https://doi.org/10.5392/JKCA.2022.22.01.035

그래프 스트림 처리를 위한 점진적 빈발 패턴 기반 인-메모리 압축 기법

In-memory Compression Scheme Based on Incremental Frequent Patterns for Graph Streams

이현병*, 신보경*, 복경수, 유재수***

충북대학교 정보통신공학과*, 원광대학교 SW 융합학과**

Hyeon-Byeong Lee(lhb@cbnu.ac.kr)*, Bo-Kyoung Shin(sbk02@cbnu.ac.kr)*, Kyoung-Soo Bok(ksbok@wku.ac.kr), Jae-Soo Yoo(yjs@cbnu.ac.kr)***

요약

최근 네트워크 기술 발전과 함께 IoT 및 소셜 네트워크 서비스의 활성화로 인해 많은 그래프 스트림 데이터가 생성되고 있다. 본 논문에서는 압축률 및 압축 시간에 대해 중점적으로 연구되던 기존의 압축 기법에 그래프 마이닝을 적용하여 스트림 그래프 환경을 함께 고려한 그래프 압축 기술을 제안한다. 또한, 최신 패턴을 유지하여 실시간으로 변화하는 스트림 그래프에서 압축 효율 및 처리속도를 향상시킨다. 본 논문에서는 그래프 스트림 처리를 위한 점진적 빈발 패턴 기반 압축 기법을 제안하였다. 제안하는 기법의 우수성을 보이기 위해 압축률과 처리시간을 기존기법과 비교하여 성능평가를 수행한다. 제안하는 기법은 그래프 데이터의 크기가 커질 때 중복되는 데이터가 많아져 기존 기법보다 빠른 처리속도를 보인다. 따라서, 빠른 처리가 요구되는 스트림 환경에서 제안하는 기법을 활용할 수 있다.

■ 중심어 : | 빅데이터 | 그래프 스트림 | 그래프 압축 | 빈발 패턴 | 프로버넌스 |

Abstract

Recently, with the development of network technologies, as IoT and social network service applications have been actively used, a lot of graph stream data is being generated. In this paper, we propose a graph compression scheme that considers the stream graph environment by applying graph mining to the existing compression technique, which has been focused on compression rate and runtime. In this paper, we proposed Incremental frequent pattern based compression technique for graph streams. Since the proposed scheme keeps only the latest reference patterns, it increases the storage utilization and improves the query processing time. In order to show the superiority of the proposed scheme, various performance evaluations are performed in terms of compression rate and processing time compared to the existing method. The proposed scheme is faster than existing similar scheme when the number of duplicated data is large.

■ keyword : | BigData | Graph Stream | Graph Compression | Frequent Pattern | Provenance |

* 본 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원 (No.2014-3-00123, 실시간 대규모 영상 데이터 이해·예측을 위한 고성능 비주얼 디스커버리 플랫폼 개발)(No.2021-0-02082, CDM-Cloud: Multi-Cloud 데이터 보호 및 관리 플랫폼), 중소벤처기업부 '산업전문인력역량강화사업'의 재원으로 한국산학연협회(AURI)(2021년 기업연계형연구개발 인력양성사업, 과제번호 : S3047889) 및 농촌진흥청 연구사업 (세부과제번호: PJ01624701)지원에 의해 이루어짐.

