공학석사학위논문

최대 이분 매칭 기반의 가지치기를 활용한 부분 그래프 동형 알고리즘 성능 향상

Improving Subgraph Isomorphism with Pruning by Maximum Bipartite Matching

2021 년 2 월

서울대학교 대학원

컴퓨터공학부

최 윤 영

최대 이분 매칭 기반의 가지치기를 활용한 부분 그래프 동형 알고리즘 성능 향상

Improving Subgraph Isomorphism with Pruning by Maximum Bipartite Matching

지도교수 박 근 수

이 논문을 공학석사 학위논문으로 제출함

2020 년 11 월

서울대학교 대학원

컴퓨터공학부

최 윤 영

최윤영의 석사 학위논문을 인준함

2020 년 12 월

위 원 장 문 병 로 (인)

부위원장 박 근 수 (인)

위 원 Srinivasa Rao Satti (인)

요약

대형 그래프에 대한 분석은 최근 소셜 네트워크, 생물 정보학, 화학 등 다양한 분야에서 점차 중요해지고 있다. 그래프 분석에서 가장 핵심적인 문제 중 하나로 부분 그래프 동형 문제 (subgraph isomorphism) 가 있다.

부분 동형 그래프 문제를 backtracking 활용하여 해결하는 많은 알고리즘이 제시되어왔다. 하지만 최악의 경우 지수적인 시간 복잡도를 가지는 backtracking 특성상 여전히 특정 입력에서는 답을 찾기 위해서 지나치게 긴 시간을 소모하기 때문에 여전히 실제 문제에 적용이 어려운 경우가 존재한다.

본 논문에서는 backtracking 과정에서 알고리즘 동작 시간을 길게 만들 수 있는 요인을 탐구하고, 이를 해결함으로써 실행 시간을 획기적으로 줄일 수 있는 새로운 기법을 소개한다. 또한, 기존 알고리즘 중 state-of-the-art 인 DAF[4] 에 기법을 적용한 경우와 그렇지 않은 경우를 구현하고 실제 그래프 데이터 상에서 실험을 진행하여 제시한 기법이 문제를 효과적으로 해결하는데 도움이 될 수 있는 방법임을 입증했다.

주요어: 부분 그래프 동형

학번: 2019-27354

Contents

List of Figures

Chapter 1

서론

대형 그래프에 대한 분석은 최근 소셜 네트워크 [3], 생물 정보학 [8], 화학 [10] 등 다양한 분야에서 점차 중요해지고 있다. 그래프 분석에서 가장 핵심적인 문제 중 하나로 부분 그래프 동형 문제 (subgraph isomorphism) 가 있다. 부분 그래프 동형 문제란 쿼리 그래프 q 와 데이터 그래프 G 가 주어졌을때, G 에 존재하는 q 의 embedding 의 존재여부를 확인하는 문제이다. 데이터 그래프 G에서 쿼리 그래프 q의 embedding은 q 의 각 정점 u 를 G 의 정점 v 로 mapping 하는 전사 함수로, 각 매핑 (u, v) 을 구성하는 두 정점의 레이블(label) 이 같아야 하며, q의 각 간선에 대해 이들의 양 끝 두 정점의 대응을 서로 연결하는 간선이 G에 존재해야한다. 여기서 레이블은 그래프의 각 정점을 1에서 Σ 사이의 정수로 매핑시키는 함수이다.

부분 동형 그래프 문제를 backtracking 활용하여 해결하는 많은 알고리즘이 (e.g., VF2[2], Turbo-iso[5], CFL-Match[1], DAF[4]) 제시되어왔다. 하지만 최악의 경우 지수적인 시간 복잡도를 가지는 backtracking 특성상 여전히 특정 입력에서는 답을 찾기 위해서 지나치게 긴 시간을 소모하기 때문에 여전히 실제 문제에 적용이 어려운 경우가 존재한다.

본 논문에서는 backtracking 과정에서 알고리즘 동작 시간을 길게 만들 수 있는 요인을 탐구하고, 이를 해결함으로써 실행 시간을 획기적으로 줄일 수 있는

새로운 기법을 소개한다. 또한, 기존 알고리즘 중 state-of-the-art 인 DAF[4] 에 기법을 적용한 경우와 그렇지 않은 경우를 구현하고 실제 그래프 데이터 상에서 실험을 진행하여 제시한 기법이 문제를 효과적으로 해결하는데 도움이 될 수 있는 방법임을 입증했다.

본 논문의 구성은 다음과 같다. 2장에서는 부분 그래프 동형 문제의 정의와 제시한 기법의 등장 배경 등 필요한 배경 지식을 설명한다. 3장에서는 DAF 알고리즘의 전체적인 구조에 대해서 설명한다. 4장에서는 새롭게 제시된 기법이 어떻게 동작하는지에 대해서 설명한다. 5장에서는 실험 결과를 제시하고, 6장에서는 결론을 작성한다.

Chapter 2

배경지식

2.1 그래프

본 논문에서 사용하는 그래프 g는 $g = (V(g), E(g), L_g)$ 과 같이 표현한다. $V(g)$ 는 정점의 집합이며 $E(g)$ 는 간선의 집합, L_g 는 레이블의 집합이다. 레이블은 그래프의 각 정점을 1에서 Σ 사이의 정수로 매핑시키는 함수이다. 본 논문에서는 방향성이 없는(undirected) 연결된(connected) 그래프에 대해서만 논의하기로 한다.

Definition 2.1 (Induced subgraph)**.** *정점 $V(q)$ 의 부분 집합 S에 대해서, induced subgraph는 S를 정점 집합으로 가지고, $E(q)$ 중에서 양 끝점이 모두 S에 포함하는 모든 간선들만 간선의 집합으로 포함하는 그래프를 의미한다. 이 그래프는 q[S] 으로 표현한다.*

2.2 부분 그래프 동형 문제(Subgraph isomorphism)

Definition 2.2 (Embedding)**.** *쿼리 그래프 $q = (V(q), E(q), L_q)$와 데이터 그래프 $G = (V(G), E(G), L_G)$ 가 주어졌을 때, embedding $M : V(q) \rightarrow V(G)$ 은 다음을 만족하는 함수이다.*

1. *M 은 단사 함수이다. (i.e., 모든 $u \neq u' \in V(q)$ 에 대해서 $M(u) \neq M(u')$ 를 만족한다).*

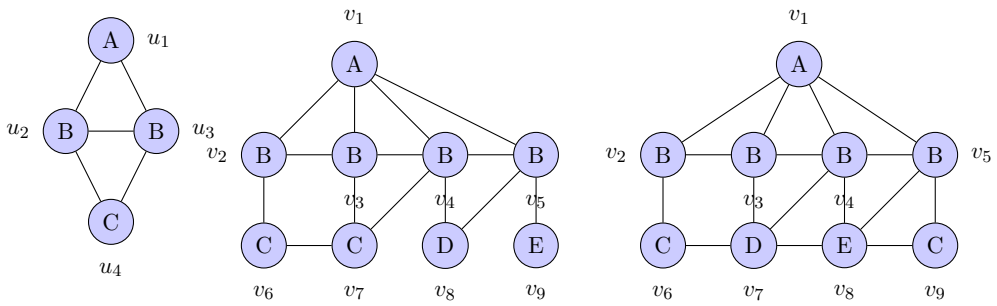
2. *모든 $u \in V(q)$에 대해서 $L_q(u) = L_G(M(u))$ 를 만족한다.*

3. *모든 $(u, u') \in E(q)$에 대해서 $(M(u), M(u')) \in E(G)$ 를 만족한다.*

Definition 2.3 (Partial embedding)**.** *쿼리 그래프 q의 $S \subset V(q)$ 에 대한 induced graph 의 embedding 을 partial embedding 이라고 한다. 특별히 $S = V(q)$ 이라면 full embedding 이라고 한다.*

Definition 2.4 (부분 그래프 동형 문제(Subgraph isomorphism))**.** *쿼리 그래프 q의 embedding 이 데이터 그래프 G에 존재한다면 q는 G에 subgraph ismorphic 하다고 한다. q가 G에 subgraph isomorphic 한지 판단하는 문제를 subgraph isomorphism 이라고 한다.*

Figure 2.1의 쿼리 그래프 q 는 데이터 그래프 G_1에 대해서 embedding $\{(u_1, v_1)$, (u_2, v_3), (u_3, v_4), (u_4, v_7) $\}$ 이 존재한다. 하지만, 데이터 그래프 G_2에 대해서는 embedding 이 존재하지 않다.



(a) **Query q** (b) **Data graph G_1** (c) **Data graph G_2**

Figure 2.1: **Subgraph isomorphism**

2.3 Directed acyclic graph

Definition 2.5 (DAG(directed acyclic graph))**.** *방향성이 있는 연결된 그래프 g에 cycle 이 존재하지 않는다면, 해당 그래프는 DAG라고 부른다. 주어진 DAG*

에 대해서, incoming 간선이 없는 정점을 root 이라고 부르며, outgoing 간선이 없으면서 incoming 간선이 하나뿐인 정점을 leaf 이라고 부른다.

2.4 이분 그래프(Bipartite graph)와 최대 이분 매칭(Maximum bipartite matching)

Definition 2.6 (이분 그래프(Bipartite graph))**.** *주어진 그래프 $g = (V(g), E(g), L_g)$ 에 대해서 다음과 같은 조건을 만족하는 $V_1, V_2 \subset V(g)$ 를 정의할 수 있다면, 이분 그래프(Bipartite graph) 라고 한다.*

1. $V_1 \cap V_2 = \emptyset$
2. $V_1 \cup V_2 = V(g)$ *이다.*
3. $e \in E(g)$ *의 끝점은 각각* V_1 *과* V_2*에 속한다.*

Definition 2.7 (최대 이분 매칭(Maximum bipartite matching))**.** *주어진 그래프 $g = (V(g), E(g), L_g)$ 에 대해서 공통되는 끝점을 가지지 않는 간선의 부분 집합을 매칭(matching) 이라고 한다. 매칭들 중에서 가장 집합의 크기가 큰 매칭을 최대 매칭(maximum matching) 이라고 한다. 만약 그래프가 이분 그래프였을 경우, 최대 이분 매칭(maximum bipartite matching) 이라고 한다.*

Figure 2.2a 에서 주어진 그래프에 대한 최대 이분 매칭은 $\{(u_2, v_1), (u_3, v_2), (u_5, v_3)\}$ 이다. 최대 이분 매칭을 구성하는 간선은 진한 회색으로 표시하였다.

