ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 25, No. 4, pp. 215-222, 2019. 4
https://doi.org/10.5626/KTCP.2019.25.4.215

관계형 데이터베이스 상에서 사용하는 최단 경로 탐색 알고리즘(ALT) 설계와 구현

(Design and Implementation of Shortest Path Search Algorithm(ALT) for Relational Database)

이 용 현 † 임 동 혁 †† 구 해 모 ††† 김 형 주 ††††
(YongHyun Lee) (DongHyuk Im) (Heymo Kou) (Hyoung-Joo Kim)

요 약 최단 경로 탐색은 교통 정보, 네비게이션, 컴퓨터 네트워크 등 그래프로 표현될 수 있는 다양한 도메인에서 사용되는 주제이다. 빅데이터 시대가 도래함에 따라 이러한 그래프 데이터들은 과거에 비해 그 정보와 크기가 비교 불가능할 정도로 커지고 있다. 이에 따라 보다 큰 그래프를 효과적으로 탐색 및 분석할 필요성이 대두되었다. 이를 해결하고자 데이터베이스 내부에서 직접 그래프 분석을 하는 여러 연구들이 진행되어 왔다. 본 논문에서는 최단 경로를 관계형 데이터베이스 내부에서 직접 휴리스틱하게 탐색하는 In-RDBMS ALT 알고리즘(A* 알고리즘, Landmark, Triangle Inequality)을 제안한다. 또한 기존의 In-RDBMS 최단 경로 탐색 기법과의 성능 비교를 통하여 In-RDBMS ALT 알고리즘의 효율성에 대해서도 논의해보고자 한다.

키워드: ALT 알고리즘, 최단 경로 탐색, In-DB 분석, 관계형 데이터베이스

Abstract Shortest path search is a major topic in various domains that can be represented by graphs such as traffic information, navigation and computer network. As we approach the age of big data, these graph data and information are getting larger and larger in size compared to the past. Therefore, there is the need to effectively search and analyze large graph data. In order to solve this problem, there have been several researches on graph analysis directly in the database. In this study, an In-RDBMS ALT algorithm(A* algorithm, Landmark, Triangle Inequality) that heuristically searches the shortest paths directly in relational database is proposed. The efficiency of In-RDBMS ALT algorithm is also discussed through comparison with existing In-RDBMS shortest path search algorithm.

Keywords: ALT algorithm, single-source-shortest path, in-db analysis, relational database

· 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.R0113-15-0005, 대규모 트랜잭션 처리와 실시간 복합 분석을 통합한 일체형 데이터 엔지니어링 기술 개발)

† 비 회 원 : 서울대학교 컴퓨터공학과(Seoul Nat'l Univ.)
leeyh@idb.snu.ac.kr
(Corresponding author임)
†† 비 회 원 : 호서대학교 컴퓨터정보공학부 교수
dhim@hoseo.edu
††† 비 회 원 : 서울대학교 컴퓨터공학과
hmkou@idb.snu.ac.kr
†††† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@snu.ac.kr

논문접수 : 2018년 6월 28일
(Received 28 June 2018)
논문수정 : 2018년 12월 19일
(Revised 19 December 2018)
심사완료 : 2019년 1월 9일
(Accepted 9 January 2019)

정보과학회 컴퓨팅의 실제 논문지 제25권 제4호(2019. 4)

1. 서 론

그래프는 두 도시간의 길을 안내하기 위한 네비게이션, 머신들 간의 네트워크, 웹 페이지 링크, 소셜 네트워크 서비스(SNS) 등 하나의 지점을 나타내는 노드(Node)와 노드들 사이의 관계를 나타내는 엣지(Edge)들로 표현될 수 있는 다양한 분야에서 사용되고 있다[1]. 해당 분야의 기술이 발전하고, 참여하는 대상들이 늘어남에 따라 해당 그래프들의 크기는 점점 커지고 있다. 이에 따라 그래프를 저장하는 데에 더 많은 저장 공간을 필요로 하게 되었다. 미국 전 지역의 건물 위치에 대한 그래프나 페이스북, 트위터 등 SNS 사용자들간의 네트워크 같이 일반적인 메인 메모리의 용량보다 큰 그래프 데이터를 다루고자 할 경우, 이는 반복적인 디스크 I/O를 야기하게 되고 직접적인 성능의 저하로 이루어진다. 본 논문에서는 매우 큰 사이즈의 그래프를 대상으로 한 분석에서 데이터를 분석 프로그램에 불러올 필요없이 RDBMS(Relational DataBase Management System) 내부에서 직접 분석하는 In-Database 적인 기법을 제안한다.

하나의 그래프에서도 사용자가 원하는 정보가 무엇인지에 따라 두 노드 사이의 최단 경로를 찾는 것이 목적일 수도 있고, 모든 노드 쌍들 간의 최단 경로를 필요로 할 수도 있으며, 최소 신장 트리를 구하는 등 그래프에 적용할 수 있는 알고리즘들은 매우 다양하다. 이러한 알고리즘들 중 본 논문에서는 네비게이션, 네트워크 라우팅, 컴퓨터 게임, SNS에서의 관계 탐색, 대중 교통 등 많은 어플리케이션에서 실제로 사용되며[1], 그래프 탐색 쿼리에서 대표적인 두 노드 사이의 최단 경로 탐색 알고리즘을 RDBMS에서 In-Database 방식으로 구현하고자 한다. In-RDBMS에서의 최단 경로 탐색에 대한 개념은 J. Gao et al.이 제안하였고[2], 자신들이 제안한 FEM 프레임워크를 사용하여 Dijkstra 알고리즘을 RDBMS에서 구현한 바 있다. 본 논문에서는 보다 효과적인 최단 경로 탐색을 위하여 휴리스틱한 방법을 제안한다.