접수일자 : 2021년 09월 27일 심사완료일 : 2021년 10월 20일

수정일자 : 2021년 10월 20일

교신저자 : 유재수, e-mail : yjs@cbnu.ac.kr

I. 서론

최근 소셜 네트워크, 사물인터넷(Internet of Things), 모바일 기기, 생물 같은 복잡한 연결 관계를 표현하기 위해 그래프 데이터가 활용되고 있다. 예를 들어, 소셜 네트워크에서는 사용자는 정점으로 사용자와의 팔로우 친구관계는 간선으로 표현하고, 생물 정보학에서는 정점에는 유전자 정보, 유전자간 관계는 간선으로 표현할 수 있다. 이러한 서비스에서 생성되는 그래프는 실시간으로 확장되고 변화되며 정보의 양이 거대한 특징이 존재한다. 그래프 데이터는 그래프를 구성하는 정점, 간선이 동적으로 변화되는 특성을 가진다. 실제 환경을 고려했을 때 그래프 데이터는 실시간적으로 발생한다. 객체를 나타내는 정점의 추가/삭제/변경 또는 객체와 객체의 관계를 나타내는 간선의 추가/삭제/변경과 같은 그래프의 변화가 불규칙적으로 발생한다. 이러한 변화를 그래프 스트림이라 한다.

그래프 스트림은 빈발 패턴 검출, 이상 감지, 서브 그래프 매칭 등 다양한 분야에 사용되고 있다. 예를 들어, 페이스북, 인스타그램과 같은 소셜 네트워크 환경에서는 정점에 사용자의 정보(Lable)이 존재할 때, 사용자들이 실시간으로 친구 추가 또는 삭제가 이루어지거나 게시글에 댓글이나 좋아요를 남기는 등의 이벤트가 발생할 때마다 그래프가 갱신되고 생성된다. 소셜 네트워크에서 생성된 그래프는 사용자간의 관계 및 정보를 그래프로 표현하고 인적 네트워크를 분석하거나 유사한 사용자가 정보를 소비하는 패턴에 맞추어 관심사를 추천하는데 활용할 수 있다.

그래프를 실시간으로 처리하거나 많은 양의 그래프 데이터를 압축 저장하기 위한 기법[1-8]들이 연구됐다. 기존 그래프 압축 알고리즘은 그래프 마이닝을 이용하여 정점과 간선에 공통으로 나타난 그래프를 기준 패턴으로 결정하고, 기준 패턴에서 변경된 사항을 기술하는 방법을 사용한다. 기존의 그래프 마이닝 알고리즘은 대부분 정적 그래프를 입력으로 가정하고 있어 메모리에 저장하는 것에 한계가 있다. 기존 압축 알고리즘은 그래프의 기준 패턴과 완전히 새로운 패턴이 나타났을 때 패턴을 다시 탐색하는 과정이 필요해 압축률이 감소한다. 또한, 그래프를 압축하는 다른 기법은[3][4] 압축률은 높지만, 전처리 과정에서 수행시간이 오래 걸려 실시간으로 데이터가 생성되는 환경에 적용하기 어려운 문제점이 존재한다.

본 논문에서는 압축률 및 압축 시간에 중점적으로 연구되던 기존의 압축 기법에 그래프 마이닝을 적용하여 스트림 그래프 환경을 함께 고려한 그래프 압축 기술을 제안한다. 스트림 그래프에 대하여 효율적으로 압축을 수행하기 위해 시간에 따른 정점과 간선의 변화를 고려하여 압축하는 스트림 그래프 압축 기법을 제안한다. 제안하는 기법은 스트림 그래프를 인-메모리 환경에서 활용하기 위하여 정점과 간선의 변화를 프로버넌스 데이터를 사용하여 기록하고, 그래프 데이터를 사전 인코딩하여 압축한다. 패턴을 관리하면서 프로버넌스도 함께 관리되기 때문에 최신 패턴의 이력 관리 및 변화사항에 대해 확인한다. 그 결과, 그래프 데이터의 크기를 감소하여 인-메모리 환경에 많은 그래프 데이터를 유지하여 빠른 처리를 수행할 수 있게 한다. 또한, 최신 패턴을 유지하여 실시간으로 변화하는 스트림 그래프에서 압축 효율을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 연구들을 분석하고 기존 기법의 문제점을 기술한다. 3장에서는 제안하는 그래프 압축 기법에 대해 설명한다. 4장에서는 제안하는 기법의 우수성을 확인하기 위해 성능 평가 결과를 기술한다.