최대 이분 매칭을 찾는 문제는 다항 시간에 해결이 가능하다. Figure 2.2a 과 같이 정점을 $V_1 = \{u_1, u_2, u_3, u_4, u_5\}$ 와 $V_2 = \{v_1, v_2, v_3, v_4\}$ 으로 분할이 가능한 이분 그래프가 입력으로 들어왔을 때, Figure 2.2b 와 같이 그래프를 생성한다. 규칙은 아래와 같다.

- 기존에 존재하지 않던 두 개의 정점 s, t 를 추가한다.
- s 으로부터 V_1 에 속하는 모든 정점을 향하도록 방향성 간선을 추가한다.
- V_2 에 속하는 모든 정점으로 부터 t로 향하도록 방향성 간선을 추가한다.

- 기존에 존재하던 간선은 V_1 에 속한 정점에서 V_2 에 속한 정점으로 방향을 부여해준다.

최대 이분 매칭과 새롭게 만든 그래프 에서의 최대 유량 값은 동일하다. 최대 유랑을 찾는것은 Ford-Fulkerson 알고리즘과 같은 다항 시간 알고리즘이 존재하기 때문에 최대 이분 매칭을 다항 시간에 구하는 것이 가능하다.

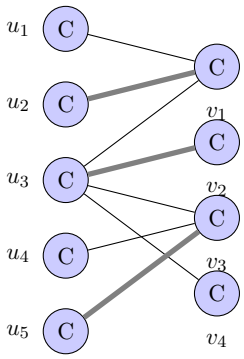
2.5 Subgraph isomorphism 알고리즘

최근 연구에서는 backtracking 기반하는 알고리즘, 특히 filtering 과정과 verification 과정으로 구성된 filtering-verification 구조를 사용하는 부분 그래프 동형 알고리즘 (e.g., VF2[2], Turbo-iso[5], CFL-Match[1], DAF[4]) 이 좋은 성능을 보여주고 있으므로, 해당 구조에 속하는 알고리즘에 초점을 맞춘다.

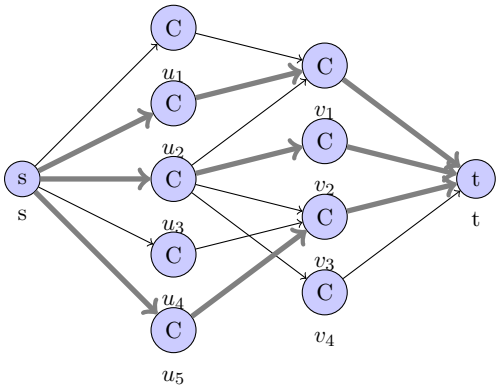
1. Filtering 과정은 쿼리 그래프의 각 정점에 대해서 매핑이 될 수 없는 데이터 그래프의 정점을 최대한 제거함으로써 실제로 embedding 을 찾는 과정에 들어가는 연산을 최소한으로 하기 위한 전처리 과정이다.

2. Verification 과정은 backtracking 을 수행하는 과정으로 filtering 과정을 거치고 남은 매핑이 될 수 있는 데이터 정점들의 후보들 (candidates) 을 쿼리 정점에 대해서 순차적으로 매핑을 하면서 모든 정점에 매핑이 된 full embedding 을 찾아가는 과정이다. Verification 은 partial embedding 을 full embedding 이 나올때까지 확장하는 과정이라고 할 수 있다.

Filtering 과정은 단순히 verification 을 하기 위한 자료구조를 생성하는 전처리 과정의 의미 이상으로 그래프의 크기에 대해서 다항시간의 시간 복잡도만을 사용하여 지수적으로 증가하는 verification 의 시간 복잡도를 최대한 낮추는 역할을 한다. 본 논문에서 개발한 기법은 해당 구조에 속하는 알고리즘 중에서 state-of-the-art 알고리즘인 DAF 에 적용하여 실험을 진행한다.



(a) **Bipartite graph g**



(b) **Find maximum bipartite matching by max flow algorithm**

Figure 2.2: **Maximum bipartite matching**

Chapter 3

DAF 알고리즘

본 챕터에서는 DAF 알고리즘의 전체적인 동작 구조와 중요한 기법에 대해서 설명하도록 하겠다.

3.1 알고리즘 개요

Algorithm 1 에서는 DAF의 대략적인 작동 구조를 설명하고 있다.

1. BuildDAG 에서는 입력으로 들어온 쿼리 그래프 q와 데이터 그래프 G가 주어였을 때, 쿼리 그래프 q에 방향성을 부여해서 DAG 인 q_D 를 만들어준다.

2. BuildCS 에서는 DAG q_D 와 데이터 그래프 G에 대해서, CS자료구조를 생성해준다. filtering-verification 단계에서 filtering 단계에 해당한다.

3. Backtrack 의 경우에는 이전에 생성된 CS 자료구조를 활용해서 쿼리 그래프 q의 embedding이 데이터 그래프 G에 존재하는지 확인한다. Filtering-verification 단계에서 verification 에 해당한다.

Algorithm 1: DAF

Input: a query graph q, a data graph G

Output: true/false(whether there is any isomorphic subgraph)

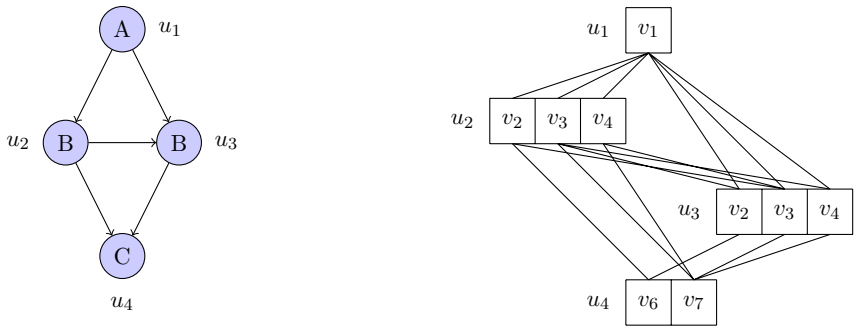
```
1 q_D ← BuildDAG(q, G);
2 C ← BuildCS(q, q_D, G);
3 M ← ∅;
4 if Backtrack(q, q_D, C, M) then
5 |   return true
6 else
7 |   return false
```

3.2 CS 구조와 DAG-Graph DP

주어진 쿼리 그래프 q와 데이터 그래프 G에 대해서, CS (candidate space) 구조는 모든 $u \in V(q)$ 에 대해서 정의된 $C(u) \subset V(G)$ 으로 구성되어있다. C(u) 는 u 에 mapping 이 될 수 있는 데이터 정점들의 집합을 의미힌다. Backtracking 과정은 C(u) 에서 정점을 순차적으로 선택해서 mapping 을 반복하는 과정으로 생각할 수 있다.

Filtering 을 사용하기 위해서 쿼리 그래프 q으로부터 DAG q_D 을 만든다. Root 가 될 정점 r을 선택하고 r 에서부터 BFS (breath first search) 를 실행한다. 방문을 한 이후, 모든 간선에 대해서 먼저 탐색된 정점으로 부터 나중에 탐색이 된 정점쪽으로 방향성을 부여함으로써 directed graph 를 만들어준다. 결과물은 항상 DAG 가 된다. 모든 $(u, u') \in E(q)$ 에 대해서 $v \in C(u)$ 정점과 $v' \in C(u')$ 정점이 데이터 그래프에서도 연결되어 있었다면, CS 자료구조상에서도 연결해준다. 이후에 DAG 상에서 정의되는 DAG-DP 를 사용하여 C(u) 의 크기를 작게 만든다.

Figure 3.1a 의 경우에는 BFS 방문 순서가 u_1, u_2, u_3, u_4 일 경우 생성되는 DAG 이다. Figure 3.1b 의 경우에는 DAG-DP 를 활용했을때 최종 CS 자료구조의 결과물이다. 해당 논문의 filtering 은 DAF 와 동일하기 때문에 상세한 동작 구조의 설명은 생략하겠다.



(a) q_D (b) **C**

Figure 3.1: q_D **and CS structure**

3.3 Backtracking

Definition 3.1 ($N^u_{u_c}(v)$)**.** *주어진 쿼리 그래프의 DAG q_D와 데이터 그래프 G 으로부터 생성된 CS 자료구조 C가 있다고 가정하자. $u, u_c \in V(q_D)$ 에 대해서 u_c 가 u 가 자식 정점일 경우, $N^u_{u_c}(v)$ 는 다음과 같은 조건을 만족하는 $v_c \in V(G)$ 의 집합이다.*

1. *$v_c \in C(u_c)$ 이다.*
2. *v_c 와 v 는 G에서 서로 인접한다.*

Definition 3.2 (Extendable vertex)**.** *만약 partial embedding M 이 주어진 상황에서 mapping 이 된 쿼리 정점의 집합인 $V(q_D)$의 부분 집합을 S라고 표현하자. $u \notin S$ 에 대해서, 만약 q_D 에서의 모든 부모가 S상에 포함되어있다면 u를 extendable vertex 이라고 정의한다.*

Definition 3.3 (Extendable candidates)**.** *쿼리 그래프 q와 데이터 그래프 G, 그리고 그로 부터 생성된 DAG인 q_D 와 CS 구조 C가 주어졌다고 가정하자. Partial embedding M 에 대해서 mapping 이 되지 않은 정점 u의 mapping 이 된 q_D 상에서의 부모들을 $p_1, p_2, ..., p_k$ 이라고 표현하자. Extendable candidates $C_M(u)$ 는 $C_M(u) = C(u) \cap (\bigcap_{i=1}^{k} N^{p_i}_u(M(p_i)))$ 으로 정의된다. 만약 mapping 이 된 부모가 없다면, $C_M(u) = C(u)$ 를 만족한다.*

Algorithm 2: BACKTRACK(q, q_D, C, M)

```
1  if |M| = |V(q_D)| then
2      return true;
3  else
4      Select a next extendable vertex u;
5      foreach v ∈ C_M(u) do
6          if v is unvisited then
7              M' ← M ∪ {(u, v)};
8              Mark v as visited;
9              if Backtrack(q, q_D, C, M') == true then
10                 return true;
11             Mark v as unvisited;
12     return false;
```

Backtracking 과정은 $M = \emptyset$ 의 partial embedding 으로부터 시작해서 extendable vertex 를 선택하고 extendable candidates 에서 mapping을 시도할 정점을 선택해서 embedding의 조건을 만족할 경우 mapping을 하는 과정을 반복함으로써 full embedding으로 확장해가는 과정이라고 할 수 있다. Algorithm 2에 상세한 과정이 적혀있다.