최단 경로 탐색의 대표적인 휴리스틱 알고리즘은 A* 알고리즘인데[3], 해당 지점에서 도착점까지의 거리가 얼마인지를 휴리스틱하게 예측한다. 위도와 경도가 나와 있거나 x값과 y값이 주어진 2차원의 경우 유클리디안 거리(Euclidean distance)를 사용하여 휴리스틱하게 예측하지만 일반적인 그래프에서는 이러한 좌표 값들이 주어지지 않은 경우가 많다. 본 논문에서는 A* 알고리즘을 그래프에 맞게 변형시킨, 엣지에 가중치가 있는 (weighte d edges graph) 그래프를 대상으로 한 ALT 알고리즘[4,5](A* 알고리즘, Landmark, Triangle Inequality)을 In-RDBMS 상에서 사용하는 기법을 제안한다. A. V. Goldberg et al.이 제안한 ALT 알고리즘은 그래프의 특정 노드 혹은 다수 노드들을 Landmark로 설정한 후, A* 알고리즘의 휴리스틱 거리 계산을 위하여 해당 노드들을 이용한 Triangle Inequality 성질을 사용하는 알고리즘이다. 또한 RDBMS상에서 Dijkstra 알고리즘을 구현한 기존의 연구와의 비교를 통하여 ALT 알고리즘의 성능 및 효율성도 논의하고자 한다.

2. 관련 연구

2.1 In-RDBMS Graph Operation

데이터베이스에서 직접 그래프 마이닝을 하거나 그래프 관련 알고리즘을 적용하는 방식이 완전히 새로운 패러다임은 아니다. 그래프 데이터를 다루기 위한 그래프 데이터베이스 또한 이미 존재한다[6,7]. 그래프 데이터베이스의 성능을 향상시키기 위하여 그래프 데이터베이스를 위한 그래프 패턴 관련 알고리즘을 제안하거나 그래프 데이터베이스의 새로운 인덱스 구조를 제안하는 등의 방안은 X. Yan et al.이나 C. Wang et al.에 의해 제안되어 왔다[8,9]. 그러나 그래프 데이터베이스는 오랫동안 발전해온 관계형 데이터베이스에 비해 안정성, 보안성 및 성능이 개선될 필요가 있다[2]. C. Vicknair et al.은 대표적인 그래프 데이터베이스인 Neo4j[6]와 대표적인 관계형 데이터베이스인 MySQL을 비교하며, 그래프 데이터베이스의 인덱스 방식이 문자열 기반이므로 관계형 데이터베이스에 비해 숫자 관련 쿼리가 비효율적이라고 지적하였으며, 보안성 문제 또한 지적하였다[10]. 이러한 문제들을 해결하기 위하여 J. Gao et al.은 관계형 데이터베이스를 사용한 그래프 탐색이 좋은 대안이 될 수 있다고 주장하였다. 그래프 데이터베이스와 관계형 데이터베이스는 데이터 저장, 버퍼, 인덱싱, 최적화 등에서 상당히 많은 부분을 공유하며, 그래프 관련 연산과 관계형 연산이 같이 요구되는 경우라도 관계형 데이터베이스는 이를 충분히 다룰 수 있기 때문이다[2].

2.2 Shortest Path Search Algorithm

최단 경로 탐색 알고리즘은 하나의 출발 노드에서 다른 노드까지의 최단 경로를 찾는 Single-Source Shortest-Path (SSSP)와 모든 노드 사이의 최단 경로를 찾는 All-Pairs Shortest-Path(APSP)로 크게 나뉜다[1]. 본 논문에서 해결하고자 하는 문제는 SSSP이며, Dijkstra 알고리즘이 대표적이다[11]. 또 다른 대표적인 알고리즘인 Bellman Ford 알고리즘은 Dijkstra 알고리즘과 비교하여 음수 값을 가지는 엣지에 대해서도 적용이 가능하며, 음수 값을 가지는 사이클을 탐지도 가능한 장점이 있지만, 비교적 속도가 느린 단점도 있다[1].

