https://doi.org/10.5392/JKCA.2018.18.02.438

그래프 이력 데이터 접근을 위한 효과적인 저장 관리 기법

Efficient Storage Management Scheme for Graph Historical Retrieval

김기훈*, 김이나**, 최도진*, 김민수*, 복경수*, 유재수*
충북대학교 정보통신공학과*, 충북대학교 빅데이터협동과정**

Gihoon Kim(gihoonkim@cbnu.ac.kr)*, Ina Kim(inakim@cbnu.ac.kr)**,
Dojin Choi(mycdj91@cbjnu.ac.kr)*, Minsoo Kim(mskim88@cbnu.ac.kr)*,
Kyoungsoo Bok(ksbok@cbnu.ac.kr)*, Jaesoo Yoo(yjs@cbnu.ac.kr)*

요약

최근 소셜 네트워크, 인용 네트워크 등 여러 분야에서 다양한 그래프 데이터가 활용되고 있다. 시간에 따라 그래프가 동적으로 변화함에 따라 변경 내용 추적 및 특정 시점 그래프 검색을 위해 그래프 이력 데이터를 관리하는 것이 필요하다. 대부분의 이력 데이터는 시간에 따라 부분적인 변화가 발생하기 때문에 시간 단위로 데이터를 저장할 경우 변경되지 않은 데이터가 중복 저장된다. 본 논문에서는 시간별 그래프의 중복 저장을 최소화하기 위한 그래프 이력 저장 관리 기법을 제안한다. 제안하는 기법은 그래프의 변화를 계속적으로 탐지하여 과거 그래프와 중복되는 서브 그래프를 하나의 중복 스냅샷에 저장한다. 중복 스냅샷에는 다수의 델타 스냅샷이 연결되어 각 시간에 따른 변화 데이터를 유지한다. 중복 스냅샷에 저장된 중복 데이터를 공통으로 관리하여 공간의 효율을 향상시킨다. 또한, 해당 시점의 그래프를 탐색하기 위해서 중복 스냅샷과 델타 스냅샷을 연결하였다. 제안하는 기법의 우수성을 보이기 위해 다양한 성능평가를 수행한다.

■ 중심어 : | 그래프 저장 | 이력 그래프 | 그래프 검색 | 소셜 데이터 | 동적 그래프 |

Abstract

Recently, various graph data have been utilized in various fields such as social networks and citation networks. As the graph changes dynamically over time, it is necessary to manage the graph historical data for tracking changes and retrieving point-in-time graphs. Most historical data changes partially according to time, so unchanged data is stored redundantly when data is stored in units of time. In this paper, we propose a graph history storage management method to minimize the redundant storage of time graphs. The proposed method continuously detects the change of the graph and stores the overlapping subgraph in intersection snapshot. Intersection snapshots are connected by a number of delta snapshots to maintain change data over time. It improves space efficiency by collectively managing overlapping data stored in intersection snapshots. We also linked intersection snapshots and delta snapshots to retrieval the graph at that point in time. Various performance evaluations are performed to show the superiority of the proposed scheme.

■ keyword : | Graph Storage | Historical Graph | Graph Retrieval | Social Data | Dynamic Graph |

* 본 연구는 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단(No. 2016R1A2B3007527), 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보 · 컴퓨팅기술개발사업(No. NRF-2017M3C4A7069432), 2014년도 산업통상자원부의 재원으로 한국에너지기술평가원(KETEP)의 에너지인력양성사업으로 지원받아 수행한 인력양성 성과입니다.(No. 20164030201330)

접수일자 : 2017년 11월 15일
수정일자 : 2017년 12월 08일
심사완료일 : 2017년 12월 13일
교신저자 : 유재수, e-mail : yjs@chungbuk.ac.kr

I. 서 론

객체들 사이에 관계 또는 상호 작용 표현을 위해 컴퓨터 분야에서 오랜 기간 동안 그래프 데이터에 관한 연구가 진행되고 있다[1-3]. 그래프는 객체 정보를 정점과 정점간의 연결성을 표현하는 간선으로 구성되어 있다. 각 정점과 간선은 정보가 포함될 수 있다. 예들들어, 도로망에서 하나의 정점은 교차로를 표현 할 수 있고, 정점과 정점의 연결인 간선은 교차로에서 다른 교차로까지 연결된 도로 정보를 포함할 수 있다. 도로망과 같이 급격한 변화가 일어나지 않는 그래프 데이터를 정적인 그래프 데이터라고 한다. 정적인 데이터는 변화가 존재하지 않기 때문에 새로운 데이터에 대해 고려하지 않는다.

최근 소셜 네트워크, 인용 네트워크, IoT 등의 분야에서는 새로운 정보 생성되고 변화되고 있다. 이와 함께 시간에 따라 그래프 데이터도 계속적으로 변화하는 동적 그래프를 생성한다. 대표적으로 facebook의 친구관계, twitter의 팔로우, 리트윗과 논문, 특허의 인용정보 등이 있다[4]. 대표적인 소셜 네트워크 서비스 중 하나인 facebook은 분당 약 330만 건의 데이터가 생성된다[5]. 그래프의 형태가 지속적으로 변화하는 동적 그래프에서 많은 양의 과거 이력 데이터가 발생한다. 과거 이력데이터는 과거 그래프에 대한 분석에 필요하다. 과거 그래프 데이터 접근이 필요한 분석의 예로“시간의 흐름에 따른 정보의 확산이 어떻게 되었는가?”, “2013년에 소셜 네트워크에서 가장 영향력이 높은 사용자는 누구인가?”, “2010년부터 2015년까지의 기간 중 네트워크 범위가 가장 작은 해는 언제인가?”등과 같은 상황이 있다[6][7]. 위와 같은 상황을 분석하기 위해 새롭게 생성된 이력 데이터와 기존의 이력 데이터로 효율적인 접근과 저장을 위한 기법이 필요하다.