주어진 partial embedding에 대해서 다양한 extendable vertex 가 존재할 수 있고, 그 중 어떤 것을 먼저 mapping 을 시도할 것인지를 선택하는 기준도 여러가지가 존재한다. 해당 논문에서는 가장 작은 extendable candidates size 를 가지는 extendable vertex 를 선택하는 방법인 candidate size order 을 선택한다.

3.4 Leaf decomposition

현재 subgraph isomorphism 알고리즘들 중에서 가장 좋은 성능을 보여주는 논문인 CFL-Match, Turboiso, DAF [1, 4, 5]는 leaf decomposition strategy를

사용했다. 해당 기법은 쿼리 그래프를 leaf 정점의 집합과 아닌 집합 V'로 구분한 이후 q[V'] 에 대해서 partial embedding 을 먼저 찾고 나서야 나머지 $V(q) \setminus V'$ 에 대해서 mapping을 시도하는 기법이다. Leaf 정점에 매핑시키는 과정은 search space가 permutation order 로 증가시키는 원인이 될 수 있다. 따라서, non leaf 정점을 먼저 mapping 을 해줌으로써 전체적인 search space 의 공간을 줄이는 목적으로 제시되었고, 기존 방법을 사용한 알고리즘들에서 좋은 성능을 보이는 것이 실험을 통해서 입증되었다.

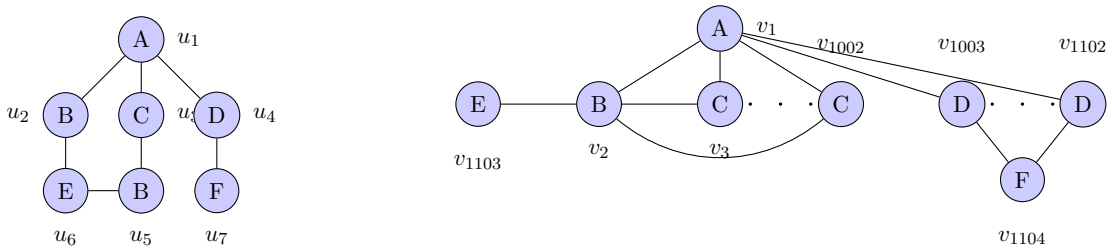
3.5 Pruning by failing sets

Figure 3.2는 새로운 입력으로 들어온 쿼리 그래프 q와 데이터 그래프 G이다. Figure 3.3는 backtracking 과정을 search tree 형태로 (각각의 node는 partial embedding 을 의미한다. 트리의 깊이가 깊어질 수록 partial embedding 이 확장되었다는 것을 의미한다) 표현한 것이다. M 을 root 로 가지는 subtree 에서 full embedding 을 찾지 못하고 M으로 다시 돌아온 상황을 가정하자. M이 root인 subtree 대응되는 search space 에 full embedding 이 없는 원인은 u_4 의 mapping 과는 무관한 것을 알 수 있다. 따라서, Figure 3.3 상에서 회색으로 표현된 M의 sibiling 들이 나타내는 partial embedding을 확장하는 시도도 역시 full embedding 을 찾지 못한다. DAF에서는 이러한 유형의 search tree node 들을 pruning 함으로써 불필요한 계산을 줄였다.

Definition 3.4 (Ancestor)**.** *DAG인 그래프 g_D 가 주어진다고 가정하자. 정점 u 에서 부터 v으로의 path가 존재할 경우 u는 v의 ancestor이라고 한다. 정점 $u \in V(g_D)$ 의 ancestor 들의 집합을 anc(u) 으로 표현한다.*

Definition 3.5 (Ancestor closed)**.** *DAG인 그래프 g_D에 대해서 $S \subset V(g_D)$가 주어졌다고 가정하자. 임의의 $u \in S$ 에 대해서 u의 ancestor가 모두 S의 원소라면, S는 ancestor closed 하다고 한다.*

편의상 $S \subset V(q)$ 과 partial embedding M 이 주어졌을 때, partial embedding M[S] 는 $\mathrm{M[S]} = \{(u, v) | (u, v) \in M \wedge u \in S\}$ 으로 정의한다.



(a) **q** (b) **G**

Figure 3.2: **Query grpah q and data graph G**

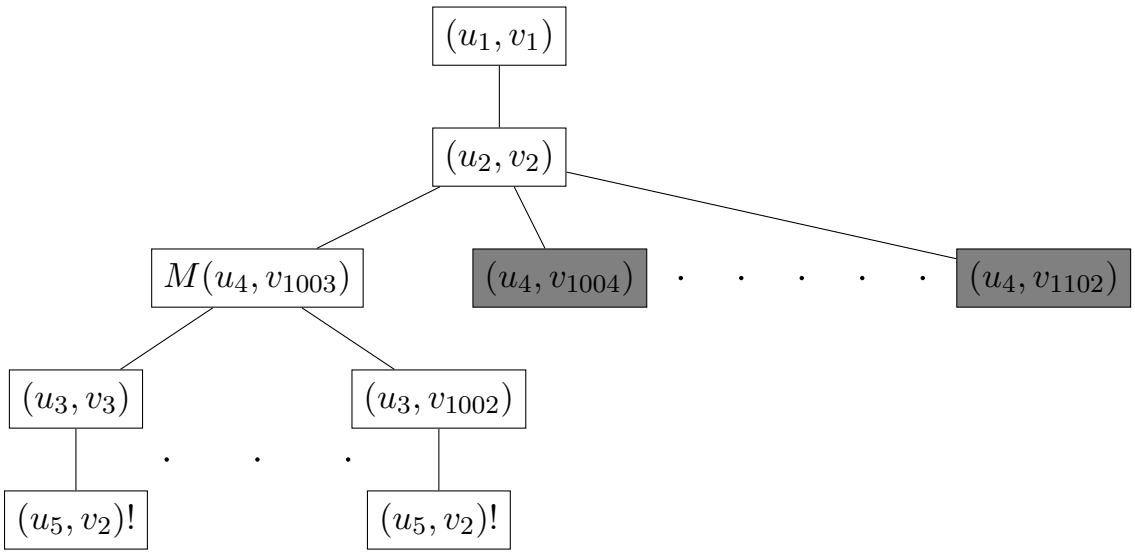


Figure 3.3: **Search tree**

Definition 3.6 (Failing set)**.** *Partial embedding M에 대해서, 정의된 $F_M \subset V(q)$ 이 다음과 같은 failure property 을 만족하면서 ancestor closed 인 경우 failing set 이라고 한다.*

- *$M[F_M]$ 의 extension 인 full embedding 은 존재하지 않는다.*

Figure 3.3 에서의 M에대한 failing set의 예시는 $F_M = \{u_1, u_2, u_3, u_5\}$ 이고, 이 때, $M[F_M] = \{(u_1, v_1), (u_2, v_2)\}$ 을 만족한다. 즉, u_4 의 mapping 을 변경해도 embedding 이 없음을 알 수 있다. 이를 lemma 으로 일반화 시키면 다음과 같다

Lemma 3.1. *Search tree 상의 정점 M 이 (u,v) mapping 을 통해서 생성이 되었*

다고 가정하자. M에 대해서 정의된 failing set F_M 에 대해서, $u \notin F_M$ 의 조건이 만족한다면 M의 sibiling 인 정점들은 full embedding 으로 확장될 수 없다.

그렇기 때문에 만약 임의의 partial embedding M 에 대해서 F_M 을 계산하는 것이 가능하다면 앞선 lemma 의 조건이 만족되는 경우 mapping 을 해야하는 M 의 sibling 을 탐색하는 것은 불필요하다는 것을 미리 알 수 있다. 따라서 해당 탐색 정점에 가지치기 함으로써 연산 시간을 절약하는 것이 가능하다.

failing set 을 계산하는 방법의 경우에는 DAF 에서는 bottom up fashion 기반의 방법을 제시했다. 우선 Search tree node 를 크게 leaf node 와 internal node 으로 구성했다. 만약 leaf node 에 대해서 failing set 이 잘 정의되어 있다면, 다음과 같이 internal node 에 대해서 failing set 을 계산했을 때, 앞서 언급한 failing property 와 ancestor closed 성질이 만족됨을 DAF 논문이 증명했다.

Lemma 3.2. *(u,v) 를 매핑함으로써 생성된 internal node M 이 주어져있다고 가정하고, 다음 정점으로 u_n 을 선택한다고 가정하자. 아래와 같이 internal node 에 대해서 F_M 을 계산한다면 F_M 은 failure property 와 ancestor closed 성질을 만족한다.*

- *만약 적어도 하나의 child node M_i 에 대해서 $F_{M_i} = \emptyset$ 을 만족한다면 $F_M = \emptyset$ 으로 정의한다.*
- *모든 child node M_i 에 대해서 $F_{M_i} \neq \emptyset$ 인 경우에는 다음과 같이 계산한다.*
  - *$u_n \notin F_{M_i}$ 를 만족하는 F_{M_i} 가 존재한다면 $F_M = F_{M_i}$ 으로 정의한다.*
  - *그러한 child 가 없다면, $F_M = \cup_{i=1}^{k} F_M$ 으로 정의한다.*

위 lemma 가 성립하기 때문에 leaf search tree node 에 대해서만 failing set 이 정의된다면, 모든 정점에 대해서 failing set 을 정의하는 것이 가능하다. Leaf 정점은 크게 extendable candidates 이 $\emptyset$ 이라서 발생하는 emptyset-class, 선택한 candidate 정점이 기존에 mapping 된 visited 정점이라 발생하는 conflict-class, 그리고 embedding 을 찾은 embedding-class 가 있다. 각각의 경우에 대해서 failing set 을 아래와 같이 정의할 수 있다.