그래프에 정확히 명시된 거리만을 가지고 경로를 탐색하는 Dijkstra나 Bellman Ford 알고리즘과 달리 휴리스틱한 방법도 존재하는데, 대표적인 알고리즘이 A* 알고

리즘이다. A* 알고리즘은 출발 노드에서 중간 노드까지의 거리를 고려할 뿐만 아니라 중간 노드에서 도착 노드까지의 휴리스틱 거리를 추측하여 두 거리를 한꺼번에 고려하는 방식이다[3]. 해당 알고리즘은 휴리스틱 함수가 중간 노드에서 도착 노드까지의 거리를 과평가 하지 않는 허용적 휴리스틱(Admissible heuristic)이라면 알고리즘이 최적의 해를 찾는 것이 보장된다. A. V. Goldberg et al.은 A* 알고리즘에 출발 노드와 도착 노드의 중간에 있는 노드를 Landmark로 설정하고, Triangle Inequality를 적용한 ALT 알고리즘을 제안하였다. ALT 알고리즘의 휴리스틱 함수는 실제 거리를 과평가하지 않는 허용적 휴리스틱이다. ALT 알고리즘은 Landmark 노드로부터 다른 노드들까지의 거리와 경로를 미리 계산하는 전처리를 필요로 한다[4,5]. 전처리에 대한 추가적인 비용이 소요됨에도 불구하고, 1회의 전처리만으로 다양한 노드들을 대상으로 한 최단거리 탐색 반복이 가능하고, 전처리의 비용이 크지 않으므로 오버헤드는 크지 않다.

2.3 FEM Framework

J. Gao et al.은 대표적인 최단 경로 탐색 알고리즘인 Dijkstra 알고리즘을 In-RDBMS에서 SQL로 구현하기 위해 FEM 프레임워크를 제안하였다[12]. Dijkstra 알고리즘의 의사(pseudo) 코드는 아래의 알고리즘 1과 같다[11,13]. dist(v)는 출발점에서 v 노드까지의 거리를 의미하며, distw(u,v)는 노드u와 노드 v 사이의 거리를 의미한다.

Algorithm1. Dijkstra Algorithm

```
dist[src] ← 0
for all v in V except src      ( V : all nodes in
graph )
   do dist[v] ← ∞
do Visited ← Ø
do Q ← V
While Q is not empty
   do u ← node in Q with smallest distance
   do Visited ← Visited ∪{ u }
   do remove u from Q
   for each v in u's neighbors
      if dist[v] > dist[u] + distw(u, v)
         then dist[v] ← dist[u] + distw(u, v)
return dist
```

J. Gao et al.이 Dijkstra 알고리즘을 SQL 문으로 보다 효과적으로 구현하기 위하여 제안한 FEM 프레임워크는 다음과 같다. Frontier 연산자는 이미 방문한 노드들 중에서 확장시킬 노드를 frontier 노드로 선택한다. Expand 연산자는 선택된 frontier 노드의 이웃 노드들을 찾음으로써 frontier 노드를 확장한다. Merge 연산자는 확장된 frontier 노드의 이웃 노드들을 visited nodes에 포함시킨다. 그림 1은 이러한 과정을 보여주는데, 이를 하나의 주기로 볼 수 있다[2]. 목적 노드를 찾을 때까지 주기를 반복한다. 각 연산자는 관계형 대수(Relational Algebra)로 나타내어 질 수 있고, 이는 각 연산자들이 SQL문으로 나타내어 질 수 있음을 의미한다. 이러한 3가지 연산을 반복 적용함으로써 Dijkstra 알고리즘을 In-RDBMS에서 구현할 수 있다. 해당 방식으로 구현한 최소 경로 탐색 기법은 대표적 그래프 데이터베이스인 Neo4j를 사용한 방법보다 더 나은 성능을 보였다[2,12].

3. In-RDBMS에서 휴리스틱한 최단 경로 탐색

3.1 ALT 알고리즘

J. Gao et al.이 관계형 데이터베이스에서 최단 경로 탐색을 위하여 FEM 프레임워크로 구현한 Dijkstra 알고리즘은 어떠한 추정 혹은 추측을 하지 않고, 엣지들에 직접 주어진 값만을 사용하여 최단 경로를 탐색한다[2]. 본 논문에서는 A. V. Goldberg et al.이 제안한 휴리스틱 최단 경로 탐색 기법인 ALT 알고리즘을 FEM 프레임워크와 유사한 개념을 사용하여 관계형 데이터베이스 내부에서 구현하는 방안을 제안한다.

출발 노드를 s, 도착 노드를 t, s에서 t로 가는 최단 경로 탐색 중 현재 탐색 혹은 검사 중인 노드를 c, s로부터 c까지의 거리를 d(s,c)라고 가정하겠다. Dijkstra 알고리즘은 s부터 c까지의 거리인 d(s,c)가 최소가 되도록 c를 선택한다. 그러나 ALT 알고리즘에서는 A* 알고리즘과 비슷하게 s에서 c까지의 거리 d(s,c) 뿐만 아니라 c에서 t까지의 거리를 추정하여 함께 고려한다. 즉, d(s,c) + d(c,t)가 최소가 되도록 c를 선택한다. 그래프 각 노드들의 좌표 값들을 알 경우, 휴리스틱한 추정값을 구하는 방안은 여러가지

