그램 실행 환경 및 방법

Python 3.8를 사용하여 작성하였으나 채점환경인 Python 3.7.10에서도 문제없이 돌아갈 것으로 예상되며 다른 컴파일러는 필요없다.

<실행방법>

① cmd를 실행하여 프로그램(main.py, backtracker.py)가 들어있는 디렉터리로 들어간다.

② python main.py --data_path <data graph path> --query_path <query graph path> --cs_path <candidate set path>

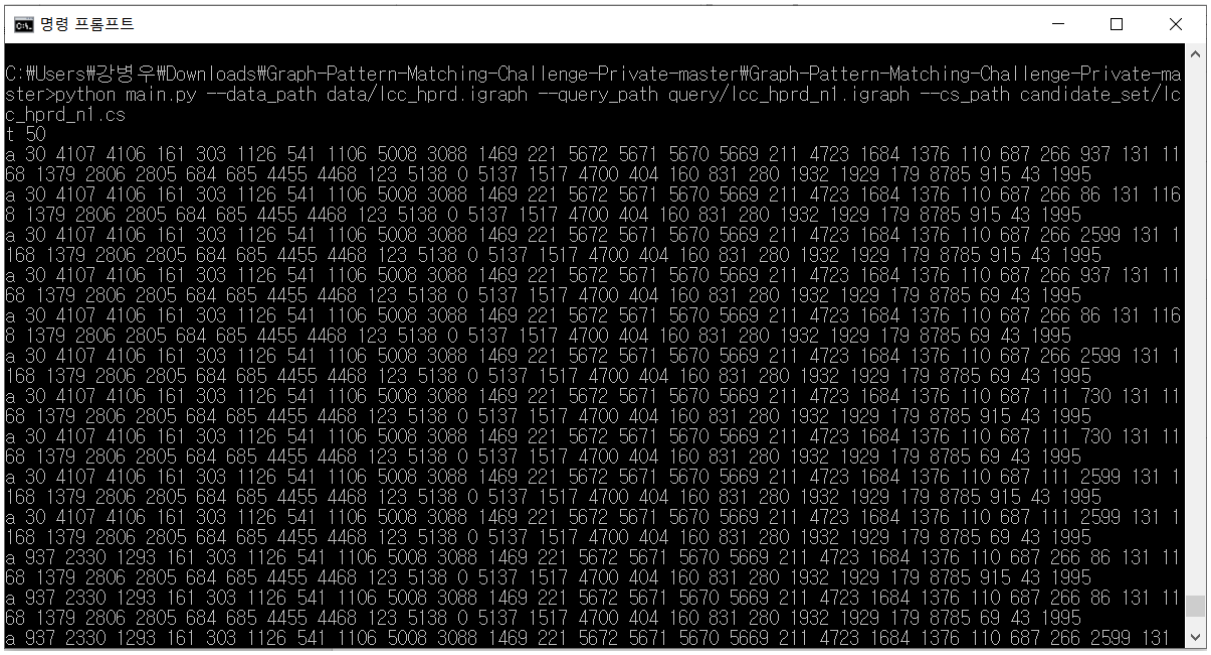
을 실행시킨다. (README 참조)

python main.py --data_path data/lcc_hprd.igraph --query_path query/lcc_hprd_n1.igraph --cs_path candidate_set/lcc_hprd_n1.cs

<실행 예시>


명령 프롬프트
C:₩Users₩강병우>cd C:₩Users₩강병우₩Downloads₩Graph-Pattern-Matching-Challenge-Private-master₩Graph-Pattern-Matching-Challenge-Private-master
C:₩Users₩강병우₩Downloads₩Graph-Pattern-Matching-Challenge-Private-master₩Graph-Pattern-Matching-Challenge-Private-master>python main.py --data_path data/lcc_hprd.igraph --query_path query/lcc_hprd_n1.igraph --cs_path candidate_set/lcc_hprd_n1.cs

<실행 결과 예시>


명령 프롬프트
C:₩Users₩강병우₩Downloads₩Graph-Pattern-Matching-Challenge-Private-master₩Graph-Pattern-Matching-Challenge-Private-master>python main.py --data_path data/lcc_hprd.igraph --query_path query/lcc_hprd_n1.igraph --cs_path candidate_set/lcc_hprd_n1.cs
t 50
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 937 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 86 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 937 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 86 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 730 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 730 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 30 4107 4106 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 937 2330 1293 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 86 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 915 43 1995
a 937 2330 1293 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 86 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 831 280 1932 1929 179 8785 69 43 1995
a 937 2330 1293 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 2599 131