Lemma 3.3. *Search tree 에서 발생하는 leaf search tree node M 이 (u,v)가 추가되어서 만들어진 것이라고 가정하자. 상황에 따라서 F_M 아래와 같이 정의할 경우 failing property 와 ancestor closed 성질을 만족한다.*

1. *v 가 기존에 mapped candidate 이라면 conflict class 이라고 정의하며, search tree 상에서는 (u,v)! 으로 표현한다. $M = \{..., (u', v), ..., (u, v)\}$ 형태의 partial embedding 이라면, F_M = anc(u) ∪ anc(u') 으로 정의한다.*

2. *u 의 extendable candidates 의 size 가 0인 경우도 있다. 이런 경우에는 emptyset-class 이라고 정의하고, $F_M = anc(u)$ 으로 계산한다.*

3. *M 이 full embedding 에 해당한다면, embedding-class 이라고 정의한다. 이 경우에는 $F_M = \emptyset$ 가 된다.*

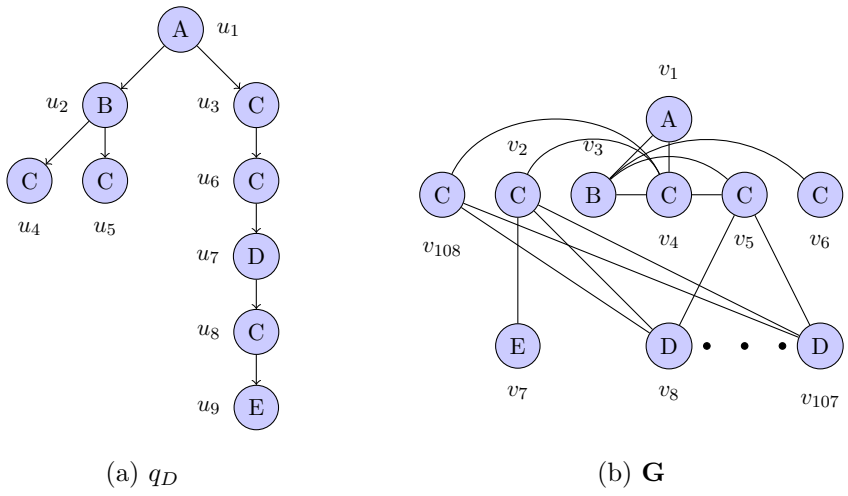
앞서 제시한 Lemma 3.2, Lemma 3.3 에 의해서 모든 search tree node 에 대해서 failing set 을 정의하는 것이 가능하다.

Chapter 4

최대 이분 매칭에 활용한 가지치기

본 장에서는 새롭게 제시한 효율적인 가지치기 기법에 대해서 소개한다.

4.1 최대 이분 매칭을 활용한 가지치기 기법 과정



(a) q_D (b) **G**

Figure 4.1: **Query graph q(q_D) and data graph G**

Figure 4.1에서 제시된 쿼리 그래프 q와 데이터 그래프 G에 대해서 현재 $M_1 = \{(u_1, v_1), (u_2, v_3), (u_3, v_4), (u_6, v_5)\}$ 을 찾은 상황이라고 가정한다. DAF 는 leaf

decomposition strategy 를 따르기 때문에 non leaf 인 u_7 과 u_8 을 추후에 순서대로 mapping을 완료한 이후에야 $\{u_4, u_5, u_9\}$ 의 정점 집합들을 매핑하는 것을 시도할 것이다. 하지만, M_1 partial embedding 은 $\{u_4, u_5\}$ leaf 에 mapping 을 할 수 있는 candidates 은 v_6 뿐이기 때문에 injective 성질을 만족하는 mapping 을 찾을 수 없다. 이렇게 leaf 에 대한 injective 를 만족할 수 있는 mapping 이 없다는 사실을 미리 발견할 수 있다면 Figure 4.2a 과 같이 많은 불필요한 search tree 를 탐색하는 비용이 발생하지 않는다. 즉, partial embedding 이 주어진 상황에서, leaf node 정점과 extendable candidates 간의 일대일 대응이 존재하는지 파악하고 가지치기를 한다면 소모 시간에서의 많은 개선이 있을 수 있다.

Non leaf 정점 또한 비슷한 문제가 발생할 수 있다. $M_2 = \{ (u_1, v_1), (u_2, v_3), (u_3, v_4), (u_6, v_2) \}$ 의 partial embedding 을 찾은 상황을 가정하자. 이 경우에는 이전에 문제가 되었던 $\{u_4, u_5\}$ 에 대해서는 mapping 을 만들 수 있는 2개의 candidates 가 $\{v_5, v_6\}$ 으로 존재하기 때문에 injective 해야 한다는 성질을 위배하지 않는다. 그러나, non leaf 쿼리 정점인 u_8 의 경우에는 extendable candidates 은 $\{v_2\}$ 뿐이지만, 이 정점은 이미 mapping 이 되었기 때문에 injective 한 mapping 을 찾을 수 없다. 그렇기에 Figure 4.3a 와 같이 불필요한 search tree 의 정점 방문이 일어난다.

위 두 상황은 partial embedding 이 결정된 상황에서, mapping 이 되지 않은 정점들이 extendable candidates 이 주어졌을때 일대일 대응을 찾을 수 없어서 낭비되는 연산들이다. 이를 해결하기 위해서 아래와 같은 자료구조를 만든다.

Definition 4.1 (B_M)**.** *쿼리 그래프 q 와 데이터 그래프 G에 대해서, 그로부터 만들어진 CS 구조 C가 주어졌다고 가정하자. Partial embedding 이 M 인 상황에서 편의상 mapping 이 된 쿼리 그래프 정점의 집합과 mapping 이 된 데이터 그래프 정점의 집합을 각각 $V(q)_M$ 와 $V(G)_M$ 으로 표현한다. 이분 그래프 B_M 은 다음과 같이 정의된다.*

- $V(B_M) = (V(q) \setminus V(q)_M) \cup (V(G) \setminus V(G)_M)$
- $u \in (V(q) \setminus V(q)_M)$ *와* $v \in (V(G) \setminus V(G)_M)$ *이* $v \in C_M(u)$ *관계식을 만족한다면 u 와 v 사이에 간선을 연결한다.*

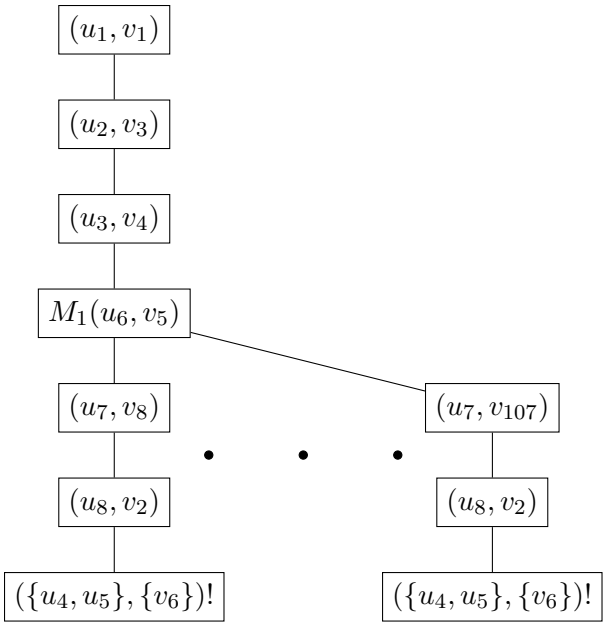
Lemma 4.1. *만약 B_M 에서의 maximum bipartite matching 의 크기가 $|V(q) \setminus V(q)_M|$ 보다 작다면, M은 full embedding 으로 확장될 수 없다.*

이는 직관적인 사실인데, 만약 주어진 lemma 의 조건이 만족된다면, 이후에 어떠한 mapping 을 시도하더라도 injective 하도록 모든 정점에 mapping 을 할 수 없기 때문이다. 그렇기 때문에, lemma 의 조건을 만족할 때 더 backtracking 을 진행하지 않고 pruning 을 한다면, 해당 partial embedding 이 root 인 subtree 를 탐색해야 했던 비용을 절약할 수 있다. 앞서서 문제가 되었던 partial embedding M_1 과 M_2 에 상황을 다시 살펴보자.

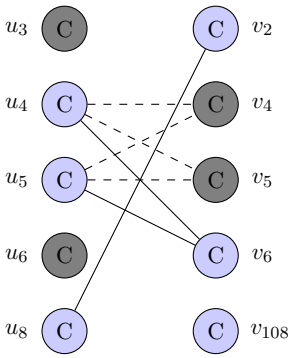
전체 B_M 을 표현하기 보다는 라벨이 C인 정점들로 구성된 induced graph 에 대해서만 표현했는데, 이는 induced graph 에 대해서만 위 lemma 의 조건을 만족해도 전체 그래프에서도 위 lemma 의 조건을 만족하기 때문이다. B_M 은 mapping 이 되지 않은 정점들만 대상으로 하지만, M_1 와 M_2 의 예시의 형태를 통일시키기 위해서 이미 mapping 이 된 정점도 그렸다. 대신 회색으로 정점을 표현했다. 절취선과 실선은 query 정점과 데이터 정점간의 extendable 의 관계를 나타낸다. Extendable candidates 이 mapping 이 되지 않았다면 실선으로 실제 B_M 에 포함된다. 하지만 Extendable candidates 이 mapping 이 된어서 B_M 에 포함이 되지 않는다면 절취선으로 표현하고 B_M 에는 포함되지 않는다.

Figure 4.2b 은 B_{M_1} 그래프에서부터 라벨이 C인 정점들로 구성된 induced graph 이다. B_{M_1} 의 최대 이분 매칭의 크기는 2이다. 이 사실을 통해서 $\{u_4, u_5, u_8\}$ 에 대해서 전부 injective 한 mapping 을 찾을 수 없다는 사실을 미리 알아서 pruning 이 가능하다. 이는 앞서 언급한 leaf 정점 $\{u_4, u_5\}$ 에 대한 injective mapping 이 존재하지 않는 사실을 간접적으로 확인한 것이다.

Figure 4.3b 은 B_{M_2} 그래프에서부터 라벨이 C인 정점들로 구성된 induced graph 이다. 해당 bipartite graph 에서 찾은 최대 이분 매칭은 0이며, 이 사실을 통해서 $\{u_4, u_5, u_8\}$ 에 대해서 전부 injective 한 mapping 을 찾을 수 없다는 사실을 미리 알아서 pruning 이 가능하다. 이는 앞서 언급한 non leaf 정점 u_8 에 injective 한 성질을 만족하도록 mapping 을 하는 것이 불가능하다는 것을 간접적으로 확인한 것이다.