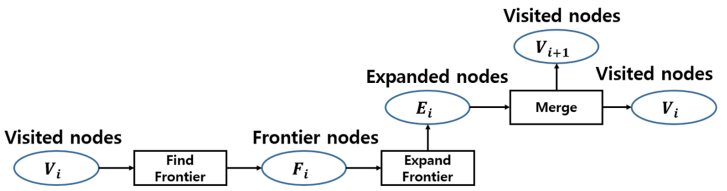

그림 1 FEM 프레임워크 진행 과정
Fig. 1 FEM framework process

제시되어 있다. 예를 들어 2차원 좌표값일 경우 유클리디안 거리를 계산할 수 있다. 그러나 인터넷 웹 상에서의 네트워크나 SNS 사용자간의 네트워크는 좌표 값들로 나타내기가 쉽지 않다. 좌표적인 개념을 도입하기 어려운 해당 그래프들도 휴리스틱한 추정을 가능케 해주는 알고리즘이 ALT 알고리즘이다. ALT 알고리즘은 Lan- dmark 노드를 설정하고, Landmark 노드로부터 다른 노드들 간의 거리와 경로를 전처리로 미리 계산한 값들을 이용하여, 삼각형의 각 모서리의 길이는 다른 두 모서리의 합보다 항상 작다는 성질인 Triangle-Inequality를 적용한 추정값을 결정한다. 그림 2는 ALT 알고리즘의 이러한 작동 순서를 보여준다. 그림 1의 FEM 프레임워크 진행 과정과 비교해 보면, 새로운 노드를 확장할 때 휴리스틱한 거리를 사용하므로 해당 값을 구하는 과정이 추가된 것을 볼 수 있다.

임의의 노드 v, w와 Landmark 노드 l이 있다고 가정할 때, 이들을 꼭지점으로 하는 삼각형의 모서리들은 각각 d(l,v), d(l,w), d(w,v)이다. d(l,v)가 가장 긴 모서리일 경우 d(l,v) - d(l,w) ≤ d(v,w)가 성립하며, d(l,w)가 가장 긴 모서리일 경우 d(l,w) - d(l,v) ≤ d(w,v) 가 성립한다. d(w,v)가 가장 긴 모서리일 경우 d(l,v) - d(l,w) ≤ d(w,v)와 d(l,w) - d(l,v) ≤ d(w,v)가 모두 성립한다. 따라서 | d(l,v) - d(l,w) | ≤ d(w,v)이 항상 성립한다고 볼 수 있다. [4] d(l,v)와 d(l,w)는 전처리된 값을 통해 알 수 있으므로, d(w,v)에 대한 lower-bound를 알 수 있고, 해당 lower-bound를 휴리스틱 추정값으로 사용한다.

위의 방식으로 구해진 lower-bound는 d(w,v)를 과평가하지 않으므로 해당 휴리스틱 함수는 허용적 휴리스틱이고, ALT 알고리즘은 최단 경로 탐색이 보장된다. 출발 노드로부터 현재 노드까지의 실제 거리와 현재 노드로부터 도착 노드까지의 휴리스틱 거리를 합산한 값이 최소가 되도록 현재 노드 선택을 반복하는 것이 ALT 알고리즘이다.

3.2 RDBMS에서의 ALT 알고리즘

RDBMS 내부에서 최단거리 탐색을 위한 ALT 알고리즘을 구현하기 위하여 사용한 테이블들의 구조는 표 1과 같다. data 테이블은 최단거리를 탐색하고자 하는 그래프 구조들을 나타내는 테이블이다. 1개의 행은 1개 엣지의 출발 노드, 도착 노드, 해당 엣지의 거리 정보를 가진다. landmark 테이블은 최단거리 탐색 분석 전에 전처리되어 입력되며, landmark 노드로부터 다른 노드들까지의 거리를 나타낸다. priority_q 테이블은 다음 번에 탐색할 후보군 노드들의 정보를 가지며, ALT 알고리즘의 연산 중간 결과들이 저장된다.

본 논문에서 제안하는 FEM 프레임워크 개념을 차용한 In-RDBMS ALT 알고리즘의 전체 구조 의사코드는 아래의 알고리즘 2와 같다.

알고리즘 1에 설명된 Dijkstra 알고리즘과 알고리즘 2에 설명된 RDBMS에서의 ALT 알고리즘은 매우 비슷한 방식으로 전개된다. 결정적인 차이는 Dijkstra는 출발 노드로부터 현재 노드 c까지의 거리인 dist(c)만을 고려하지만, ALT 알고리즘은 현재 노드 c로부터 도착 노드까지의 예상 거리까지 함께 고려한 alt_dist(c)를 고려한다는 점을 알고리즘 2에서 확인할 수 있다. 그림 2와 알고리즘 2에서 언급하였듯 RDBMS 내부에서의 ALT 알고리즘은 크게 find frontier, expand, merge의 3단계로 구성되어 있다. 각 단계에 대한 SQL 형식의 의사코드는 아래와 같다. 먼저 이미 방문한 노드들의 이웃들 중에서 다음 번에 방문할 노드로 alt_dist가 가장 작은 노드들을 찾는다.