그래프 이력 데이터를 관리하기 위해서는 각 시간의 과거 그래프를 저장하는 방법과 과거 시점의 데이터에 접근하는 방법이 필요하다[8]. 그래프 데이터 저장을 위한 기존의 일반적인 저장 구조는 각 시간에 따른 그래프의 상태를 개별적으로 저장한다. 시간에 따른 변화가 많은 동적 그래프 데이터의 경우 데이터의 변화가 부분적으로 발생한다[9]. 일부 데이터가 변화하는 상태를 모든 시간의 그래프로 유지하게 되면 많은 양의 중복 데이터가 저장되게 된다. 같은 정보의 중복 데이터 저장은 저장 공간 낭비의 원인이 된다. 예를 들어, 소셜 네트워크에서 한 사용자의 친구 관계 그래프는 전체 친구 관계의 변화가 아닌 일부 친구 관계의 변화만 존재한다.

그래프 이력 데이터를 효율적으로 저장하고 접근하기 위해 크게 복사 기법(Copy)과 이력 기법(Log)으로 두가지 유형의 연구가 진행되고 있다[10]. 모든 데이터를 복사하여 유지하는 방법과 데이터의 변화이력을 추적하여 해당 시점의 데이터를 생성하는 방법이다. 데이터를 복사하는 방법은 시간의 변화에 따른 데이터를 모두 보존하는 방법이다. 모든 데이터가 저장되어 있기 때문에 해당 시점에 대해 빠른 접근을 할 수 있다. 하지만 중복되는 데이터를 모두 저장하기 때문에 공간적 비용 소모가 크다. Konstantinos Semertzidis가 제시한 VERSION GRAPH[11]는 데이터 보존 기간을 설정하여 데이터를 지정된 기간 동안 보존하는 기법이다. 과거의 모든 이력 데이터를 저장하기 위해선 데이터 보존 기간이 무기한으로 늘어나고 모든 과거 데이터를 저장해야 한다. 변화 이력을 추적하는 방법은 모든 데이터를 저장하지 않고 변화 이력만을 저장하고 있기 때문에 높은 공간 효율을 보인다. 하지만 원하는 시점의 그래프 데이터를 확인하기 위해선 최초의 그래프에서 변화 이력을 순차적으로 접근하기 때문에 접근속도가 느린 단점이 있다. DeltaGraph[12]는 불필요한 데이터를 저장을 줄이기 위해서 데이터의 변화 이력을 활용하여 저장 공간의 효율을 높였다. 하지만 변화이력을 순차적으로 접근하기 때문에 그래프 데이터 접근이 비효율적이다. 기존의 기법은 두 접근 방법의 상호 반비례 관계로 저장과 접근 관점에서 각각 좋지 않은 결과를 보인다. 모든 상황을 모두 고려하여 보다 효율적인 이력 그래프 데이터의 저장과 접근을 위해서 두 방법을 모두 적절히 고려한 기법이 필요하다.

본 논문에서는 스냅샷을 활용한 효율적인 이력 그래프 저장 관리 기법을 제안한다. 동적 그래프의 중복 데이터 관리를 위해 중복 스냅샷에 저장된 중복데이터를

공통으로 관리하여 중복 저장을 감소 시켜 공간의 소모를 줄였다. 또한, 해당 시점의 그래프를 탐색하기 위해서 중복 스냅샷과 델타 스냅샷을 연결하였다. 제안하는 기법은 기존의 기법들의 공간 소모의 문제를 해결하기 위해 이력 데이터를 중복 저장하지 않기 때문에 공간적 효율을 높일 수 있다.

본 논문의 구성은 다음과 같다. 제 II장에서는 관련 연구를 설명한다. 제 III장에서는 제안하는 이력 그래프 관리 기법을 기술한다. 제 IV장에서는 기존 기법과의 비교를 통한 성능평가를 기술한다. 제 V장에서는 본 논문의 결론과 향후 연구 방향을 제시한다.

II. 관련 연구

최근 그래프 데이터를 저장하는 기법과 이력 데이터 접근을 위한 방법이 많이 연구 되었다. 기존의 대용량의 그래프 데이터를 저장하기 위한 확장성 좋은 구조와 그래프 데이터의 변화이력을 활용한 과거 그래프 접근 기법이 있다. STINGER는 대용량의 그래프 데이터를 위한 확장 가능한 그래프 저장 구조이다[13]. 연결 리스트 기반의 구성으로 연속적으로 확장하는 동적 그래프 데이터를 표현할 수 있다. 간선에 대한 빠른 접근을 위해 정점 테이블과 간선 테이블을 따로 관리하여 해당 정점에 연결된 간선을 블록 단위로 저장하는 구조이다. 간선 테이블이 따로 있기 때문에 원하는 간선의 정보에 직접적으로 접근할 수 있다. 하지만 그래프의 이력이 아닌 현재 그래프의 상태만을 표현할 수 있기 때문에 과거 시점에 대한 그래프 데이터 표현이 불가능하다.

DegAwareRHH는 동적 그래프 데이터를 분산하여 저장하기 위한 구조이다[14]. 대용량의 데이터의 효율적인 분산 저장을 위해서 robin-hood hashing 기법을 활용하여 데이터의 지역성을 유지한다. 정점을 Middle-high-degree와 Low-degree 테이블로 나누어 저장한다. Middle- high-degree 테이블의 경우 다수의 정점의 정보를 하나의 테이블에 저장하고 각각의 정점은 정점으로부터 출발하는 간선들의 집합을 포인터로 가리키는 구조이다. Low-degree 정점은 단순한 하나의 테이블로 정점과 정점으로부터 출발하는 간선들이 함께 저장되는 구조이다. Hash를 활용한 지역성 유지로 대용량의 그래프 데이터를 효율적으로 분산 저장할 수 있다.

동적 그래프 데이터 저장을 위해 확장성과 기존 구조의 저장 공간 낭비를 해결하기 위해 LLAMA[15]가 제안되었다. 다중 버전의 배열을 활용한 확장 가능한 그래프 저장구조이다. 하지만 LLAMA와 같은 구조는 그래프 분석을 위한 각 시간별 접근을 할 수 없고 현재 상태의 그래프 데이터만을 유지하며, 각 스냅 샷에 접근하기 위해서는 첫 번째 변화 이력부터 단계적으로 접근하기 때문에 많은 처리시간을 요구한다.