(a) **Search tree for** M_1



(b) **Bipartite for** M_1

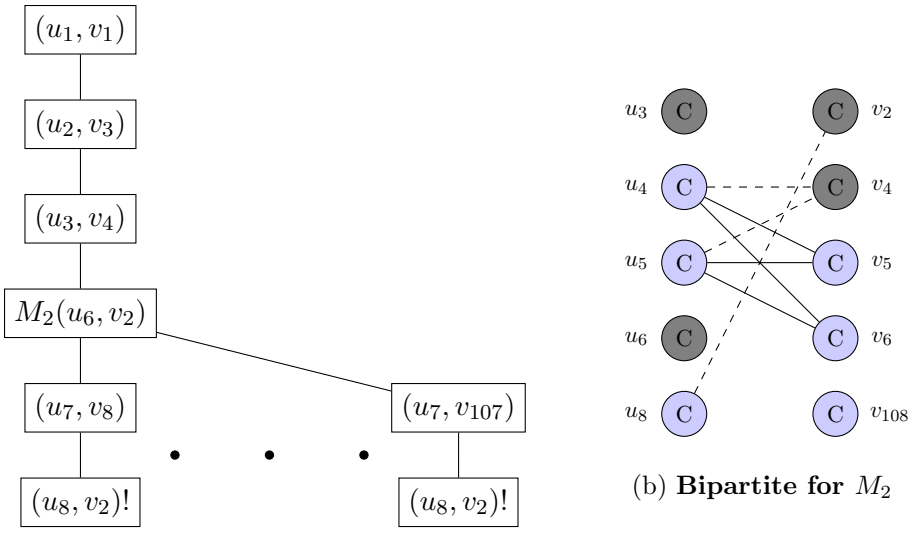
Figure 4.2: M_1

4.2 Failing set for pruning by maximum bipartite matching

제시한 maximum bipartite matching 을 이용한 가지치기 기법은 DAF 가 제시했던 failing set 을 이용한 가자치기 기법을 바로 적용하기 힘들게 한다. 이분 최대 매칭에 의해서 pruning 이 되는 search tree 의 정점은 leaf 정점이 되는데 이는 앞서서 정의한 3가지 경우 중 (conflict-class, emptyset-class, 그리고 embedding-class) 어느것에도 속하지 않기 때문에 failing set 이 정의가 되지 않는다. 따라서 maximum bipartite matching 에 의해서 pruning 되는 leaf 정점에 대해서 failing set 을 새롭게 정의할 필요해야 failing set 을 활용한 가지치기 기법을 적용할 수 있다. 본 논문에서는 해당 leaf 정점을 "pruning-class" 라고 새롭게 명명한다.

Partial embedding M 이 "pruning-class" 에 속할 때, 전체 그래프 B_M 에 대해서 다음을 정의한다.

Definition 4.2 ($\pi_M(u)$)**.** $u \in V(B_M)$ *에 대해서,* u *를 포함하는 connected com-*



(a) **Search tree for** M_2

(b) **Bipartite for** M_2

Figure 4.3: M_2

ponent 를 $\pi_M(u)$ *이라고 표현한다.* $\pi_M(u)$ *의 maximum bipartite matching 의 크기가* $|V(\pi_M(u))|$ *보다 작다면 unmatchable 하다고 정의한다.*

이후 소개할 증명의 편의를 위해서 unmatchable 인 connected component 의 구성 성분과 연관된 집합을 다음과 같이 정의한다.

Definition 4.3. *Partial embedding M 인 상황에서, 아직 mapping 이 이루어지지 않은 정점* $u \in V(q)$ *에 대해서 다음을 정의한다.*

- $UC_M(u) : V(\pi_M(u)) \cap (V(q) \setminus V(q)_M)$.
- $UE_M(u) : \bigcup_{u' \in UC_M(u)} C_M(u')$.
- $UV_M(u) : V(G)_M \cap UE_M(u)$.
- $UF_M(u) : = UE_M(u) \setminus UV_M(u)$.
- $UB_M(u) : \{M^{-1}(v) | v \in UV_M(u)\}$.

각각의 정의는 다음과 같은 의미를 가진다.

- $UC_M(u)$: $\pi_M(u)$ 을 구성하는 mapping 이 되지 않은 쿼리 그래프의 정점들의 집합.
- $UE_M(u)$: $UC_M(u)$ 의 쿼리 정점들의 extendable candidates 의 합집합.
- $UV_M(u)$: $UE_M(u)$ 에서 mapping 이미 이루어진 데이터 그래프의 정점들의 집합.
- $UF_M(u)$: $UE_M(u)$ 에서 mapping 이루어지지 않은 데이터 그래프의 정점들의 집합.
- $UB_M(u)$: $UV_M(u)$ 의 정점들에 매핑을 시킨 쿼리 정점의 집합.

M_1 의 예시를 참조하면 현재 mapping된 데이터 그래프 정점이 $V(D)_{M_1} = \{v_1, v_3, v_4, v_5\}$ 인 상황에서 unmatchable connected component 는 $\pi_{M_1}(u_4)$ 뿐이다. $\pi_{M_1}(u_4)$ 의 정점중에서 mapping 이 되지 않은 쿼리 정점의 집합은 $UC_{M_1}(u_4) = \{u_4, u_5\}$ 을 만족하고 extendable candidates 의 합집합은 $UE_{M_1}(u_4) = C_{M_1}(u_4) \cup C_{M_1}(u_5) = \{v_4, v_5, v_6\}$ 이다. 이 extendable candidates 들의 집합인 $UE_{M_1}(u)$ 은 mapping 이 된 정점의 집합인 $UV_{M_1}(u) = V(D)_{M_1} \cap UE_{M_1}(u) = \{v_4, v_5\}$ 와 mapping 이 되지 않은 정점의 집합인 $UF_{M_1}(u) = UF_{M_1}(u) \setminus UV_{M_1}(u) = \{v_6\}$ 으로 나눌 수 있다. Mapping 이 완료된 정점들이 매핑이 된 쿼리 정점들을 다 모으면 $UB_{M_1}(u_4) = \{M_1^{-1}(v_4),\ M_1^{-1}(v_5)\} = \{v_3, v_6\}$ 가 된다.

위에서 정의한 성분들을 바탕으로 pruning-class 에 속한 임의의 partial embedding M 의 failing set F_M 은 아래와 같이 정의한다.

Definition 4.4. *Maximum bipartite matching 에 의해서 pruning 이 된 search tree node M 의 B_M 은 적어도 하나의 unmatchable connected component 를 가지고 있고 그 중 하나인 $\pi_M(u)$ 를 선택하자 (여러 unmatchable connected component 가 존재할 수 있지만, M 에 대해서 특정한 하나를 선택할 수 있는 기준이 존재한다고 가정한다.). $\pi_M(u)$ 에 대해서 정의된 F_M 은 아래와 같이 정의된다.*

- $F_M = \bigcup_{u' \in UC_M(u) \cup UB_M(u)} anc(u')$

F_M 이 failing set 이 되기 위해서는 ancestor closed 이어야 하며, failing property 를 만족해야한다. Ancetor closed 의 경우에는 자명하게 성립한다. Failing property 를 만족한다는 것은 아래 lemma 에서 증명한다. 아이디어는 pruning class 에 속한 partial embedding M 이 만약 unmatchable connected component $\pi_M(u)$ 를 가지고 있다고 할 때, 모든 $UC_M(u)$ 에 대해서 injective 한 mapping 집합을 $M[F_M]$ 의 extension 에서는 생성할 수 없다는 사실을 보이는 것이다.

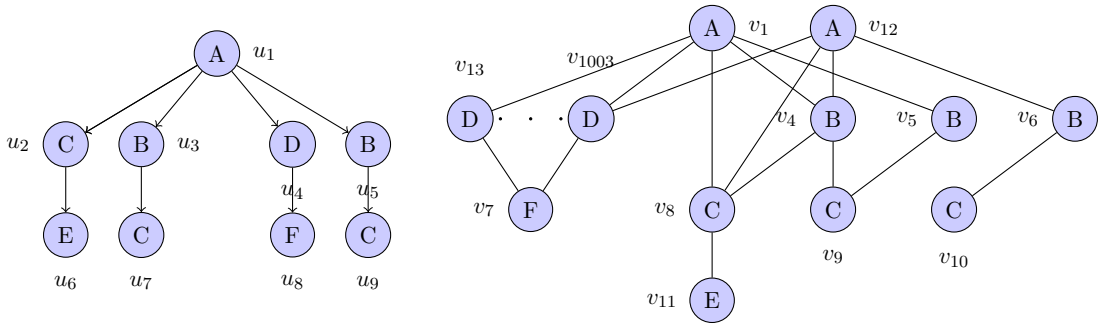
Lemma 4.2. *Pruning-class 에 속하는 partial embedding M 에 대해서 정의된 F_M 은 failing property 를 만족한다. 즉, partial embedding $M[F_M]$ 의 extension 인 full embedding 은 존재하지 않는다.*

Proof. M 은 적어도 하나의 unmatchable connected component 를 가지고 있고 그 중에서 $\pi_M(u)$ 를 선택하자. $u' \in UC_M(u)$ 의 M 상에서의 모든 mapping 이 된 parent 들의 집합을 편의상 $p_M(u')$ 으로 표현하자. Partial mapping 과 상관없이 DAG q_D 상에서의 parents 의 집합을 $p(u')$ 으로 표현하자.

F_M 은 정의에 의거하여 anc(u') 을 포함한다. Ancestor 의 정의에 의하여 parent 또한 ancestor 이다. 따라서, $p(u') \subset anc(u')$ 를 만족한다. 종합적으로 $p(u') \subset F_M$ 을 만족한다. $M[F_M]$ 은 F_M 에서부터 partial embedding M 에서 mapping 인 된 점들을 고른 것이기 때문에 $p_{M[F_M]}(u')$ 은 $p(u')$ 중에서 mapping 이 된 점의 집합이 된다. 따라서 $p_M(u') = p_{M[F_M]}(u')$ 을 만족한다. 또한 각각 parent $p' \in p_M(u')$ 에 mapping 이 된 candidate 은 $M(p')$ 으로 동일하다.

Extendable candidates 의 정의상 mapping 된 parent 의 집합이 동일하고, 각각의 parent 에 mapping 이 된 candidate 이 동일하다면 같은 extendable candidates 를 가진다. 따라서, $C_M(u') = C_{M[F_M]}(u')$ 가 성립된다.

