Algorithm HW - GraphPatternMatchingChallenge

원자핵공학과 2017-11655 전명섭
자유전공학부 2019-18841 김시연

1. 프로그램이 선택한 Matching Order

- **Candidate-size order**

- Select vertex u in query graph such that $|C_M(u)|$ is minimum
- 프로그램 코드에서는 `extendable_cs[u].size()` 가 최소인 vertex를 선택합니다.
- 프로그램에서 모든 extendable vertex들은 candidate-size order를 따르는 list인 queue(구현 과정에서 queue의 자료구조로 priority queue를 사용하였지만, list 구현이 더 성능이 좋았기 때문에 list로 변경되었음)에 저장됩니다.

2. 프로그램 구조

- `main/main.cc`: pattern matching을 수행하는 프로그램 program의 소스 코드입니다.
- `src/common.h`, `src/candidate_set.cc`, `src/graph.cc`, `include/candidate_set.cc`, `include/graph.h`: challenge에서 주어진 candidate set과 graph를 다루는 라이브러리입니다. `src/common.h`에 디버깅을 위한 간단한 매크로와 함수를 추가하였습니다.
- `src/dag.cc`, `src/dag.h`: DAG를 다루는 라이브러리로, `src/graph.cc`와 `include/graph.h`를 참고하여 만들었습니다. `Dag` 클래스의 생성자는 query graph q를 바탕으로 DAG q_D를 생성합니다. $\min_{u \in V(q)} |C(u)|/\deg_q(u)$를 root node로 하고 BFS로 query graph를 탐색하며 먼저 방문한 vertex에서 나중에 방문한 vertex의 방향으로 query graph의 모든 edge들에 방향을 배정합니다. 서로 이웃하는 vertex들에 대해 먼저 방문한 vertex를 parent, 나중에 방문한 vertex를 child로 정의하였습니다. q_D 의 adjacency list(`Dag::child_adj_list`)와 q_D^{-1} 의 adjacency list(`Dag::parent_adj_list`)에 대한 정보를 `Dag` 클래스에 담도록 하였습니다.
- `checker/checker.cc`, `include/checker.h`: program의 출력이 맞는 embedding인지 확인하는 프로그램 checker의 소스 코드입니다.

- src/checkergraph.cc, include/checkergraph.h: checker에서 사용하는 graph를 다루는 라이브러리로, src/graph.cc와 include/checkergraph.h를 바탕으로 만들었습니다. checker에서는 주어진 data graph를 parse할 때 모든 edge들의 list를 만들어야 합니다. 하지만 이 과정을 원래 graph 라이브러리에 구현하면 program의 성능에 영향을 줄 수 있으므로 program과 checker에서 사용하는 graph 라이브러리를 분리하였습니다.
- checker_ex, test: checker와 program의 정확성 확인과 디버깅에 사용하기 위해 만든 샘플 입력입니다.
- to_json.pl, show_graph.py: igraph 파일에 담긴 그래프를 시각화하는 데 사용된 스크립트입니다.

3. 프로그램이 Backtracking 을 수행하는 방법

- search_stack에는 현재까지 mapping된 Vertex들이 방문한 시간 순서대로 들어있습니다. Backtrack::PrintAllMatches의 가장 바깥쪽 while 루프를 돌며 search_stack을 업데이트하고, search_stack이 비면 프로그램을 종료합니다.
- while 루프를 시작할 때 search_stack이 꽉 차 있으면 마지막으로 방문한 query vertex에 대한 방문하지 않은 extendable candidate들을 하나씩 대입하여 해당하는 embedding들을 출력합니다. 10만개 이상 출력되면 프로그램을 종료합니다.
- Backtrack::PrintAllMatches는 branch-and-bound의 형식을 따르고 있으며, while 루프를 한 번 돌 때마다 최대한 많이 bound했다가 1회 branch(다음 매핑과 extendable children 찾기)합니다.

- 초반 작업

- 주어진 query graph query로부터 DAG dag를 생성합니다.
- u는 dag의 루트로 초기화합니다. search_stack에 u를 넣고, extendable_cs[u]에 모든 candidate vertex들을 넣어 놓고 시작합니다.

- **Bound** (위로 올라가기 – 내려가서 했던 작업들을 돌려 놓기)

- 먼저 현재 query vertex의 child vertex들에 대하여, 각 vertex의 extendable candidate들을 모두 삭제합니다.
- 다음으로 queue의 원소들 중 extendable_cs가 비어 있는 원소들을 삭제합니다.
- 만약 extendable_cs에 아직 branch를 수행하지 않은 extendable candidate가 남아 있다면 bound 과정은 여기에서 종료됩니다. extendable candidate가 남아 있지 않다면, 현재 query vertex에 대한 탐색을 종료해야 합니다.
- 먼저 현재 query vertex에 mapping된 vertex를 미방문 상태로 돌려 놓고, mapping 상태를 초기화합니다.
- 다음으로 search_stack에서 이전 vertex를 찾아 u를 이전 vertex로 변경한 뒤 bound를 다시 수행합니다.

- **Branch** (아래로 내려가기)

- Vertex u의 mapping 대상으로 data graph의 vertex v를 선택했다면, for문을 돌면서 u의 child vertex들이 extendable한지 검사합니다. vertex u의 각 child는 자신의 모든 parent vertex가 mapping된 상태여야 extendable vertex가 될 수 있습니다.
- u의 child vertex child가 extendable vertex라면, extendable_cs[child] (C_M(child))를 계산합니다. cs 속 child의 candidate vertex가 extendable하려면, child의 모든 parent 노드들에 대해 해당 vertex가 mapping된 data vertex와 candidate vertex 사이에 edge가 존재해야 합니다. 또한 extendable candidate의 정의에 따르면 방문된 vertex도 extendable할 수 있지만, 해당 vertex는 child의 embedding이 될 수 없으므로 아직 방문되지 않은 vertex만 extendable_cs[child]에 포함시켰습니다.
- 만약 어떤 child가 extendable vertex임에도 extendable candidate가 존재하지 않는다면, 현재 query vertex에는 v를 mapping할 수 없습니다. 따라서 지금까지 만들어 둔 C_M(child) 를 초기화하고 다음 vertex 후보로 넘어갑니다. 이 경우가 아니라면, child를 extendable_children에 추가합니다.
- 모든 Children 검사가 끝나면 extendable_children의 원소들을 queue에 집어넣습니다.
- queue에서 가장 적합한 원소를 꺼내서 해당 원소로 u의 값을 변경하고 search_stack에 u를 넣은 뒤 루프를 다시 실행합니다.

4. 프로그램을 돌리는 환경/돌리는 방법

언어: C++11

IDE: VS Code

How to build: VS Code의 C/C++ 패키지를 사용하거나, repository의 최상단 디렉터리에서 터미널에 다음과 같이 입력합니다.

```
$ cmake -B build
$ make -C build
```

How to run(example) : 빌드 이후 터미널에 다음과 같이 입력합니다.

```
PS C:\Users\Owner\Dev\cpp\GraphPatternMatchingChallenge> build\main\program.exe data\lcc_hprd.igraph
query\lcc_hprd_n1.igraph candidate_set\lcc_hprd_n1.cs > output.txt
```

```
> build\main\program.exe data\lcc_hprd.igraph query\lcc_hprd_n1.igraph
candidate_set\lcc_hprd_n1.cs > output.txt
```

- output.txt에 data 그래프 lcc_hprd.igraph와 query 그래프 lcc_hprd_n1.igraph에 대해 subgraph matching을 수행한 결과가 출력됩니다.