II. 관련연구

1. 그래프 압축 기법

그래프를 실시간으로 처리하거나 인-메모리 기반의 저장을 위하여 많은 양의 그래프 데이터를 압축 저장하기 위한 기법들이 연구됐다[1-11]. 그래프의 고차원의 복잡한 구조와 다양한 형태로 인해 기존의 그래프 마이닝 기술들을 직접적으로 적용하기 어려운 문제가 있다. 그래프 압축을 수행하기 위한 방법으로 크게 사전 기반 압축기법과 구조적 압축 기법 두 가지 접근 방향을 가지고 그래프 압축을 수행한다.

1.1 사전 기반 압축 기법

C. A. Packer가 제안한 GraphZip[1]은 최소 설명 길이 원칙(MDL)과 함께 사전 기반 압축 방식을 사용하여 그래프 스트림에서 최대 압축 패턴을 발견하는 기법을 제안했다. 그래프 스트림에서 이전에 확인한 패턴을 적용하고, 이전 패턴에서 확장된 새로운 패턴을 저장하여 압축률이 높은 패턴 사전을 만든다. [1]과 같은 대부분 알고리즘은 정점과 간선에 공통으로 나타난 그래프를 기준 패턴으로 결정하고, 기준 패턴에서 변경된 사항을 기술하는 방법을 사용한다. 하지만 [1]에서 제안한 기법은 그래프 패턴의 최대 크기가 정해져 있고, 일정 크기 이상 패턴이 늘어나면 패턴을 탐색하는 시간이 오래 걸려 스트림 환경에 적합하지 않다. 또한, 기준 패턴과 다른 패턴이 나타났을 때 적용할 수 있는 패턴이 적거나 없어 패턴을 다시 탐색하는 과정이 필요해 압축률이 감소한다. 본 논문에서는 이러한 문제를 해결하기 위해 빈발 패턴 마이닝 기법을 사용하며 [1]에서의 처리과정과 비교했을 때, 비교적 빠르게 결과를 도출하는 방법을 제안한다.

1.2 구조적 압축 기법

실제 스트림 환경에서 생성되는 그래프 데이터는 하나의 정점에 연결성을 지닌 Stars, Chains, Cliques, Tree, Bipartite forms와 같은 형태의 데이터 구조로 표현할 수 있다. 이러한 기법은 스트림 환경에서 그래프가 입력되었을 때 반복되는 데이터에서 자주 나오는 독특한 그래프 구조를 찾고 병합하여 새로운 데이터 구조를 만든다. 최근 제안된 구조적 압축기법에는 B. Dolgorsuren가 제안한 StarZIP[4] 알고리즘이 있다. [4]에서는 Star형의 서브 그래프 세트만을 고려해 찾고 재 정렬(Re-Order)하는 과정을 거쳐 압축을 위한 인코딩 기술을 적용하고, 그 그래프들을 점진적으로 업데이트하는 기법을 제안했다. [3][4]와 같은 알고리즘은 압축률은 높지만, 그래프의 구조(Star, Clique, Bipartite….)를 찾는 전처리 과정에서 수행 시간이 오래 걸려 실시간으로 데이터가 생성되는 환경에 적용하기 어려운 문제점이 존재한다. 변환하는 값이 커지면 비압축 데이터를 저장하는 것보다 효과가 떨어지게 된다. 또한, 그래프 구조를 발견하지 못하면 압축률이 떨어지고 성능 저하가 발생한다.