기존의 그래프 저장 구조는 데이터의 확장성을 고려한 구조이다. 과거 데이터에 대한 접근 방법이 고려되지 않아 현재의 그래프 데이터로의 접근만이 가능한 구조이다. DeltaGraph는 시간에 따라 변화한 그래프 데이터의 변화 이력을 저장하여 만든 그래프 데이터 관리 시스템이다[12]. 계층적 구조로 과거 이력을 관리한다. 말단 노드에 시간에 따른 그래프 데이터가 있고 상단에는 각 그래프 데이터에 접근하는 중간 노드로 구성되어 있다. 한 시점의 데이터에 접근하기 위해선 해당 시점까지의 최단 경로를 계산하여 말단 노드에 접근해야 하며, 말단 노드 사이의 그래프 상태에 대해선 이력을 추적하여 순차 접근을 한다. 과거 이력 검색을 위한 검색 기법의 경우 데이터의 접근을 위해서 모든 이력을 탐색하기 때문에 데이터 접근 속도가 느린 문제점이 있다.

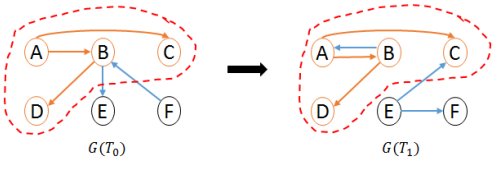
III. 제안하는 그래프 관리 기법

1. 전체 구조

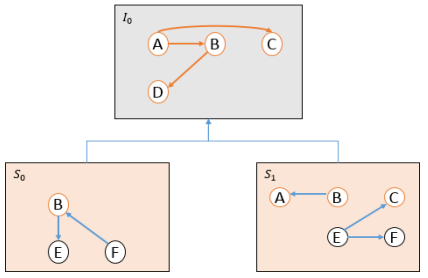
동적 그래프 데이터 분석을 위해서 이력 그래프 데이터의 저장과 접근이 필요하다. 동적 그래프는 중복되는 데이터가 많이 발생하기 때문에 중복저장으로 인한 공간이 낭비된다. 본 논문에서는 중복 데이터의 중복 저장을 최소화하기 위해 새로운 그래프 저장관리 기법을 제안한다. 중복되는 데이터를 최소화하여 과거 그래프 데이터에 접근을 위해 중복 스냅샷과 델타 스냅샷으로

데이터를 분할하여 관리한다. 중복 스냅샷은 각 시간의 그래프 데이터에서 사용자가 설정한 중복 임계치 이상의 중복 그래프 데이터로 생성된다. 중복된 그래프를 제외한 나머지 서브 그래프는 델타 스냅샷에 저장된다. 중복되는 데이터를 중복 스냅샷에서 공통으로 관리하기 때문에 저장 공간의 낭비를 감소시킨다. 하나의 스냅샷은 출발 정점과 도착 정점 테이블을 포함한다. 출발 정점은 도착 정점 테이블의 시작 위치를 나타내는 오프셋과 연결된 정점의 수로 두 정점이 연결된 간선을 표현한다. 중복 스냅샷과 델타 스냅샷의 접근으로 해당 시점의 그래프 데이터에 접근할 수 있다.

[그림 1]은 제안하는 이력 그래프 저장 관리 구조를 나타낸 것이다. 제안하는 기법의 전체 구조는 상단의 중복 스냅샷(intersection snapshot)과 하단의 각 시간별 변화 데이터를 갖는 델타 스냅샷(delta snapshot)으로 구성된다. 시간별 그래프의 중복되는 데이터를 중복 스냅샷에 공통적으로 저장한다. 델타 스냅샷은 중복된 데이터에서 변화된 데이터만을 저장하여 공간 효율을 높인다. 스냅샷 접근을 위해서 중복 스냅샷과 연결되어 있는 델타 스냅샷의 시간정보를 유지한다. [그림 1]의 (a)에서 $G(T_0)$와 $G(T_1)$는 중복되는 서브 그래프가 존재한다. 중복되는 서브 그래프 ($G(A,B,C,D)$)는 중복 스냅샷 I_0에 저장된다. T_0 시점의 중복되는 부분을 제외한 서브 그래프 $G(B,E,F)$는 델타 스냅샷 S_0에 저장되고 T_1 시점의 서브 그래프 $G(A,B)$와 $G(C,E,F)$는 S_1에 저장된다. 중복 스냅샷 I_0와 델타 스냅샷 S_0, S_1을 연결하여 해당 시점의 그래프 $G(T_0)$와 $G(T_1)$을 표현할 수 있다.



(a) 동적 그래프의 데이터 중복



(b) 이력 데이터 저장 구조

그림 1. 제안하는 이력 그래프 저장 관리 구조

2. 스냅샷 생성과정

제안하는 기법의 전체 구조는 스냅샷으로 구성되어 있다. 각각의 스냅샷은 중복 스냅샷과 델타 스냅샷으로 분류된다. 스냅샷의 생성은 최초의 그래프 데이터로부터 시간에 따라 순차적으로 중복되는 서브 그래프는 검출한다. 검출된 서브 그래프의 중복률이 임계값보다 큰 시간까지의 그래프를 하나의 중복 스냅샷으로 생성한다. 중복 스냅샷으로 생성한 시간까지의 그래프 데이터에서 중복 검출된 서브 그래프에 포함되지 않는 각 시간의 서브 그래프를 델타 스냅샷으로 생성하여 중복 스냅샷과 연결한다. 위와 같은 과정을 반복하여 제안하는 기법의 전체 구조를 구축한다.

전체 스냅샷의 생성 과정을 [그림 2]와 같다. 순서도에서 k, i, n은 중복 그래프와 델타 그래프의 생성을 위한 카운트 값이다. 스냅샷의 생성은 최초의 그래프 $G(T_n)$이 중복 그래프로 선정된다. 선정된 그래프 $G(I_k)$는 다음 시점의 그래프인 $G(T_{n+1})$과 중복률을 비교하여 중복률이 중복 임계치인 θ보다 클 경우 두 그래프의 중복 서브 그래프를 $G(I_k)$로 갱신한다. 중복 서브 그래프의 갱신한 뒤, 다음 시점의 그래프 $G(T_{n+2})$와 반복적으로 비교하여 새로운 중복 서브 그래프를 갱신한다. 만약 중복 그래프와 새로운 그래프의 중복률이 θ보다 작을 경우 해당 $G(I_k)$를 중복 스냅샷으로 생성하고 $G(T_i)$부터 $G(T_n)$까지의 그래프를 델타 스냅샷으로 생성하여 중복 스냅샷과 연결한다. 위와 같은 순서를 반복적으로 수행하여 제안하는 기법의 전체 구조를 구성한다.