표 1 In-RDBMS ALT 알고리즘을 위한 테이블 스키마
Table 1 Table schema for In-RDBMS ALT algorithm

data	src_id, dst_id, weight
landmark	lmk_id, dst_id, weight
priority_q	id, alt_dist, real_dist, visited

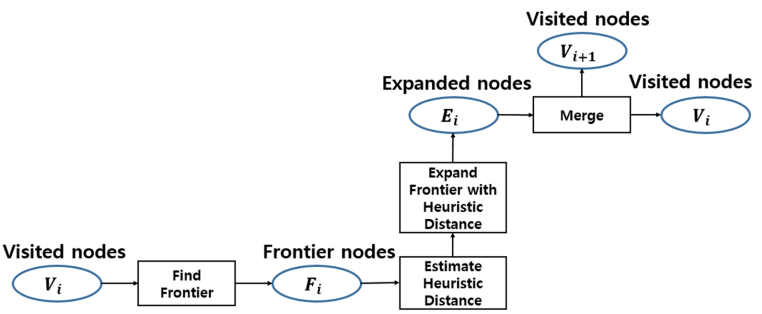

그림 2 ALT 알고리즘 진행 과정
Fig. 2 ALT Algorithm process

Algorithm 2. In-RDBMS ALT Algorithm

```
Load graph data into RDBMS
Load landmark data into RDBMS
Declare priority queue PQ
Insert the source node into PQ

While PQ is not empty
    frontier ←node in PQ with smallest distance
            [ find frontier ]
        do mark frontier as visited
        if frontier == destination node :
                Break
         #  shortest path is found
        candidates ← frontier's neighborhood
nodes
             [ expand ]
         for c in candidates :  [ merge ]
                if c in PQ :
                    if new alt_dist[c] < existing
alt_dist[c] :
                            do update alt_dist[c]
                else :
                        do insert c into PQ
          do remove frontier from PQ
Done
```

```
SELECT id, alt_dist
FROM priority_q
WHERE alt_dist = minimum alt distance in priority_q
AND visited = false
```

위 쿼리의 결과로 다음 번에 방문할 후보 노드군들이 찾아지며, 그 중 1개를 선택하여 frontier 노드로 선택한다. (find frontier 단계) 해당 frontier 노드의 이웃 노드들의 alt_dist를 각각 계산한 후, 이웃 노드가 이미 priority_q 테이블에 있는 경우와 없는 경우 2가지로 나뉘어진다. 이웃 노드가 이미 priority_q 테이블에 있다면, 새로 구해진 alt_dist 값이 priority_q 테이블에 있는 기존 alt_dist보다 작을 경우에만 priority_q의 alt_dist 값을 새로운 값으로 갱신한다. 새로 구해진 alt_dist 값이 기존 alt_dist 보다 클 경우에는 기존 alt_dist 값을 유지한다.

```
UPDATE priority_q
SET alt_dist = ( pq.real_dist + dt.weight + lmk.weight )
     real_dist = ( pq.real_dist + dt.weight )
FROM priority_q as pq, data as dt, landmark as lmk
WHERE pq.id = 'frontier'
AND  frontier's neighbor node in dt
AND  neighbor node.visited = false
AND  new alt_dist of neighbor node is shorter than
     existing alt_dist
```

찾아진 이웃 노드가 이전에 발견 된 적이 없고, 처음 탐색될 경우, priority_q 테이블에 해당 노드 정보가 없으므로 넣어준다. 처음 발견된 이웃 노드일 경우, 당연히 방문된 적이 없으므로 visited = false로 설정한다. (expand 및 merge 단계)

```
INSERT INTO priority_q ( id, alt_dist, real_dist, visited )
( SELECT pq.id as id,
          ( pq.real_dist + dt.weight + lmk.weight ) as
            alt_dist,
          ( pq.real_dist + dt.weight ) as real_dist,
            false as visited
FROM priority_q as pq, data as dt, landmark as lmk
WHERE pq.id = 'frontier'
AND frontier's neighbor node in dt
AND neighbor node.visited = false )
```

A. V. Goldberg et al.은 Landmark 전처리 시에 Landmark로부터 다른 노드들까지의 최단 거리와 최단 경로를 한꺼번에 저장하였는데[4], 본 논문에서는 전처리의 부담 및 전처리된 데이터의 크기를 줄이기 위하여 Landmark로부터 다른 노드들까지의 거리만을 계산하였다.

4. 실 험

4.1 Data sets & Environment

본 실험에서는 엣지의 가중치 존재 여부가 다른 2개의 실제 데이터와 인위적으로 생성한 랜덤 데이터들에 대하여 실험을 진행하였다. 사용한 랜덤 데이터들은 다수의 노드들과 해당 노드들 사이의 엣지가 무작위로 생성된 랜덤 그래프이며 실제 데이터는 미국 뉴욕시의 교통 그래프 데이터[14]와 페이스북 사용자간의 친구 관계 데이터[15]이다. 각 랜덤 데이터에서 엣지들이 가중치가 있는 경우는 1부터 20 사이의 임의의 값을 배정하였고, 가중치가 없는 경우는 모두 1을 배정하였다. 본 데이터들을 대상으로 다수의 임의 노드 2개(출발점 및 도착점)를 선택하여 In-database FEM 알고리즘 및 ALT 알고리즘의 성능을 비교하였다. FEM 알고리즘의 입력값은 directed graph이고, ALT 알고리즘의 입력값은 directed graph와 1개의 Landmark로부터 다른 노드들간의 거리가 전처리로 계산된 값이다. 알고리즘의 진행 방향은 출발점에서 시작하여 도착점을 탐색하는 방법과 역방향으로 도착점으로부터 출발점을 찾아내는 2가지 방법을 모두 실험하였으며, 평균치를 기록하였다. Landmark는 그래프에서 임의의 노드를 선택하는 방법과 이웃이 가장 많은 노드를 선택하는 방법, 도착점 바로 앞의 노드를 선택하는 방법 등 총 3가지로 실험을 진행하였다. 임의로 선택된 Landmark가 99% 이상의 경우에서 동등하거나 더 나은 성능을 보였기에 본 결과에서는 임의로 선택된 Landmark의 실험 결과만 표시하였다. 각 실험