Extendable candidates 은 partial embedding M 에서 mapping 이 된 candidate 의 집합인 $C_M(u') \cap UV_M(u)$ 과 아직 mapping 이 되지 않은 정점인 $C_M(u') \cap UF_M(u)$ 으로 분리된다. $C_M(u') = C_{M[F_M]}(u')$ 이기 때문에 $C_{M[F_M]}(u')$ 도 동일한 두 개의 집합으로 분리된다. 만약 $v \in C_M(u') \cap UV_M(u)$ 이라면 $v \in UV_M(u)$ 이기 때문에, $M^{-1}(v) \in UB_M(u)$ 이다. F_M 의 정의에 의해서 $UB_M \subset F_M$ 이다. 두 사실을 종합하면 $M^{-1}(v) \in F_M$ 이다. $M^{-1}(v) \in F_M$ 이면서 $(M^{-1}(v), v) \in M$ 이기



(a) $\mathbf{q}_D$

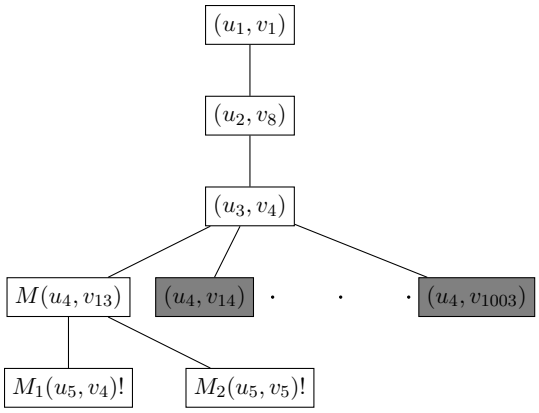
(b) **G**

Figure 4.4: **Query graph q and data graph G**

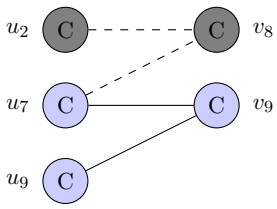
때문에 $M[F_M]$ 에 정의에 $(M^{-1}(v), v) \in M[F_M]$ 을 만족한다. 따라서 $C_{M[F_M]}(u')$ 에서 또한 v 는 mapping 이 된 candidate 이다. 만약 $v \in C_M(u') \cap UF_M(u)$ 이라면 v 는 M 에서 mapping 이 되지 않았기 때문에 partial embedding $M[F_M]$ 에서도 mapping 이 된 정점이 아니어야 한다. 따라서 v 는 $C_{M[F_M]}(u')$ 에서 역시 mapping 이 되지 않는 candidate 이다. 종합적으로, $C_M(u')$ 와 $C_{M[F_M]}(u')$ 에 포함되는 모든 candidate 의 mapping 여부는 동일하다. 따라서, partial embedding M 과 $M[F_M]$ 양쪽 모두에서 모든 $u' \in UC_M(u)$ 실제로 mapping 이 가능한 candidates 의 집합은 동일하다.

$M[F_M]$ 에서 full embedding 으로 확장하는 것이 가능하다고 가정하자. 이는 $B_{M[F_M]}$ 에서 maximum bipartite matching 의 크기가 $|V(B_{M[F_M]})|$ 이 값을 가지는 적절한 matching 을 찾을 수 있다는 것을 의미한다. Matching set 에서 $UC_M(u)$ 에 적용된 subset 은 $\pi_M(u)$ 에서 maximum bipartite matching 의 크기가 $|V(UC_M(u))|$ 이 되기 때문에 unmatchable 이라는 것에 모순이다. 따라서 $M[F_M]$ 이 full embedding 으로 확장되는 것은 불가능하고 failing property 를 만족한다. □

결과적으로 F_M 는 failing property 와 ancestor closed 성질 모두 성립하기 때문에 failing set 이다. 즉, 등장할 수 있는 모든 leaf search tree node 에 대해서 failing set 을 정의했기 때문에 Lemma 3.2 에 기반하여 모든 search tree node 의 failing set 을 정의할 수 있다.



(a) **Search tree**



(b) **Connected component**

Figure 4.5: **Search tree and connected component**

Figure 4.4 의 예시는 새로운 쿼리 그래프 q 와 데이터 그래프 G 를 제공한다. Figure 4.5a 에서 M_1 과 M_2 는 각각 conflict-class 와 pruning-class 이다. M_1 의 경우에는 $F_{M_1} = anc(u_5) \cup anc(u_3) = \{u_1, u_3, u_5\}$ 으로 계산된다. M_2 의 경우에는 Figure 4.5 와 같이 unmatchable component $\pi_{M_2}(u_7) = \{u_2, u_7, v_9\}$ 으로부터 $F_{M_2} = \{u_1, u_3, u_5, u_7, u_9\}$ 를 계산할 수 있다. $F_M = F_{M_1} \cup F_{M_2} = \{u_1, u_3, u_5, u_7, u_9\}$ 에서 $u_4 \notin F_M$ 조건을 만족하기 때문에 redundant 인 sibling search tree nodes 들을 pruning 할 수 있다.

4.3 기법의 정상 동작 여부를 위한 실험 설계

제시한 기법들은 가지치기 기법이기 때문에 extendable vertex 의 선택 순서나 extendable candidates 의 순회 순서에는 영향을 주지 않는다. 따라서, 기법을 적용하는 여부와 관계없이 backtracking 을 호출하면서 가장 먼저 찾는 embedding 은 가지치기를 적용하는 경우와 적용하지 않는 경우 모두 동일해야한다.

앞서 제시한 두 기법을 적용한 구현이 정상적으로 동작하는지 대한 간접적인 검증을 위 사실에 기반한 실험을 통해서 진행할 수 있다. 만약 제시한 기법이 정당하다면 모든 쿼리 그래프 데이터 그래프 쌍에수대해서 subgraph isomorphism 문제를 해결했을 때 기법을 적용한 경우와 아닌 경우 찾은 모든 embedding 의

집합이 동일해야한다.

본 논문에서는 좀 더 강한 검증을 위해서 subgraph isomorphism 과 밀접한 관련이 subgraph matching 문제에 대해서 실험을 진행했다. Subgraph matching 의 subgraph isomorphism 과 다른 점은 데이터 그래프에 존재하는 모든 embedding 을 전부 찾는다는 점이다. Subgraph matching 는 subgraph isomorphism 의 backtracking 과정에서 embedding 을 찾더라도 계속해서 탐색을 진행하도록 변경하는 작은 수정으로 구현이 가능하다. Subgraph matchihng 이 찾은 embedding 의 집합과 발견된 순서는 가지치기의 기법의 적용 유무와 무관하게 동일해야한다. Subgraph isomorphism 은 embedding 을 하나만 찾고 backtracking 을 종료하기 때문에 검증의 효력이 약하지만 그에 비해서 subgraph matching 의 경우에는 다수의 embedding 의 순서까지 모두 일치해야 하기 때문에 검증의 신뢰성이 올라가는 특성이 있다.

Chapter 5

성능 평가

본 장에서는 제시된 각각의 기법이 얼마 만큼의 성능 개선에 도움이 되는지 실제 그래프 데이터을 활용한 실험을 통해 입증한다.

5.1 Subgraph isomorphism

5.1.1 실험세팅 및 측정 방법

Verification 단계에서 기법의 사용 유무를 기반으로 아래 4가지 알고리즘에 대해서 실험을 진행한다. Filtering 방법의 경우에는 DAF 에서 사용한 DAG-Graph DP 를 그대로 사용한다.

- DA : DAF 에서 failing set 을 사용하지 않은 경우.
- DAF : Failing set 을 사용한 기존 DAF
- DAB : DAF 에서 failing set 을 사용하지 않고 maximum bipartite matching 을 기반으로 한 pruning 을 사용한 경우.
- DABF : Failing set 을 사용한 기존 DAF 에 maximum bipartite matching 을 기반으로 한 pruning 까지 추가적으로 사용한 경우.

DAF 소스는 저자로 부터 제공받았으며, C++ 으로 구현하였다. 실험은 CentOs 운영체제를 돌리는 Intel Xeon ES-2680 v3 2.5GHz CPUs 와 265GB 의 메모리를 보유하는 서버에서 실험을 진행했다.

데이터 IMDB 과 COLLAB [11], PCMS [6], and PPI [9] 4종류의 실제 데이터에 그래프에 대해서 실험을 진행한다. COLLAB 의 경우에는 label 이 없기 때문에 균등 분포에 기반하여 라벨을 무작위로 배정하였다. 전반적인 그래프에 대한 정보는 Table 5.1 에 소개되어 있다. 표의 각각의 column 은 다음과 같은 의미를 가진다.

- $|D|$: 데이터 그래프의 개수
- $|\Sigma|$: 라벨의 개수
- $|V(G)|$: 정점 개수
- $|E(G)|$: 간선 개수
- degree : 정점 차수

Table 5.1: Characteristics of real-world datasets

	Dataset		Average per graph			
	$\|D\|$	$\|\Sigma\|$	$\|V(G)\|$	$\|E(G)\|$	degree	$\|\Sigma\|$
PCMS	200	21	377	4,340	23.01	18.9
PPI	20	46	4,942	26,667	10.87	28.5
IMDB	1,500	10	13	66	10.14	6.9
COLLAB	5,000	10	74	2,457	65.97	9.9

쿼리 셋 쿼리는 CFQL [9] 논문에서 사용했던 random-walk 와 BFS (breath first search) 을 사용해서 생성한다. 각각의 데이터 그래프로부터 Q_{iR} (i.e., random-walk) 와 Q_{iB} (i.e., BFS) ($i \in \{8, 16, 32, 64\}$) 의 8종류의 쿼리셋을 생성한다. 여기서의 i 는 쿼리 그래프를 구성하는 간선의 개수이다. 각각의 쿼리셋은 100개의 쿼리 그래프로 구성된다.

쿼리 처리 시간(Query Processing Time) Query processing time 은 filtering 시간과 verification 시간의 합으로 계산한다. 각각의 데이터셋은 여러 개의 데이터 그래프로 구성되어 있으며, 쿼리 하나가 모든 데이터 그래프에 대해서 subgraph isomorphism 을 푸는데 소모된 시간의 누적값을 측정한다. 특정 쿼리에 대해서는 지나치게 긴 시간이 소모될 수 있기 때문에 10분의 제한시간으로 설정하며, 만약 10분 이내에 subgraph isomorphism 을 풀지 못했을 경우에는 Query processing time 을 10분으로 지정한다. 측정은 ms 단위를 사용하였다. 각각의 쿼리셋의 모든 쿼리의 query processing time 의 평균을 구한다. 단, 100개의 쿼리 중에서 4개의 알고리즘 중 적어도 하나라도 제한시간 10분 이내에 문제를 해결한 쿼리들에 대해서만 평균을 구한다.

Number of solved queries 각 100개의 쿼리셋의 쿼리 중에서 얼마나 많은 데이터 그래프에 대해서 subgraph ismorphism 을 해결했는지를 계산하는 것이다. 만약 특정 쿼리가 10분이내에 모든 데이터 그래프에 대해서 subgraph isomorphism 을 해결했다고 하더라도 시간이 10분에 가깝더라면 차이가 눈에 띄지 않을 수 있기 때문에 측정했다.