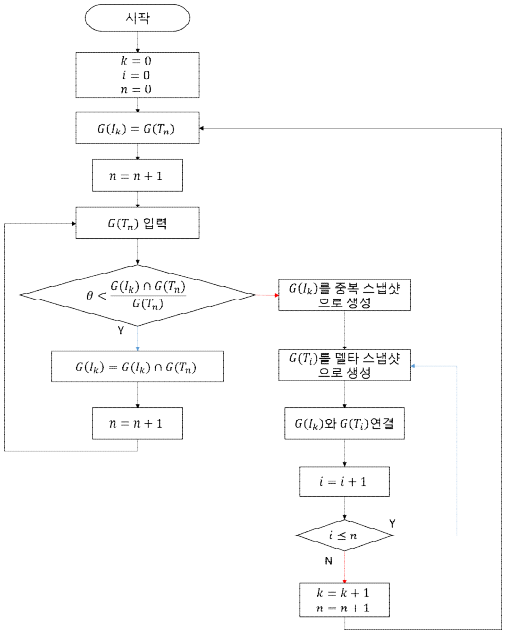


그림 2. 전체 스냅샷 생성 과정

3. 중복 스냅샷

동적 그래프는 데이터의 전체가 변경되지 않고 시간에 따라 부분적인 변화만 발생한다. 변경되지 않은 나머지 데이터는 중복되게 된다. 이러한 중복되는 데이터를 개별 관리한다면 저장 공간의 낭비가 발생한다. 저장 공간을 낭비시키는 중복 데이터를 공통으로 관리하여 공간 효율을 높이기 위해서 중복 스냅샷이 필요하다. 제안하는 기법의 중복 스냅샷은 기존 기법에서 사용하는 스냅샷과 다르게 범위 시간의 중복 데이터만을 저장하게 된다. 생성된 중복 스냅샷은 중복되는 데이터를 중복저장하지 않기 때문에 저장 공간의 낭비를 최소화 할 수 있다. 중복 스냅샷은 각 시간별 그래프에서 중복되는 서브그래프의 정점과 간선의 정보와 연결되어 있는 델타 스냅샷의 시간 정보를 포함한다.

중복 스냅샷은 각 시간별 그래프 데이터의 중복되는 데이터는 저장한다. 각 그래프 데이터에서 그림 3과 같이 시간의 흐름에 따라 변화된 그래프의 중복된 정점과 간선을 검출한다. 중복 스냅샷의 생성 방법은 다음과 같다. 먼저, 시간별 그래프의 중복검사를 순차적으로 진행한다. 중복 검사를 진행한 그래프의 중복이 임계값 이하가 되면 그 이전의 중복이 임계값을 초과하는 n개의 그래프에 대한 중복 스냅샷을 생성한다. 중복 스냅샷이 생성되면 각 시간별 그래프의 변화를 델타 스냅샷으로 만들어 연결한다. 그 후의 그래프 데이터 또한 같은 작업을 반복한다. 하나의 중복 스냅샷은 [표 1]과 같이 다수의 델타 스냅샷과 연결되어 있다.

[그림 3]과 같이 시간의 흐름에 따라 (a), (b), (c)순서로 변경된 그래프가 존재한다고 가정하자. 그래프 (a)와 (b)는 간선 (A,B), (A,C), (B,D)가 중복되고 (B,E), (F,B)가 변경되었다. (b)에서 (c)로 그래프가 변경되는 상황에서도 (a)와 (b)에서 중복된 그래프 데이터가 그대로 유지되는 것을 확인 할 수 있다. [표 1]에서 확인한다면 중복되는 간선 (A,B), (A,C), (B,D)가 I_0에 표현되고, 변경된 간선 (B,E), (F,B)가 (S_0, T_0)로 테이블에 유지된다. 이렇게 지속적으로 중복이 유지되는 상황에서 검출된 중복 데이터를 중복 스냅샷으로 생성한다.

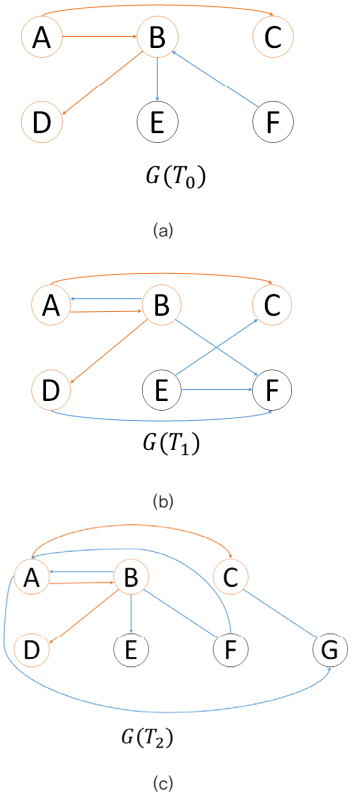


그림 3. 중복 스냅샷 생성

표 1. 중복 스냅샷과 델타스냅샷

중복 스냅샷	델타 스냅샷
I_0	(S_0, T_0), (S_1, T_1), (S_2, T_2)
I_1	(S_3, T_3), (S_4, T_4)
...	...
I_k	(S_{n-1}, T_{n-1}), (S_n, T_n)

4. 델타 스냅샷

중복 스냅샷과의 연결을 통해 하나의 전체 그래프의 표현을 위해서 델타 스냅샷을 생성한다. 델타 스냅샷은 각 시간별 데이터에서 중복 스냅샷을 제외한 나머지 서브 그래프 데이터를 저장한다. 중복 스냅샷 생성에 사용된 그래프의 수만큼의 델타 스냅샷이 하나의 중복 스냅샷에 연결되어 있다. 델타 스냅샷의 생성은 중복 스냅샷이 생성된 후, 해당 중복 스냅샷의 중복 검사 후 검출된 중복된 서브그래프를 제외한 나머지 서브 그래프를 저장한다. 모든 중복 데이터를 저장하지 않고 따로 저장하여 관리할 수 있기 때문에 공간의 효율을 높일 수 있다.