2. 빈발 패턴 마이닝

빈발 패턴 마이닝[12][13]은 데이터로부터 의미 있는 정보나 패턴을 추출하는 방법으로 데이터 분석, 생물정보학, 인덱스 등 다양한 분야에서 활용되고 있다. 이 문제를 푸는 대표적인 알고리즘인 FPgrowth[12] 알고리즘은 FP-tree라는 트리 구조를 사용하여 빈발 패턴을 효율적으로 찾아준다. 정적 환경에서 수행되는 일반적인 빈발 패턴 마이닝과 다르게 스트림 그래프에서 빈발 패턴 마이닝[12-15]은 데이터가 동적으로 변하고 정보가 많아져 데이터 관리를 위한 복잡도가 높아진다. C. Giannella가 제안한 FP-streaming 알고리즘[14]에서는 그래프 스트림 데이터가 시간마다 입력데이터의 양이 달라지는 것을 해결하기 위해 입력 시간별로 빈발 패턴을 검출한다. 자주 나오지 않는 패턴도 나중에 자주 사용될 수 있어 패턴에 반영해야 한다. 따라서 시간이 지남에 따라 빈도의 변화를 반영하도록 동적으로 조정된다. 이렇게 데이터를 관리함에 따라 모든 빈발 패턴을 정확하게 검출할 수 있다. 따라서 알고리즘은 패턴을 느리게 검출하지만 정확한 빈발 패턴만을 검출한다.

Ⅲ. 제안하는 그래프 압축 기법

1. 전체 처리 과정

본 논문에서는 실시간으로 입력되는 스트림 그래프에 대해 점진적으로 기준패턴을 찾아 압축하는 기법을 제안한다. 데이터를 최대로 압축하는 기준패턴에 대해 점진적으로 확인하고 인-메모리에 최대한 많은 양의 데이터를 유지하여 그래프 처리 속도를 향상시킨다. 스트림 그래프에서 빈발 패턴을 검출하고 압축에 적용할 때 세 가지 고려할 점이 있다. 첫째, 한정적인 저장 공간을 효율적으로 사용하고, 빠르게 빈발 패턴 검출을 해야 한다. 스트림 그래프에서 데이터는 연속적으로 입력되며 분석해야할 데이터가 실시간으로 변화된다. 따라서 어느 정도 데이터가 입력되었을 때, 데이터를 삭제하여 메모리를 확보해야 다음에 입력될 데이터에서 분석이

가능하다. 둘째. 시간에 따라 입력되는 데이터가 다르다는 것이다. 항상 일정한 패턴이 나오는 것이 아니라 시간에 따라 패턴이 달라지기 때문에, 현재 시점에서 패턴과 시간이 지난 후 패턴이 다를 수 있다. 따라서 패턴을 검출할 때 비교하는 서브 그래프의 동형성을 확인해야한다. 일반적인 서브그래프 동형 알고리즘은 그래프의 패턴이 완전 일치하는 패턴을 의미하지만 본 논문에서는 [그림 1]과 같이 그래프 G_1과 G_2가 존재할 때, G_1의 서브그래프가 G_2에 동형인 것을 의미한다. 따라서, $G_1 \subseteq G_2$인 패턴들도 동형인 그래프로 판단한다. 셋째, 시간과 빈도수를 고려하여 압축률이 가장 높은 패턴을 찾는 것이다. 빈발 패턴을 압축에 적용할 수 있도록 그래프를 패턴으로 묶어 저장해 데이터를 제한적인 메모리에 효율적으로 저장한다.

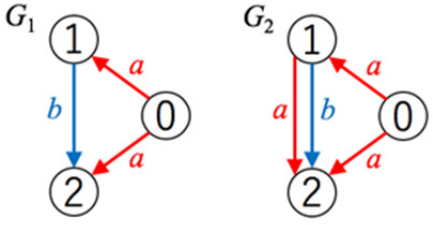

그림 1. 서브그래프 동형 패턴

[그림 2]는 제안하는 점진적 빈발패턴 압축 기법에 대한 전체 구조도를 나타낸다. 그래프 데이터가 들어올 때 기준 패턴 생성기(Reference Pattern Generator)는 빈발 서브 그래프 패턴을 추출해 점수가 가장 높은 패턴을 기준패턴으로 결정한다. 기준 패턴은 기준패턴 생성기로 추출된 패턴 중 압축률이 가장 높은 패턴을 의미한다. 이러한 과정을 반복하여 기준패턴을 전달함과 동시에 패턴 사전에 저장하여 다음 윈도우에서 활용한다. 기준 패턴은 기준패턴 생성기로 추출된 패턴 중 압축률이 가장 높은 패턴을 의미한다. 그래프 매니저(Graph Manager)는 스트림 환경에서 시간의 변화가 있을 때, 패턴 관리 정책을 사용하여 패턴을 관리하고 패턴 간의 유사도를 판별하는 역할을 수행한다. 패턴 사전(Pattern Dictionary)에는 기준 패턴 생성기에서 생성된 패턴에 관련된 모든 정보가 저장된다. 또한, 패턴에 대한 정보를 식별할 수 있는 정보가 함께 저장된다. 데이터는 저장 공간을 효율적으로 사용하기 위해 사전 인코딩을 수행해 그래프 데이터를 변환해 저장된다.