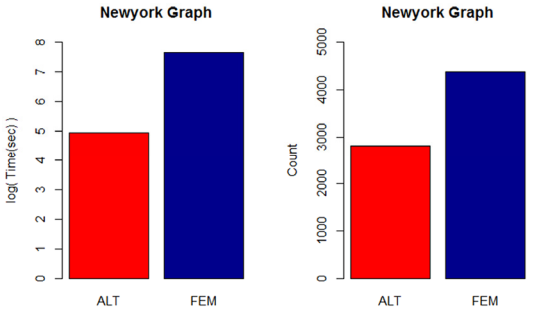

그림 3 뉴욕 그래프의 시간(좌), 탐색횟수(우) 비교

Fig. 3 Time(left), Search space(right) comparison on NewYork City graph

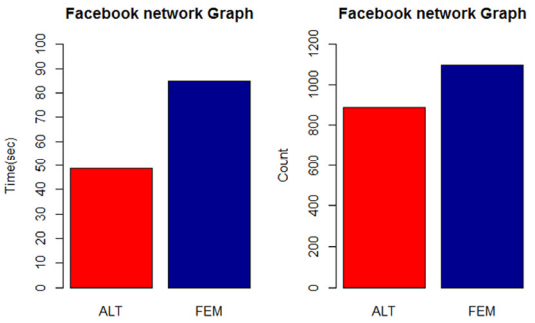

그림 4 페이스북 친구 그래프의 시간(좌), 탐색횟수(우) 비교

Fig. 4 Time(left), Search space(right) comparison on Facebook friends graph

을 20번씩 반복하였으며, 최단 경로 탐색에 걸리는 평균 시간을 기록하였다.

실험 환경으로는 Ubuntu 16.04.2 LTS 운영체제, Intel ® Xeon ® CPU E5-2620 v4 @ 2.10Ghz CPU, 16GB Samsung DDR4 RAM 5개를 사용하였고, 대상 데이터베이스로는 MonetDB Database ver.1.1을 사용하였다. MonetDB는 row 기반이 아니라, OLAP이나 의사 결정을 위한 데이터 웨어하우스에 자주 사용되는 column 기반 관계형 데이터베이스이다. 이는 큰 크기의 데이터에 대해 데이터 저장, 쿼리 프로세싱 알고리즘, 쿼리 실행 모델 등의 관점에서 보다 최적화된 기법을 제공한다[16,17]. MonetDB와 파이썬의 연결을 위하여 pymonetdb ver. 1.1.1을 사용하였으며, Landmark 전처리를 위하여 파이썬 networkx ver. 1.11을 사용하였다.

4.2 NewYork City Graph

본 데이터는 9th DIMACS Implementation Challenge and US Census Bureau에서 최단 경로 탐색을 위하여 사용된 뉴욕 도시 그래프 데이터의 일부분을 추출한 데이터이다. 총 264,346개의 노드(지점)과 366,923 개의 엣지(도로)로 구성되어 있으며, 각 엣지의 가중치는 출발 노드와 도착 노드 사이의 거리이다[14]. 총 20쌍의 임의의 출발 노드 및 도착 노드를 선택하였으며, 결과는 그림 3과 같다. ALT알고리즘이 FEM 프레임워크에 비해 최대 38.9배, 최소 1.55배, 평균적으로 약 15.2배 이상 빨랐으며, 탐색 영역은 최대 88%, 평균적으로 36% 감소하였다. 전처리 과정에서 약 10초 정도의 시간이 소요되었는데, 해당 전처리 시간을 고려하더라도 ALT 알고리즘이 FEM 프레임워크에 비해 약 14.2배 빨랐다. 전처리 과정은 탐색을 여러 번 하더라도 초기에 한번만 진행하면 되기 때문에 다양한 노드 사이의 최단거리를 반복해서 탐색할 시, 실제 오버헤드는 훨씬 줄어든다.

4.3 Facebook Networks

본 데이터는 페이스북의 사용자들간 친구 네트워크를 나타내는 실제 그래프 데이터이다. 총 4,039개의 노드(사용자)와 88,234개의 엣지(친구 관계)로 구성되어 있으며, 사용자 간 엣지들의 가중치는 없으므로[15] 모두 1로 할당하였다. 뉴욕 데이터와 마찬가지로 임의의 20쌍의 출발 노드 및 도착 노드를 선택하였다. 최단 경로 탐색을 위하여 FEM 및 ALT 알고리즘을 적용한 결과는 그림 4와 같다. 휴리스틱한 방법을 사용한 ALT가 단순 Dijkstra를 구현한 FEM보다 이웃노드 탐색영역은 약 20% 더 적고 탐색시간은 약 74% 빠르다.