저장된 서브 그래프의 출발 정점 정보가 중복 스냅샷의 출발 정점과 같은 경우, 테이블 오프셋 정보로 중복 스냅샷과 연결된다. 연결된 델타 스냅샷은 중복 스냅샷과 중복되지 않는 각 그래프의 추가 정보를 포함한다. 내부 구조는 중복 스냅샷과 비슷하다. 출발 정점과 도착정점 테이블, 시간 정보를 갖는다. 간선이 출발하는 모든 정점의 데이터를 출발정점 테이블이 유지한다. 도착정점 테이블에는 추가된 간선의 도착 정점 정보를 갖는다. 중복 스냅샷과 중복되는 간선의 도착정점은 중복 스냅샷의 도착정점 테이블의 오프셋과 연결된 정점의 수로 중복되는 그래프 데이터의 위치정보를 갖는다.

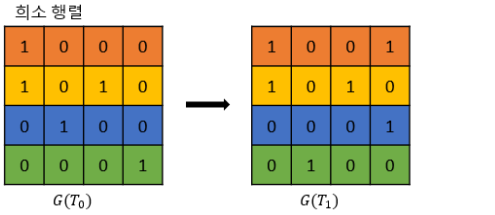
[그림 3]에서 중복된 간선 (A,B), (A,C), (B,D)를 제외한 나머지 부분이 델타 스냅샷으로 생성된다. (a)에서 간선 (B,E)와 (F,B)가 하나의 서브 그래프로서 델타 스냅샷으로 생성되고, (b)에서는 간선 (B,A),(B,F), (D,F),(E,C),(E,F)가 델타 스냅샷으로 생성된다. 최종적으로 시간 T_0의 그래프 $G(T_0)$의 델타 스냅샷은 간선 (B,E),(F,B), T_1에서의 그래프 $G(T_1)$은 (B,A),(B,F), (D,F),(E,C),(E,F), T_2에서의 그래프 $G(T_2)$는 (A,G), (B,A),(B,E),(C,F),(F,A)가 델타 스냅샷으로 생성된다.

5. 그래프 표현

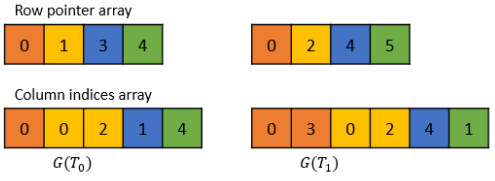
그래프 데이터는 일반적으로 행렬로 표현된다. 실제 그래프 데이터를 행렬로 표현하게 희소행렬로 표현되게 되고, 많은 양의 불필요한 공간이 필요하다. 희소 행렬은 실제 의미 있는 데이터를 저장하기 위해서 [그림 4]의 희소행렬과 같이 0으로 표현되는 무의미한 공간이 존재하게 된다. 그래프 데이터 표현에서 불필요한 정보량을 줄이기 위해서 제안하는 기법은 CSR(Compressed Sparse Row) 표현 기법은 수정하여 사용한다. 제안하는 기법에서는 중복 스냅샷과 델타 스냅샷의 연결을 위해서 CSR 기법을 수정하였다.

일반적으로 그래프 데이터는 정점과 정점을 연결하는 간선의 유무를 희소 행렬로 표현한다. 하지만 희소행렬로 그래프 데이터를 표현하게 되면 모든 정점의 수를 포함할 수 있는 크기의 행렬이 필요하다. 또한, 간선이 존재하지 않는 대부분의 행렬 값이 0으로 표현된다. 그래서 그래프 데이터의 희소행렬 표현을 효율적으로 저장하기 위해서 CSR 형태의 표현 방법을 사용한다 [16]. CSR 표현은 간선이 존재하지 않는 불필요한 행렬을 표현하지 않는 방법으로 행의 순서대로 실제 존재하는 데이터만을 연결하여 표현하는 방법이다. 제안하는 기법에서는 스냅샷 간의 연결을 위해 CSR기법을 수정하였다. [그림 4]의 (a)는 동적 그래프를 희소행렬로 표현한 것이다. 대부분의 행렬이 0으로 표시되어 불필요한 데이터를 가지고 있는 것을 확인할 수 있다. 그림 (b)는 그림 (a)의 시간 T_0, T_1의 희소행렬을 각각의 CSR 표현 기법으로 나타낸 것이다. Row pointer array에서 각각의 열을 표현하고 Column indices array의 위치를 가리킨다. Column indices array에서는 실제 데이터가 존재하는 행의 위치정보를 갖고 희소행렬을 표현한다. 이렇게 불필요한 공간을 표현하지 않음으로써 공간 효율을 높일 수 있다. 제안하는 논문에서는 CSR 표현 방법을 수정하여 각각의 스냅샷 내부에서 그래프 데이터를 표현하였다. 하지만 기존의 CSR표현 기법은 동

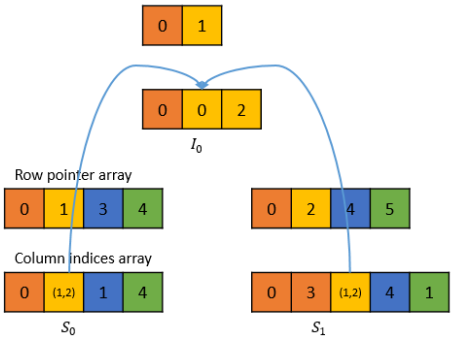
적인 환경에서의 중복 데이터를 모두 표현하는 문제가 있다. 제안하는 기법에서는 동적인 환경에서 중복되는 그래프 데이터를 정보량을 줄이고 계층구조의 스냅샷으로 연결하기 위해서 CSR 기법을 변경하였다. [그림 4]의 (c)가 변경된 CSR기법으로 그림 (a)의 희소행렬을 표현한 것이다. 중복되는 데이터를 I_0로 표현하고 S_0, S_1에 중복 데이터를 제외한 그래프 데이터를 표현하였다. I_0와 S_0,S_1의 연결을 위하여 column indices array를 수정하였다. 중복되는 데이터는 하단의 S_0, S_1의 내부에 I_0의 column indices array의 오프셋을 저장하여 연결하였다.