5.1.2 결과 분석

각 쿼리 셋에 대한 query processing time 의 평균은 fig. 5.1 에서 확인이 가능하다. Y 축이 시간 축이며 단위는 ms 를 사용으고 log scale 으로 표현했다. 하나의 쿼리 셋에 대해서 막대 그래프는 좌측부터 차례대로 DA, DAF, DAB, DABF 의 query processing time 의 평균을 보여준다. iB 의 경우에는 BFS 으로 생성한 i 개의 간선을 가지는 쿼리셋을 의미하며 iR 의 경우에는 random walk 으로 생성한 i 개의 간선을 가지는 쿼리셋을 의미한다 ($i \in \{8, 16, 32, 64\}$).

COLLAB 64R 의 경우에는 4가지의 알고리즘이 100개의 쿼리에 대해서 모두 정해진 시간에 문제를 해결하지 못해서 공란으로 두었다.

DA 와 DAB 비교에서는 전반적으로 query processing time 에서 개선이 있었다. 특히 그래프 사이즈가 커지면서 DA 에서 시간이 증가하는 쿼리 셋의 경우에는 DAB 에서 성능 개선이 있는 경향이 있었다. 특히 PPI 16R, 64B 과 IMDB 64R, COLLAB 32B 같은 경우에는 DA 에 비해서 DAB 에서 최대 100배에 가까운 많은

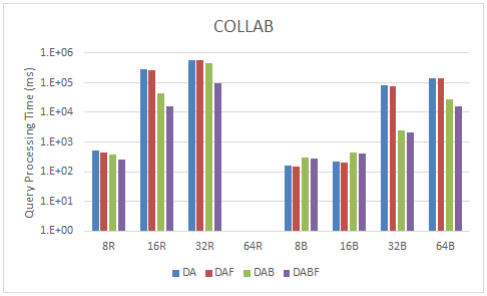
시간 개선이 있었다. 이는 제시했던 문제에 의해서 불필요한 search space 의 방문이 실제로 많이 일어나고 있는것을 간접적으로 보여주고 있으며 이러한 문제를 제시한 maximum bipartite matching 을 활용한 pruning 이 효율적으로 해결하고 있음을 입증한다.

예외적으로 DA 에서 query processing time 이 짧았던 경우에서 전반적으로 DAB 에서 시간이 단축의 효과가 없거나 오히려 시간이 증가하는 경우도 있었다. 이는 full embedding 을 적은 횟수만으로 찾아버리기 때문에 앞서 언급된 문제의 발생 빈도가 적어서 pruning 의 효과를 보지 못하고 오히려 maximum bipartite matching 을 구하는 오버해드가 시간을 증가시키는 원인으로 작용했기 떼문이라고 추측한다.

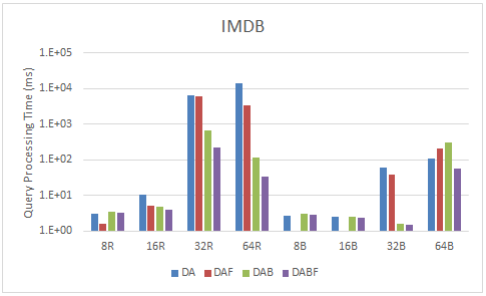
DAB 와 DABF 의 query processing time 의 평균을 계산했을 때 failing set 을 계산해서 pruning 을하는 과정을 추가한 DABF 가 더 전반적으로 성능이 더 올라가는 것을 확인할 수 있다. 특히 pcms64R 이나 PPI 32, 64R 의 경우 최대 이분 매칭으로 발생한 pruning 에 추가적으로 더 많은 시간의 개선이 있었다. 최대 200배의 개선이 있었다. 이는 제안한 failing set 을 계산하는 방법이 효과적으로 failing set 을 계산하고 있음을 의미한다.

최종적으로는 모든 기법을 함께 사용한 경우인 DABF 알고리즘이 기존 state-of-the-art 인 DAF 에 비해서 전반적으로 쿼리의 크기가 증가할 수록 성능의 개선이 최대 1,000배 까지 되는 등 더 scalability 가 있음을 입증할 수 있었다.

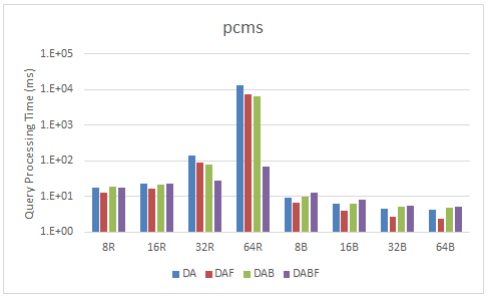
각 쿼리 셋의 100개의 쿼리들 중 알고리즘이 10분 이내로 문제를 해결한 쿼리의 개수는 fig. 5.2 에서 확인이 가능하다. COLLAB 의 경우에는 네 알고리즘이 query processing time 의 측면에서는 큰 개선을 확인하기 힘들었지만, 해결한 쿼리의 개수의 측면에서는 maximum bipartite matching 을 사용하는 pruning 을 사용함으로써 성능 개선이 있음을 관찰할 수 있다. 이러한 결과의 원인은 COLLAB 쿼리 셋에서 문제를 해결한 쿼리가 10분에 가까운 시간을 소모해서 시간 상의 차이는 나지 않은 것이기 때문인 것으로 추측한다. 제한 시간을 늘렸을 경우에 처리 시간에서도 차이가 벌어질 것으로 예상된다.



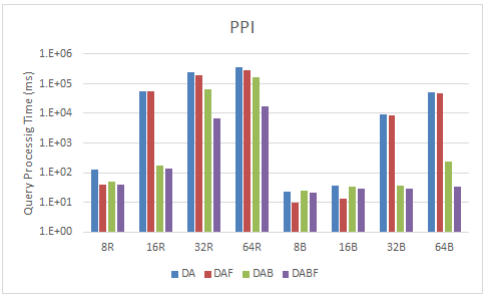
(a) COLLAB



(b) IMDB



(c) pcms



(d) PPI

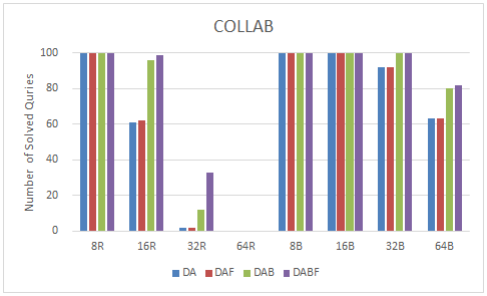
Figure 5.1: **Query processing time**

5.2 Subgraph matching

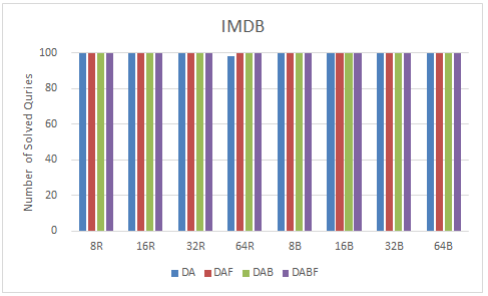
Subgraph matching 은 subgraph isomorphism 과 다르게 존재하는 embedding 을 전부 찾는 문제이다. 다만, 이전 연구들에서는 모든 embedding 을 찾는 것은 지나치게 많은 시간을 소모하기 때문에 특정 개수 만큼만 embedding 을 찾는것으로 대체하는 경우가 많았다. 본 논문에서는 DAF 에서와 같이 10^5 개의 embedding 만 찾는 세팅을 사용하였다.

5.2.1 실험세팅 및 측정 방법

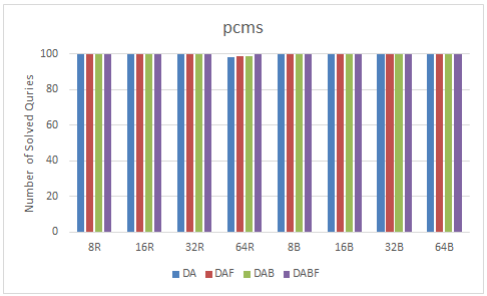
Human, Yeast, DBLP 그리고 EMAIL 을 실제 그래프 데이터로 사용했다. Yeast 와 Human 의 경우에는 단백질간의 상호작용을 그래프로 모델링 한 데이터로 [1, 4] 등 기존 연구에서 많이 사용된 데이터이다. EMAIL 과 DBLP 데이터는 각각 email 사용자 간의 통신 관계를 그래프로 모델링한 그래프 데이터, DBLP 에서의 엽구 협력 관계를 그래프로 모델링한 데이터이다. 각각은 SNAP(Stanford



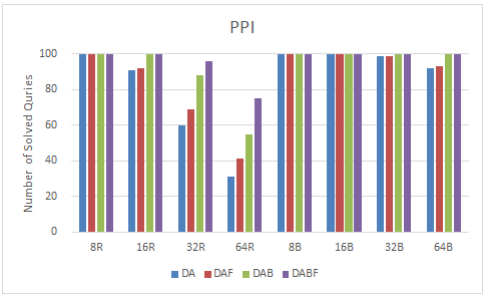
(a) COLLAB



(b) IMDB



(c) pcms



(d) PPI

Figure 5.2: **Number of solved queries**

Large Network Dataset Collection) data sets 에서 구할 수 있다 [7].

데이터 그래프로부터 쿼리 선별 방식은 [4] 에서 사용한 sparse, non-sparse 선별 방식을 사용했다. 각 데이터 셋에서 선별하는 쿼리 데이터의 크기는 DAF 와 동일하게 Yeast 의 경우에는 정점의 개수 i를 ($i \in \{50, 100, 150, 200\}$) 개로 변경하며 쿼리 그래프를 추출하며 DBLP, EMAIL 그리고 Human 의 경우에는 정점의 개수 i를 ($i \in \{10, 20, 30, 40\}$) 으로 변경하면서 그래프를 선별한다.

DBLP 과 EMAIL 의 경우에는 레이블이 없는 그래프이기 때문에 균등 분포에 기반하여 레이블을 임의로 20개의 후보군에서 선택하여 부여했다.

