과제 2 보고서

Homework 2 Report

공과대학 기계공학부 2019-19568 이지훈
공과대학 컴퓨터공학부 2019-15099 박재현

초록 : 본 과제에서는 DAG-Graph DP를 통해 필터링한 Candidate Set에서 시간 안에 최대한 많은 embedding이 발견되는 DAG-Ordering을 찾는 것을 목표로 하여 C++를 사용했다. 그리고 DAG-Ordering에 필요한 Adaptive Matching Order을 각 extendable Vertex u에 $key(u)$값을 부여해 그 값이 작은 순서로 정했다. 또한 ϵ을 바꿔가며 구현된 프로그램의 성능을 비교한 결과 ϵ이 0.75일 때 가장 최적의 성능을 보였다. 이에 특정 ϵ값에서 높은 성능이 보여지는 이유 및 값이 결정되는 이론적 근거에 대한 연구가 아직 미진하므로 추가로 연구해야 한다. 또한 $key(u)$ 내 다른 요소의 지수도 변화시키거나, 다양한 조합의 계산으로 $key(u)$를 정하거나, 새로운 변수를 도입하는 등 $key(u)$의 형태를 바꾸어가며 최적의 형태를 결정하는 후속 연구가 필요하다.

Ⅰ. 서론

1. 선행 연구 및 이론적 배경

Subgraph matching은 주어진 undirected, connected, vertex-labeled query, data 그래프에 대해 query vertices에서 data vertices로의 가능한 모든 injective homomorphism, 즉, embedding을 찾는 문제이다.[1] 이 문제는 NP-hard 문제로 알려져 있어[2] 이를 해결하는 Turbo[3], CFL-Match[4] 등 다양한 알고리즘들이 연구되어 왔고, 최근에는 DAF[5]라는 더 빠르고 새로운 알고리즘이 제안되었다.

이들 알고리즘은 공통적으로 다음과 같은 과정으로 구성된다.[6] 먼저 각 query vertex별로 대응될 가능성이 있는 후보 data vertex들을 특정한 방법을 통해 필터링한다. 이 때 걸러진 후보 data vertex들의 집합을 Candidate Set이라 한다. 이후 query 그래프를 backtracking하여 각 query vertex의 Candidate Set에 있는 vertex가 embedding을 성립시키도록 query vertex와 mapping하면서 가능한 모든 embedding 조합을 찾는다.

특히 DAF에서는 DAG-Graph DP를 통한 필터링 과정과 DAG-ordering을 이용한 Backtracking framework를 사용한 알고리즘으로 높은 성능을 보여준다.[7]

2. 연구 목적

본 과제에서는 DAG-Graph DP를 통해 필터링한 Candidate Set에 대해, 제한 시간 안에 최대한 많은 embedding을 포함하는 DAG-Ordering을 찾는 것을 목표로 한다.

Ⅱ. 연구과정 및 방법

1. 연구 과정

1.1 환경

Macbook Pro (15-inch, 2018), 2.9GHz 6-core Intel Core i9, 32GB 2400MHz DDR4 SDRAM에 macOS Big Sur 11.3을 탑재하여 사용했다. 사용 언어는 C++, Apple clang version 12.0.5이다.

1.2 연구 방법

Vertex를 탐색하며 extendable vertex를 택하는 순서를 Adaptive Matching Order라 하는데, 이 순서를 정하기 위해 특정한 계산법으로 실수 값을 각 extendable vertex에 부여하고 그 값이 작은 순서대로 택한다. 이 때 계산법에 포함되는 특정한 지수 값을 구체화하기 위해 최적의 지수를 찾는 실험을 진행한다.

실험을 위해 각주의 repository[8]를 수정해 개수 제한 없이 embedding을 출력하도록 하여 결과를 저장한다. 또한, 3개의 data graph별 8개씩의 query graph의 조합인 총 24개의 benchmark test case에 대해, 1분의 제한 시간 내에 출력하는 embedding을 저장하고 이를 checker program의 input으로 주어 그 정확성과 출력된 embedding의 수를 얻는다.

---

1) M. Han, H. Kim, G. Gu, K. Park, and W. Han.2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In Proceedings of the 2019International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1429–1446.

2) M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman &Co., 1979.

3) W. Han, J. Lee, and J. Lee. Turbo iso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In Proceedings of SIGMOD, pages 337–348, 2013.

4) F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient Subgraph Matching by Postponing Cartesian Products. In Proceedings of SIGMOD, pages 1199–1214, 2016.

5) M. Han, H. Kim, G. Gu, K. Park, and W. Han, op.cit.

6) loc. cit.

7) loc. cit.

8) https://github.com/SNUCSE-CTA/Graph-Pattern-Matching-Challenge.git

2. 실행 방법

기본 Graph Pattern Matching Challenge의 repository에서의 방법과 같은 방법으로 실행한다. 구현한 코드는 각주[9]의 Github repository에 있고 알고리즘 1이 C++로 작성되어 있다.