A. V. Goldberg et al.은 ALT 알고리즘을 가중치가 있는 그래프를 위하여 고안하였다고 언급하였으나[4], 가중치가 없는 실제데이터에서도 평균적으로 FEM 보다 나은 성능을 보여준다. 다만 이처럼 가중치가 없는 그래프의 경우 ALT의 탐색영역이 FEM의 탐색영역과 동일 혹은 매우 근접하게 되는 경우가 발생할 수 있으며, 이 경우 ALT보다 FEM이 더 빠른 경우도 있었다. 이와 같은 현상은 4.4의 가중치가 없는 랜덤 그래프에서도 동일하게 발생하였는데, 가중치가 있는 4.2의 뉴욕 그래프나 4.4의 가중치가 있는 랜덤 그래프의 경우 해당 현상이 발생하지 않았다. 가중치가 없는 그래프에서도 ALT 알고리즘의 적용이 가능하나, 항상 기존의 FEM보다 더 나은 성능을 보인다는 보장이 없기에 A. V. Goldberg et al.은 자신들의 알고리즘이 가중치가 있는 그래프를 대상으로 제안되었다고 말한다.

4.4 Random Data set

1,000개의 노드들과 랜덤하게 연결된 10,000개의 엣지들에 대해 가중치가 있는 경우와 없는 경우, 5,000개의 노드들과 랜덤하게 연결된 50,000 개의 엣지들에 대해 가중치가 있는 경우와 없는 경우, 총 4가지 경우에 대하여 실험하였다. 각 경우에 대해 총 20개의 데이터셋을 임의로 생성하였고, 최단거리 탐색을 위하여 각 데이터셋마다 10쌍 이상의 출발 노드 및 도착 노드들의 최단

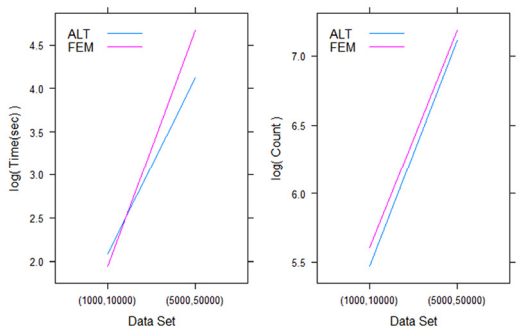

그림 5 가중치 있는 그래프 시간(좌), 탐색횟수(우) 비교
Fig. 5 Time(left), Search space(right) comparison on weighted graphs

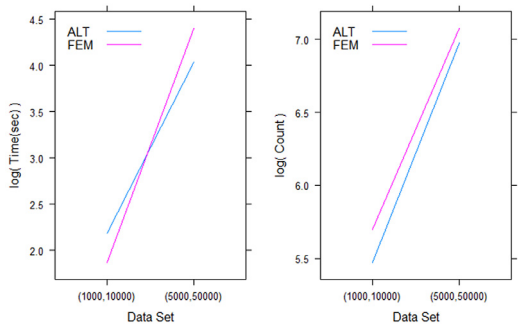

그림 6 가중치 없는 그래프 시간(좌), 탐색횟수(우) 비교
Fig. 6 Time(left), Search space(right) comparison on unweighted graphs

경로 탐색을 10회 반복 측정하였다.

그림 5는 엣지에 가중치가 있는 데이터들에 대한 실험을, 그림 6은 엣지에 가중치가 없는 데이터들에 대한 실험의 평균 연산 시간 및 탐색횟수를 보여준다. 가중치가 있는 경우, ALT는 1,000개 노드와 5,000개 노드 두 가지 경우 모두에 대해 탐색 영역이 FEM보다 약 10% 적다. 허나 1,000개 노드의 경우 ALT가 FEM보다 탐색 영역이 적음에도 불구하고, 약 27% 정도 더 많은 연산 시간이 소요 되는데, 이는 ALT 거리를 계산하는 오버헤드가 ALT로 인한 시간 감축효과보다 더 크기 때문에 발생한다. 이러한 오버헤드는 데이터 크기가 비교적 큰 5,000개 노드 데이터에서는 사라진다. ALT의 연산시간이 FEM보다 최대 3.5배, 평균적으로 73% 더 빠르다. 4.2의 실제 데이터에서는 ALT가 훨씬 압도적인 성능을 보인다.

그림 6만 보면 가중치가 없는 경우에도 ALT가 FEM보다 더 나은 성능을 보이는 것처럼 보인다. 그러나 4.3에서 언급하였듯이 가중치가 없는 그래프의 경우, ALT 알고리즘이 항상 FEM보다 더 나은 성능을 보인다고 말할 수는 없다. 랜덤 데이터 20개의 데이터 셋을 대상으로 한 230번의 최단 경로 탐색에서 가중치가 있는 랜덤 데이터의 경우, ALT가 FEM과 동일한 영역을 탐색한 경우는 단 1건이었지만, 가중치가 없는 그래프의 경우 ALT가 FEM과 동일한 영역을 탐색한 경우가 무려 21건이나 되었고 대부분의 경우 ALT가 FEM보다 탐색 시간이 더 소요되었다. 이는 ALT로 인한 시간 감축효과보다 ALT 거리를 계산하는 오버헤드가 더 크기 때문에 발생한다.