각 데이터 셋은 하나의 데이터 그래프로 구성되어 있으며, 하나의 쿼리 셋에 대해서 하나의 데이터에 대해서만 문제를 푸는 세팅이다. 입력으로 들어온 쿼리 그래프에 대해서 탐색하는 embedding 의 개수를 10^5 개로 제한했음에도 여전히 특정 그래프에서는 지나치게 긴 시간을 소모하기 때문에 제한시간을 subgraph isomorphism 실험과 동일하게 10분으로 설정했다. 측정 값은 Subgraph isomorphism 과 동일 세팅으로 query processing time 의 평균과 쿼리 셋을 구성하는 100

개의 그래프에 대해서 제한 시간 안에 해결한 쿼리 개수를 측정한다.

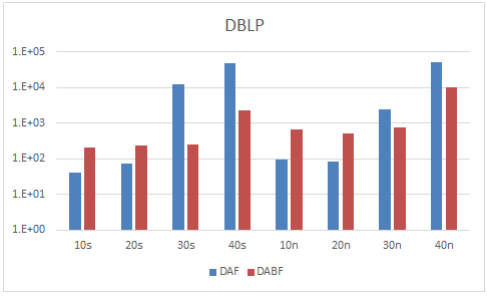
추가적으로 verification 을 진행하는 과정에서 찾은 embedding 의 집합과 탐색된 순서가 동일한지를 비교한다. 만약 제시한 기법이 정당한 기법이라면 양쪽 알고리즘에서 모두 측정 값이 동일해야한다. DAF 와 DABF 간의 비교를 진행한다. Filtering 방법은 DAF 의 DAG-DP 를 기반하는 방법을 그대로 사용했다.

5.2.2 결과 분석

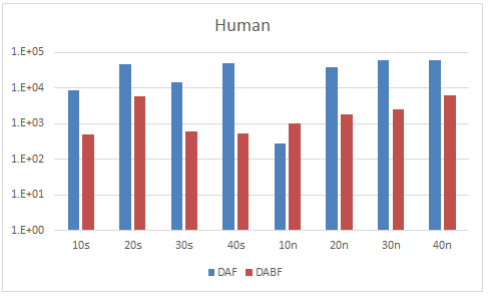
검증 실험의 경우 모든 쿼리 셋에 대해서 DAF 와 DABF 양쪽 찾은 embedding 의 집합과 찾은 순서도 동일했다. 이로써 실험적으로 maximum bipartite matching 을 사용한 가지치기 기법과 pruning-class 에 대해서 정의한 failing set 을 계산하는 방법 및 구현이 기대했던 것과 같이 정상적으로 동작하고 있음을 실험적으로 검증했다고 할 수 있다.

각각의 query processing time 의 평균은 fig. 5.3 에서 확인이 가능하다. 단위는 ms 이며 local scale 으로 표현이 되어있다. X축의 iN 과 iS 은 각각 i 개의 정점을 가지는 sparse 형태로 추출한 쿼리 그래프, non-sparse 형태로 추출한 퀴리 그래프로 구성된 쿼리 셋을 의미한다. 쿼리 그래프의 크기가 작을 때는 기법을 적용한 경우가 더 오랜 시간을 소모하는 경우가 존재했으나 그래프의 사이즈가 커질 수록 기법을 적용한 DABF 가 DAF 보다 더 좋은 성능을 보여주는 것을 확인 할 수 있다. 특히 Yeast 데이터 그래프 상에서 진행한 실험의 경우에는 200N 데이터에 대해서 약 1000 배 정도의 시간 개선이 있음을 확인할 수 있다. 이로써 시간의 측면에서도 제시한 기법이 scalability 가 우수함을 입증할 수 있었다.

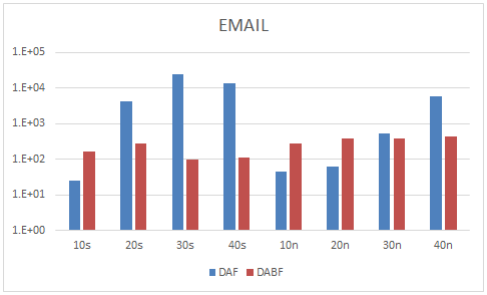
각각의 쿼리 셋에서 전체 쿼리 그래프 중에서 해결한 문제의 결과는 fig. 5.4 에서 확인이 가능하다. 이 측정치에서도 DABF 가 DAF 보다 더 우수한 성능을 보이는 것을 확인할 수 있었다. 특히 Yeast 에서는 그 차이가 많이 벌어지는 것을 확인할 수 있다.



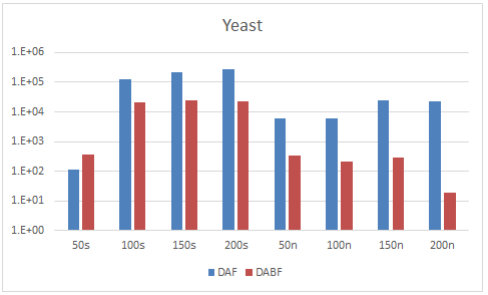
(a) DBLP



(b) Human

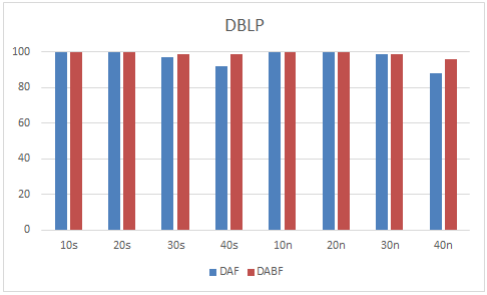


(c) EMAIL

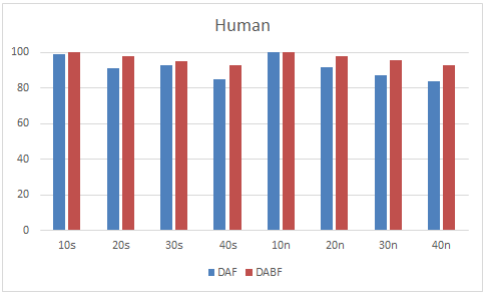


(d) Yeast

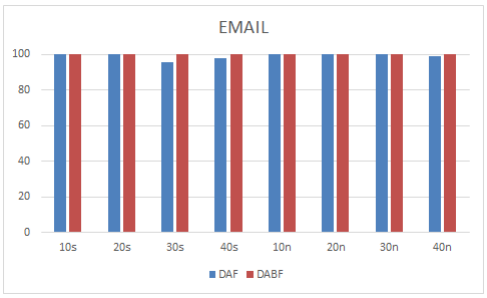
Figure 5.3: **Query processing time**



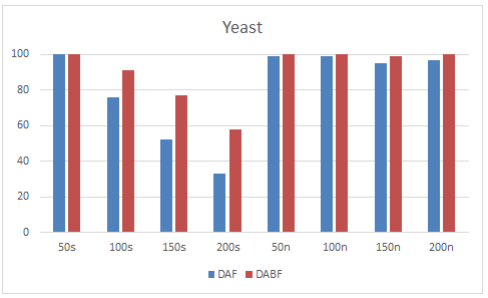
(a) DBLP



(b) Human



(c) EMAIL



(d) Yeast

Figure 5.4: **Number of solved queries**

Chapter 6

결론

본 논문에서는 subgraph ismoprhism 알고리즘의 backtracking 과정에서 사용될 수 있는 효율적인 가지치기 기법에 대해 소개했다. 또한, state-of-the-art subgraph isomorphism 알고리즘인 DAF에 기법을 적용을 한 구현을 다양한 실제 그래프 데이터 상에서 실험을 진행함으로써 제시한 기법이 효율적임을 입증했다.

DAF 에서 소개된 효과적인 가지 치기 기법중 하나인 pruning by failing set 을 제시한 기법과 함께 사용할 수 있도록 DAF 에 없는 새로운 유형의 partial embedding 에 대해서 failing set 을 계산하는 방법을 제시했다. 실험을 통해서 제시한 계산 방법을 통해서 DAF 가 제시한 pruning by failing set 의 효과를 유지시켰다는 것을 확인할 수 있었다.

새로운 기법과 pruning by failing set 을 동시에 사용했을경우 state-of-the-art subgraph isomorphism 알고리즘인 DAF 에 비해서 전반적으로 더 좋은 성능을 보였으며 특히 기존에 많은 시간을 소모하는 크키가 큰 쿼리셋에 대해서 좋은 성능을 보이는 것을 통해서 제시한 기법을

Subgraph isomorphism 문제와 관련이 있는 subgraph matching 문제에서 진행한 실험을 통해서 제시한 기법들의 구현이 정상적인 동작을 한다는 사실을 실험을 통해서 입증했다. 게다가 subgraph matching 문제를 해결하는 데에도 효과적임을 입증했다.

추후에는 다양한 subgraph isomorphism 알고리즘에 이 기법을 적용했을때 어떠한 성능의 향상이 있을지 조사함으로써 제시한 기법이 정말로 범용적인지 입증해야한다. 추가적으로 해당 논문의 실제 구현에서는 unmatchable 인 connected component 중에서 임의로 하나를 선택했지만, 여러 개의 후보군이 있을 경우에는 어떠한 것을 선택하는 것이 성능 개선에 도움이 될지도 추후에 연구해봐야 할 것이다.

Bibliography

[1] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214, 2016.

[2] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[3] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21, 2012.

[4] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1429–1446, 2019.

[5] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases.

In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2013.

[6] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. Performance and scalability of indexed subgraph query processing methods. *Proceedings of the VLDB Endowment*, 8(12):1566–1577, 2015.

[7] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[8] N Pržulj, Derek G Corneil, and Igor Jurisica. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics*, 22(8):974–980, 2006.

[9] Shixuan Sun and Qiong Luo. Scaling up subgraph query processing with efficient subgraph matching. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 220–231. IEEE, 2019.

[10] Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346, 2004.

[11] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

Abstract

Improving Subgraph Isomorphism with Pruning by Maximum Bipartite Matching

Yunyoung Choi
Department of Computer Science and Engineering
College of Engineering
The Graduate School
Seoul National University

In recent years, graphs have been playing an increasingly important role in various domains, e.g., social networks [3], bioinformatics [8], chemistry [10] etc. One of the most fundamental problems in graph analysis is subgraph isomorphism. Many practical solutions have been suggested for subgraph isomorphism. However, those algorithms show limited response time and scalability in handling real-world applications because, by the nature of backtracking, there could be many redundant computations.

In this paper, we develop a new technique to prune out some parts of the search space. Furthermore, we incorporate our method to one of them and show the efficacy by conducting experiments on several real-world datasets.

Keywords: Subgraph isomorphism
Student Number: 2019-27354