III. 연구결과

1. $key(u)$ 계산법

$$key(u) = \frac{|\text{extendable Candidates(u)}|}{(|\text{mapped parents(u)}|+1)\text{degree}(u)^{\epsilon}} \quad (1)$$

식 1. Adaptive matching Order을 위한 Extendable Vertex의 key

2. 지수 ϵ 결정

2.1 Backtracking framework pseudo code

```
PrintAllEmbeddings(datagraph, query graph, Candidate Set)

ExtendableVertex = Red-Black Tree<key, Vertex>();
for each Vertex v in query {
insert v in ExtendableVertex;
}
vertex is mapped last time = true;
do {
        if (vertex is mapped last time) {
extract minimum Vertex u in ExtendableVertex;

} else {//backtracked before
Get lastly extended query vertex u of the partial embedding;
Restore all filtered extendable Candidates in the last iteration;
}

map u with a vertex v in Candidate set which is not examined;

if (no such v exists) {
        insert u to ExtendableVertex back;
vertex is mapped last time = false;
continue;
}

Filter each vertex of extendable Candidates of neighbor of u by checking v and the vertex is a neighbor;

If (filtering made one of extendable Candidates empty) {
vertex is mapped last time = false;
continue;
}

Extend partial embedding by (u, v);

If (size of partial embedding = |query Vertices|) {
        print embedding;
vertex is mapped last time = false;
} else {
vertex is mapped last time = true;
}
} while (size of partial embedding > 0)
```

알고리즘 1. Backtracking framework based on the DAG-Ordering

I-1.에 소개된 Backtracking framework를 **알고리즘 1**로 구현했다.

2.2 지수 결정 실험

식 1의 ϵ값을 바꿔가며 backtracking을 반복해 성능을 비교했다.

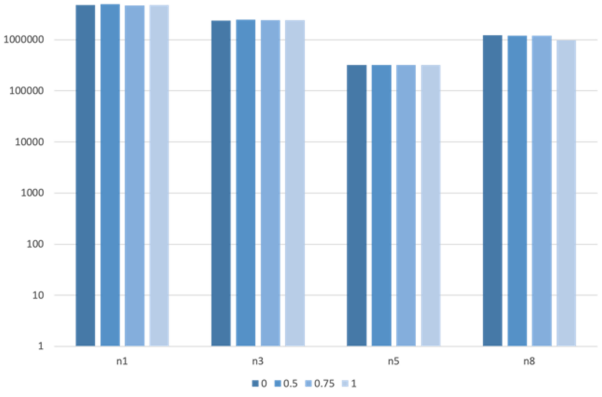

그림 1. lcc_yeast n1, n3, n5, n8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

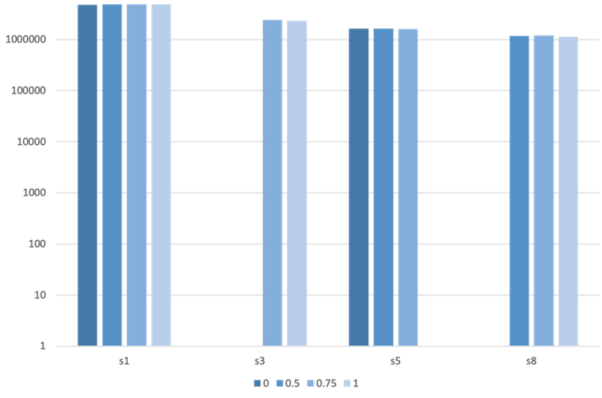

그림 2. lcc_yeast s1, s3, s5, s8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

9) https://github.com/dlwl0088/PMC21.git

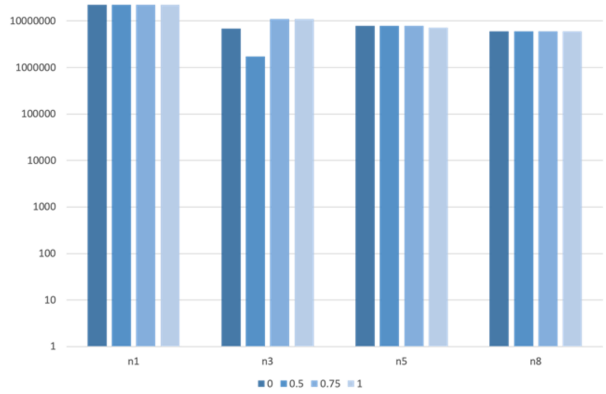

그림 3. lcc_human n1, n3, n5, n8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