5. 결 론

본 논문에서는 보다 큰 그래프 데이터에 대한 최단 경로 탐색을 위하여 A* 알고리즘에 Landmark 및 Triangle-Inequality를 접목한 ALT 알고리즘을 관계형 데이터베이스 내부에서 설계 및 구현하는 방안을 제안하였다. 출발 노드에서부터 특정 노드까지의 거리와 전처리된 Landmark 정보를 사용하여 특정 노드에서부터 도착 노드까지의 lower-bound 거리를 구할 수 있으며, 이를 휴리스틱한 추정값으로 계산한다. 본 논문의 실험에서는 가중치가 있는 그래프에 대하여 In-RDBMS 환경에서 Dijkstra 알고리즘을 구현한 FEM 프레임워크보다 In-RDBMS ALT 알고리즘이 연산 속도는 월등히 빠르며, 탐색 영역 또한 확연히 줄어들었음을 확인할 수 있었다. 가중치가 없는 그래프에 대해서도ALT알고리즘이 FEM 프레임워크에 비해 평균적으로 더 적은 영역만을 탐색한다.

Future work로는 본 논문에서 구현한 ALT 알고리즘을 보다 다양하게 변형시켜보고자 한다. ALT 알고리즘은 Landmark를 어떻게 선택하느냐에 따라 성능 차이가 존재한다. 본 논문에서는 하나의 Landmark만을 사용하였는데, 다수의 Landmark를 설정하거나 Landmark 선택 방법을 보다 다양화 하는 등의 변형을 시도해보고자 한다. 또한 보다 빠른 속도를 위하여 ALT 알고리즘을 출발점과 도착점에서 동시에 진행하는 분산 혹은 병렬처리 기법도 시도해보고자 한다.

References

[1] A. Madkour, W. G. Aref, F. U. Rehman, M. A. Rahman, and S. Basalamah, "A Survey of Shortest-Path Algorithms," *arXiv preprint arXiv:1705.02044*, 2017.

[2] J. Gao, J. Zhou, J. X. Yu, and T. Wang, "Shortest Path Computing in Relational DBMSs," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26, No. 4, pp. 997-1011, Apr. 2014.

[3] H. Reddy, "PATH FINDING-Dijkstra's and A* Algorithm's," *International Journal in IT and*

Engineering, pp. 1-15, 2013.
[4] A. V. Goldberg, and C. Harrelson, "Computing the Shortest Path: A Search Meets Graph Theory," *Proc. of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 156-165, Jan. 2005.
[5] A. V. Goldberg, and R. F. F. Werneck, "Computing Point-to-Point Shortest Paths from External Memory," *Proc. of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics*, pp. 26-40, Jan. 2005.
[6] "Neo4j," [Online]. Available: http://neo4j.org/, 2014.
[7] J. J. Miller, "Graph Database Applications and Concepts with Neo4j," *Proc. of the Southern Association for Information Systems Conference*, Vol. 2324, p. 36, Mar. 2013.
[8] X. Yan, and J. Han. "Gspan: Graph-based Substructure Pattern Mining," *Proc. 2002 IEEE International Conference on Data Mining*, pp. 721-724, Dec. 2002.
[9] C. Wang, W. Wang, J. Pei, Y. Zhu, and B. shi, "Scalable Mining of Large Disk-based Graph Databases," *Proc. of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM*, pp. 316-325, Aug. 2004.
[10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective," *Proc. of the 48th Annual Southeast Regional Conference*, pp. 42, 2010.
[11] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische mathematic*, Vol. 1, No. 1, pp. 269-271, 1959.
[12] J. Gao, R. Jin, J. Zhou, J. X. Yu, X. Jiang, and T. Wang, "Relational Approach for Shortest Path Discovery over Large Graphs," *Proc. of the Very Large Database Endowment*, Vol. 5, No. 4, pp. 358-369, Dec. 2011.
[13] M. Yan, "Dijkstra's Algorithm," *Massachusetts Institute of Technology. Regexstr*, 2014.
[14] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, 9th DIMACS Implementation Challenge - Shortest Paths, [Online] Available : http://www.dis.uniroma 1. it/~challe nge9 (2018.Jan).
[15] J. Leskovec, and J. J. Mcauley, "Learning to Discover Social Circles in Ego Networks," *Proc. of the 25th International Conference on Neural Information Processing Systems*, Vol. 1, pp. 539-547, Dec. 2012.
[16] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the Memory Wall in MonetDB," *Communications of the ACM*, Vol. 51, No. 12, pp. 77-85, Dec. 2008.
[17] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, "MonetDB: Two Decades of Research in Column-oriented Database Architectures," *IEEE Data Engineering*, Vol. 35, No. 1, pp. 40-45, 2012.

이 용 현
정보과학회 컴퓨팅의 실제 논문지
제 25 권 제 1 호 참조

임 동 혁
2003년 고려대학교 컴퓨터교육과 학사
2005년 서울대학교 컴퓨터공학부 석사
2011년 서울대학교 컴퓨터공학부 박사
2011년~2012년 서울대학교 치의학연구소 선임연구원. 2013년~현재 호서대학교 컴퓨터공학과 조교수. 관심분야는 데이터베이스, 빅데이터, 그래프 처리 및 분석

구 해 모
정보과학회 컴퓨팅의 실제 논문지
제 25 권 제 1 호 참조

김 형 주
정보과학회 컴퓨팅의 실제 논문지
제 25 권 제 1 호 참조