(a) 동적인 그래프의 희소행렬 표현



(b) CSR 표현

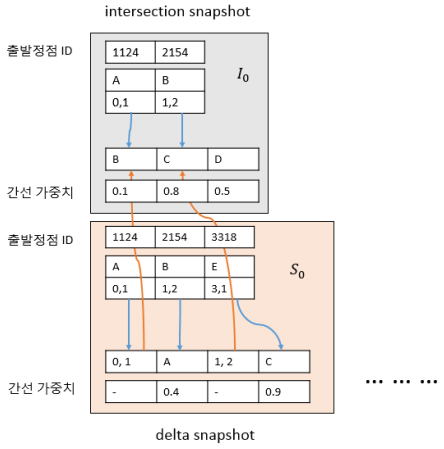


(c) 변경된 CSR 표현

그림 4. Compressed Sparse Row

	A	B	C	D	E
A	0	0.1	0	0	0
B	0.4	0	0.8	0.5	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0.9	0	0

(a) 희소 행렬



(b) 수정한 CSR 표현

그림 5. 정점과 간선의 속성 정보 표현

제안하는 기법은 스냅샷의 내부 구조에 정점과 간선의 정보를 표현할 수 있다. 각각의 정점의 속성 정보과 간선의 정보를 [그림 5]와 같이 표현 가능하다. [그림 5]의 (a)는 해당 그래프의 희소행렬이다. 그래프 데이터 표현을 위해 많은 양의 불필요한 공간이 소비되는 것을 확인할 수 있다. 그림 (b)는 (a)를 수정한 CSR 기법으로 표현한 것이다. 그림 (b)에서 희소행렬의 각 정점의 ID와 정점에서 출발하는 간선들의 가중치 정보를 하나의 중복 스냅샷과 델타 스냅샷으로 표현한 것이다. 정점 A의 출발 정점 ID인 1124를 key값으로 활용하여 데이터베이스에 실제 데이터를 찾아 갈 수 있고, 정점과

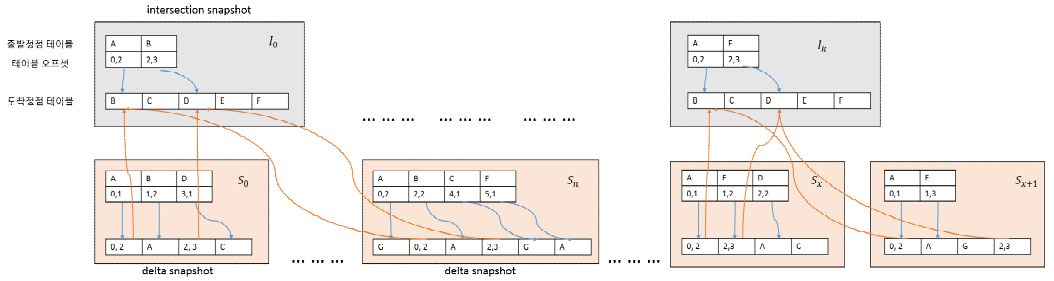


그림 6. 이력 그래프 탐색

연결된 간선(A,B)의 가중치인 0.1 또한 더욱 다양한 데이터를 넣기 원할 경우 key-value형 저장구조를 활용하여 다양한 크기의 정보를 저장할 수 있다.

6. 그래프 탐색

그래프 탐색은 해당하는 중복 스냅샷에 접근하여 중복 그래프를 읽고 하단의 델타 스냅샷을 읽어 전체 그래프를 읽게 된다. k번째 중복 스냅샷을 I_k, n번째 델타 스냅샷을 S_n로 표현했을 때 시간 T_n의 그래프는 식 (1)과 같다.

$$G(T_n) = I_k + S_n \quad (1)$$

[그림 6]의 전체 구조를 통해 $G(T_0)$를 접근한 예시이다. 먼저 [표 1]과 같은 테이블에서 T_0에 해당하는 S_0가 중복 스냅샷 I_0와의 연결을 확인한다. 해당 시간의 델타 스냅샷과 연결된 I_0의 접근이 완료되면 델타 스냅샷 S_0에 접근하여 중복 스냅샷과 마찬가지로 각 정점에 접근한다. 여기서 델타 스냅샷의 출발정점 A와 연결된 간선 중 새롭게 생성된 간선의 정보가 없으므로 I_0의 출발정점 테이블에 정점 A와 연결된 간선의 정보를 표현하는 I_0의 도착정점 테이블의 오프셋(0, 2)가 저장되어 중복된 데이터를 읽는다. I_0는 출발 정점 테이블의 A와 연결된 도착 정점 테이블 B와 C를 가리키는 오프셋 (0, 2)로 두 정점의 연결인 간선을 표현한다. 여기서 오프셋 0은 도착정점 테이블의 위치이고, 2는 출발정점 A와 연결된 도착정점의 개수이다. 델타 스냅샷의 다음 출발정점인 B의 경우 새로운 간선 (B,F)가 생성되었다. 그래서 정점 B가 가리키는 도착정점 테이블에는 새로운 연결 정보를 나타내는 정점 F를 표현하고 나머지 중복되는 정보는 I_0의 도착정점 테이블 오프셋으로 표시하여 접근한다. 최종적으로 위와 같은 절차를 통해 델타 스냅샷의 모든 정보를 읽어 해당 시간에 대한 이력 그래프 질의에 응답한다.

Ⅳ. 실험 평가

1. 실험 환경

실험 평가 대상은 가장 보편적인 이력 데이터 관리 기법인 interval-tree와 최신의 그래프 데이터 이력 검색 기법인 DeltaGraph로 수행하였다. interval-tree는 시간에 대한 색인 구조를 구축하여 각각의 이력 데이터에 접근하는 방법이고, DeltaGraph의 경우 과거 변화 이력을 탐색하여 해당 시간에 최종의 그래프 형태를 완성하는 이력 위주의 이력 그래프 탐색구조이다. 실험 환경은 [표 2]와 같이 Intel i5-4440 CPU, 8GB 메모리의 환경에서 진행하였다. 데이터는 CAIDA에서 제공하는 2004년부터 2007년까지의 인터넷 서비스 제공 변화를 표현한 그래프 데이터를 사용하였다. 각 시간별 그래프는 약 2만 5천개의 정점과 약 10만개의 간선으로 구성되어있다. 제안하는 기법의 공간 효율성과 접근속도를 평가하기 위해서 메모리 사용량과 질의응답시간을 측정하였다. 최적의 공간 효율을 확인하기 위해 제안하는 기법의 중복 임계값에 따른 자체적인 평가와 이력 검색을 위한 기존의 기법과 비교 평가하였다.