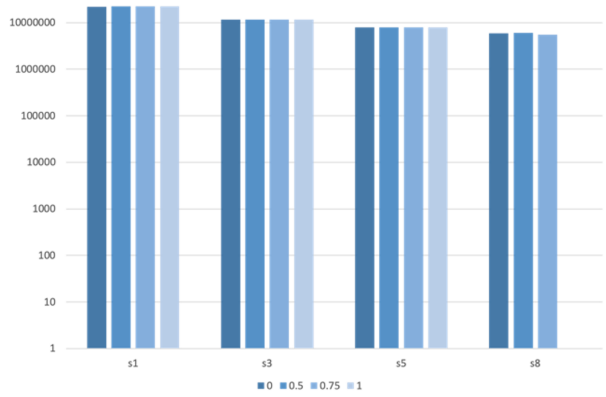

그림 4. lcc_human s1, s3, s5, s8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

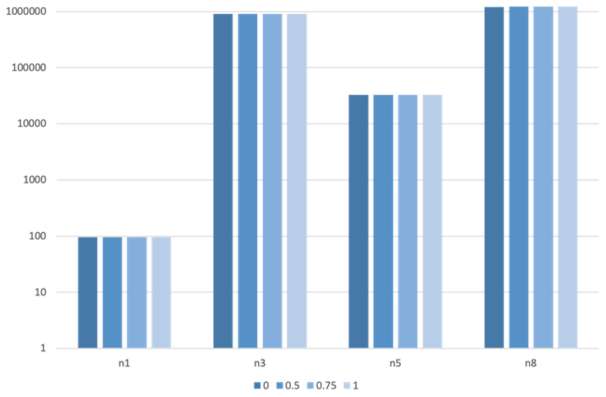

그림 5. lcc_hprd n1, n3, n5, n8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

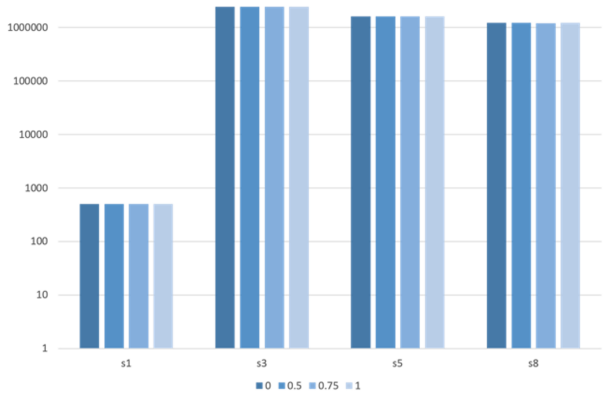

그림 6. lcc_hprd s1, s3, s5, s8 case에서 ϵ의 값에 따라 1분간 출력된 embedding의 수

먼저 checker program으로 확인한 결과, 구현한 알고리즘은 모든 input에 대해서 올바른 embedding을 출력했다.

또한 상당수의 test case에서 embedding의 수가 ϵ에 무관했지만, **그림 2, 4**의 경우 일부 ϵ에서 embedding을 출력하지 못하는 경우가 존재했다. 또한 모든 testcase에서 embedding을 하나 이상 출력한 ϵ의 값은 0.75였다. 이 때 모든 test case에서 다른 ϵ값에 비하여 더 많거나 비슷한 수의 embedding을 출력했다.

Ⅳ. 결론

먼저 결과를 통해 ϵ값에 따라 출력하는 embedding의 개수에 편차가 있었고, 일부 경우의 특정 ϵ값은 embedding을 출력하지 못함을 확인했다. 이는 **알고리즘 1**을 보면 첫 번째 embedding을 출력하기 위해 구조상 많은 backtracking이 발생하기 때문에 시간이 걸리는 것으로 보인다. 이를 통해 ϵ값에 따른 corner case가 존재해 첫 embedding의 출력까지 시간이 많이 걸리는 경우도 존재한다는 것을 알 수 있다.

반면 상당수의 test case에서 ϵ값에 따른 embedding의 수에 큰 편차가 없었는데, thermal throttling에 의한 실험 기기의 성능 저하를 고려했을 때 Adaptive matching order에 따른 알고리즘 성능의 편차가 크지 않음을 알 수 있다.

위 실험에서는 vertex degree의 지수를 변경했지만, 다른 요소의 지수를 바꾸거나 key의 계산법에 변화를 주어 성능을 추가로 연구해보아야 한다. 또한 ϵ이 0.75일 때 종합적인 benchmark test case에서 최적이라고 판단했으나, 최적값이 왜 존재하는지, 이 값의 의미는 무엇인지에 대한 이론적 근거가 부족하다. 따라서 이것이 subgraph matching 문제의 속성에 의한 것인지 아니면 우연에 의한 것인지를 가리는 후속 연구가 필요하다.

Ⅴ. 참고문헌

[1] M. Han, H. Kim, G. Gu, K. Park, and W. Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In Proceedings of the

2019International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1429-1446.

[2] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H.Freeman & Co., 1979.

[3] W. Han, J. Lee, and J. Lee. Turbo iso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In Proceedings of SIGMOD, pages 337-348, 2013.

[4] F.Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient Subgraph Matching by Postponing Cartesian Products. In Proceedings of SIGMOD, pages 1199-1214, 2016.