알고리즘 Graph-Matching Challenge 과제 보고서

컴퓨터공학부

2019-10821 박범규

2019-13914 정다운

1. Matching Order

Search tree에서 다음 노드를 정하는 matching order는 'candidate-size order'와 'first fit'을 같이 사용하였다. 우선 candidate size가 1인 node를 우선 먼저 연결한다. 이를 통해 결과가 항상 바뀌지 않는 노드를 미리 처리함으로써 backtracking의 단계가 이전으로 돌아갈 때 너무 깊게 돌아가지 않도록 방지한다. 그 후에는 index 순서대로 돌며 'first fit' 즉, 가장 먼저 매칭되는 노드를 먼저 연결한다. 다양한 방법을 시도해보았지만, 이 방법이 가장 효과적으로 매칭이 되어 이 방법을 선택하게 되었다.

2. Backtracking

Backtracking은 조건을 검사하고 candidate을 선택한 후, 다음 노드를 선택해 방문하는 방식으로, 재귀 함수 Backtrack을 만들어 수행했다. candidate을 선택한 후, 다른 노드와 겹치거나 query에서는 이어졌는데 data 상으로는 이어지지 않았다면 조건 위배로 더 이상 경로를 탐색하지 않는 기초적인 pruning을 사용했다.

구체적인 구현은 다음과 같다.

Function Backtrack:

list: query의 모든 노드에 대해, 선택한 candidate를 저장하는 배열.

check: query의 모든 노드에 대해, 방문 여부를 저장하는 배열

level: 현재 진행중인 Backtrack의 깊이를 나타내는 변수. 현재 list에 들어있는 노드의 개수 와도 동일하다.

Input: data, query, cs, starting vertex v

for each candidate i in candidate set of v:

```
    //candidate i가 조건에 맞는지 검사
    if i is not connected as it should be or i is already in list:
        continue
    //조건에 맞다면 방문 흔적 남기기
    list[v] = i
    check[v] = true
    //모든 노드를 다 돌았다면 출력 후 리턴
    if level == query:
        print out every candidates in list
        return
    //다음 노드 선택 후 방문
    v' = select next vertex to visit
    Backtrack(data, query, cs, v')
//현재 노드의 모든 candidate을 검사했다면 방문 흔적 지우기
list[v] = -1
check[v] = false
return
```

3. Environment

1)컴파일러: gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0

2)컴파일 방법:

```
cd build
make
```

3)실행 방법:

```
cd build
```

./main/program <data graph file> <query graph file> <candidate set file>