표 2. 실험 환경

구분	내용
CPU	Intel i5-4440 CPU
Memory	8GB
사용 데이터	CAIDA AS relationships data

2. 메모리 사용량 비교

[그림 7]은 제안하는 기법의 메모리 사용량을 비교하였다. 중복 스냅샷 생성에 각각 0.5, 0.6, 0.7의 임계값을 두고 실험 하였다. 임계값은 중복 스냅샷을 생성할 때 시간별 그래프의 중복률이다. 실험의 결과로 0.6의 임계값으로 중복 스냅샷을 생성하였을 경우 다른 임계값과 비교하여 공간효율이 최대 28% 정도 향상된 것을 확인하였다. 너무 작은 임계값으로 중복 스냅샷이 생성될 경우 델타 스냅샷의 크기가 증가하여 메모리 사용량이 증가하게 되고, 너무 큰 임계값으로 생성할 경우 중복 스냅샷 당 연결되어 있는 델타 스냅샷의 수가 감소하여 스냅샷의 수가 증가하기 때문에 위와 같이 적정 임계값에서 최적의 효율을 보이는 것을 확인할 수 있었다.

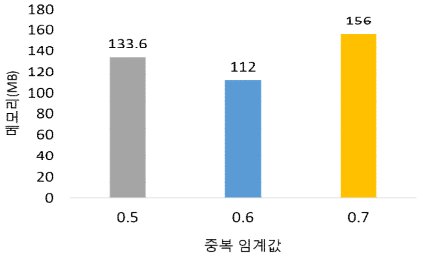


그림 7. 중복률에 따른 메모리 사용량

[그림 8]은 기존 기법과의 공간 효율을 비교하였다. 제안하는 기법은 가장 좋은 메모리 효율을 보인 임계값이 0.6인 상태이다. interval tree는 데이터 접근을 위해 모든 중복 데이터를 저장하기 때문에 제안하는 기법과 비교하여 약 80%의 공간 낭비를 보였고, DeltaGraph의 경우 이력의 추적을 위해 많은 양의 이력 데이터를 저장함으로써 제안하는 기법보다 약 47%의 많은 공간을 소모하였다. [그림 7]을 통해서 제안하는 기법은 효율적인 중복 데이터의 활용을 통해 비교 기법 보다 좋은 성능을 보이는 것을 확인할 수 있다.

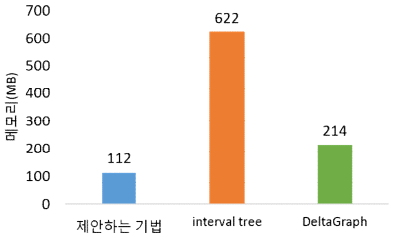


그림 8. 기존 기법과 제안하는 기법의 메모리 사용량 비교

3. 질의 처리 시간

[그림 9]은 각각의 중복률에 따른 질의 처리 시간을 비교하기 위해 앞서 진행한 0.5, 0.6, 0.7의 임계값으로 생성된 저장 구조의 질의 처리 시간을 비교하였다. 임계값이 0.6인 경우에 가장 좋은 성능을 확인할 수 있었다. 임계값이 작은 경우 하나의 중복 스냅샷 당 델타 스냅샷의 수가 많아 중복 스냅샷에 접근하는 시간이 감소하여 보다 좋은 성능이 보였다. 임계값이 큰 경우 많은 양의 중복 스냅샷의 생성으로 해당 중복 스냅샷으로의 접근 시간이 증가하여 질의 처리 시간이 증가하였다. 공간의 효율과 검색 성능을 고려하여 알맞은 임계값의 선택이 필요하다.

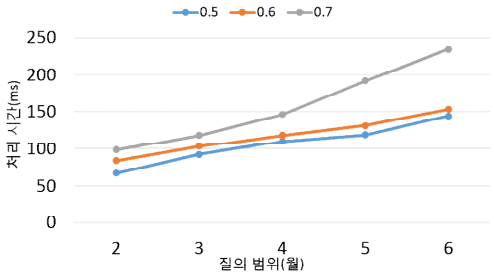


그림 9. 중복률에 따른 범위질의 처리 시간

[그림 10]은 질의 범위에 따른 처리 시간을 DeltaGraph와 비교한 것이다. 질의 범위는 월단위로 수행하였다. 적은 범위의 질의가 내려질 경우 DeltaGraph의 이력 접근 시간이 최소화되어 제안하는 기법 보다 다소 좋은 성능을 보였다. DeltaGraph는 질의 범위가 증가할수록 그래프 이력 접근 시간이 증가하여 처리 시간이 증가하는 모습을 보인다. 제안하는 기법은 질의 수가 증가하여도 중복 스냅샷에 연결된 델타 스냅샷 만

을 읽게 되어 처리 시간이 크게 증가하지 않는 것을 확인하였다.

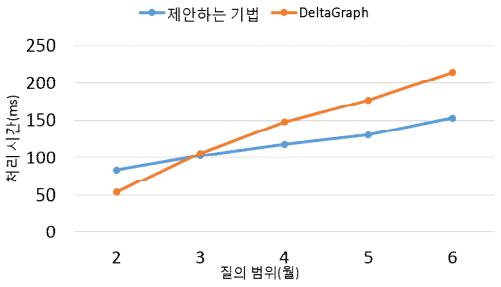


그림 10. 질의 범위에 따른 질의처리 시간

V. 결 론

본 논문에서는 동적 그래프에 대한 효과적인 이력 관리를 위한 그래프 관리 기법을 제안하였다. 시간에 따라 변화하는 동적 그래프를 중복 스냅샷과 델타 스냅샷으로 관리하여 중복되는 데이터를 최소화하였다. 변형된 CSR 표현 기법을 활용하여 이력 관리 정보량을 감소시킨다. 이력 그래프 검색 시 중복을 제외한 변화 이력만을 접근하여 검색성능 향상시킬 수 있었다. 제안하는 기법은 이력 그래프 데이터를 효율적으로 저장하기 위한 구조이다. 제안하는 기법은 소셜 네트워크와 같은 동적 그래프 데이터의 통계 분석을 위한 이력 그래프 접근에 효율적으로 활용될 수 있다. 실험 결과로 기존의 기법과 비교하여 최대 약 80%의 공간효율 향상을 확인하였다. 하지만 질의처리 결과 넓은 범위의 질의에서 기존의 기법보다 약 20%향상 되었지만, 작은 범위의 질의에 대해서 좋은 성능을 보장할 수 없었다. 향후 검색 성능을 더욱 향상시키기 위해서 최적화된 중복 데이터 검출과 순차적 접근 구조를 개선할 수 있는 색인 구조를 구축할 계획이다.

참 고 문 헌

[1] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “GraphX: graph processing in a distributed dataflow framework,” Proc. USENIX Symposium on Operating Systems Design and Implementation, pp.599-613, 2014.

[2] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” Proc. ACM SIGMOD International Conference on Management of Data, pp.135-146, 2010.

[3] 임종태, 복경수, 유재수, “대용량 그래프 환경에서 스카이라인을 이용한 서브 그래프 유사도 측정 기법,” 한국콘텐츠학회 종합학술대회, pp.47-48, 2017.

[4] 유병국, 김순홍, “소셜네트워크 분석을 통한 마케팅 전략,” 한국콘텐츠학회논문지, 제13권, 제5호, pp.396-407, 2013.

[5] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, “One trillion edges: Graph processing at facebook-scale,” Proceedings of the VLDB Endowment, Vol.8, No.12, pp.184-1815, 2015.

[6] https://carestruck.org/happens-internet-minute/

[7] H. He and A. K. Singh, “Graphs-at-a-time: query language and access methods,” Proc. ACM SIGMOD International Conference on Management of Data, pp.405-418, 2008.

[8] K. Semertzidis and E. Pitoura, “Time Traveling in Graphs using a Graph Database,” Proc. Workshops of the EDBT/ICDT 2016 Joint Conference, 2016.

[9] A. G. Labouseur, P. W. Olsen, and J. H. Hwang, “Scalable and Robust Management of Dynamic Graph Data,” Proc. International Workshop on Big Dynamic Distributed Data, pp.43-48, 2013.

[10] B. Salzberg and V. Tsotras, “Comparison of access methods for time-evolving data,” ACM Computing Surveys, Vol.31, No.2, pp.158-221,

1999.

[11] K. Semertzidis, E. Pitoura, and K. Lillis, "TimeReach: Historical Reachability Queries on Evolving Graphs," Proc. International Conference on Extending Database Technology, pp.121-132, 2015.

[12] U. Khurana and A. Deshpande, "Efficient snapshot retrieval over historical graph data," Proc. International Conference on Data Engineering, pp.997-1008, 2013.

[13] D. A. Bader, J. Berry, A. Amos-Binks, D. Chavarria-Miranda, C. Hastings, K. Madduri, and S. C. poulos, "STINGER: Spatio-temporal interaction networks and graphs (STING) extensible representation," Georgia Institute of Technology, Tech. Rep., 2009.

[14] K. Iwabuchi, S. Sallinen, R.Pearce, B. Van Essen, M. Gokhale, and S. Matsuoka, "Towards a Distributed Large-Scale Dynamic Graph Data Store," Proc. International Parallel and Distributed Processing Symposium Workshops, pp.892-901, 2016.

[15] P. Macko, V. J. Marathe, D. W. Margo, and M. I. Seltzer, "LLAMA: Efficient graph analytics using Large Multiversioned Arrays," Proc. International Conference on Data Engineering, pp.363-374, 2015.

[16] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, "Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide," SIAM, pp.315-336, 2000.

저 자 소 개

김 기 훈(Gihoon Kim) 준회원

- 2016년 2월 : 충북대학교 정보통신공학과(공학사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과 석사과정

<관심분야> : 그래프 저장구조, 스트림 그래프, 빅데이터 등

김 이 나(Ina Kim) 준회원

- 2016년 2월 : 충북대학교 생화학과(이학사)
- 2016년 9월 ~ 현재 : 충북대학교 빅데이터협동과정 석사과정

<관심분야> : 소셜네트워크, 데이터마이닝, 빅데이터 등

최 도 진(Dojin Choi) 준회원

- 2014년 2월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 한국교통대학교 컴퓨터공학과(공학석사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 연속 질의 처리, 그래프 스트림, 빅데이터 등

김 민 수(Minsoo Kim) 준회원

- 2013년 2월 : 충북대학교 정보통신학부(공학사)
- 2014년 9월 : 매크로 임팩트 연구원
- 2017년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2017년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 빅데이터, RDF, 그래프, 고차원 인덱스 등

복 경 수(Kyoungsoo Bok) 종신회원

- 1998년 2월 : 충북대학교 수학과(이학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2005년 8월 : 충북대학교 정보통신공학과(공학박사)
- 2005년 3월 ~ 2008년 2월 : 한국과학기술원 정보전자연구소 Postdoc
- 2008년 3월 ~ 2011년 2월 : 가인정보기술 연구소 연구원
- 2011년 3월 ~ 현재 : 충북대학교 전자정보대학 정보통신공학부 초빙교수

<관심분야> : 데이터베이스 시스템, 이동 객체 데이터베이스, 이동 P2P 네트워크, 소셜 네트워크 서비스, 빅데이터 등

유 재 수(Jaesoo Yoo) 종신회원

- 1989년 2월 : 전북대학교 컴퓨터공학과(공학사)
- 1991년 2월 : 한국과학기술원 전산학과(공학석사)
- 1995년 2월 : 한국과학기술원 전산학과(공학박사)
- 1995년 2월 ~ 1996년 8월 : 목포대학교 전산통계학과 전임강사
- 1996년 8월 ~ 현재 : 충북대학교 전자정보대학 정교수

<관심분야> : 데이터베이스 시스템, 멀티미디어 데이터베이스, 센서 네트워크, 바이오 인포메틱스, 빅데이터 등