2. 기준 패턴 생성기

빈발 패턴은 그래프 데이터로부터 의미 있는 정보나 패턴을 추출하는 방법으로 데이터 분석, 생물 정보학, 인덱스 등 다양한 분야에서 활용되고 있다. 기준 패턴 생성기는 빈발 서브 그래프 패턴을 추출해 점수가 가장 높은 패턴을 기준 패턴으로 결정한다. 이러한 과정을 반복하여 기준패턴을 전달함과 동시에 패턴 사전에 저장하여 다음 윈도우에서 활용한다. 두 번째 탐색부터는 기존에 탐색 된 그래프와 현재 시점의 그래프 동형 여부를 검사하는 과정을 반복하여 수행하고 그래프를 확장한다.

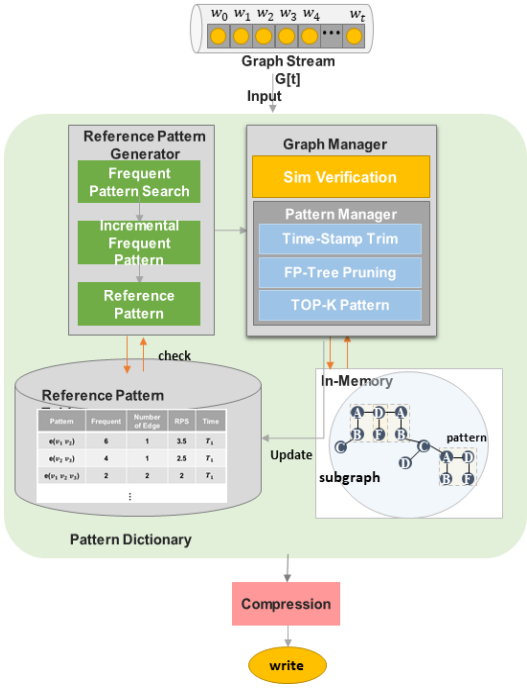

그림 2. 제안하는 점진적 빈발 패턴 압축 기법 전체 구조도

기준패턴 생성기에서는 [그림 3]과 같이 초기 빈발 패턴 탐색(Initial Frequent Patterns Search), 점진적 빈발 패턴 탐색(Incremental Frequent Pattern

Search), 빈발 패턴 추출(Extracting Reference Pattern) 3가지 과정을 거쳐 점수가 가장 높은 패턴을 기준패턴으로 결정한다.

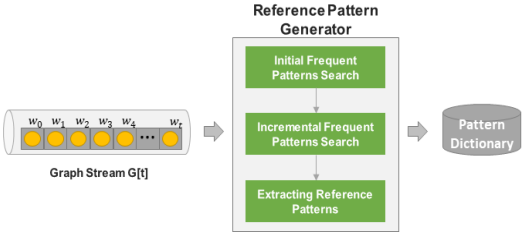

그림 3. 기준패턴 생성기 수행 과정

2.1 초기 빈발 패턴 탐색(Initial Frequent Patterns Search)

초기 빈발 패턴 탐색(Initialize Frequent Pattern Search)에는 그래프가 기존에 존재하지 않거나 또는, 사전에 패턴이 없을 때 수행된다. 이러한 경우 패턴 사전에 저장된 기준 패턴이 존재하지 않는다. 따라서, 기준패턴 생성기에서 그래프를 배치 크기로 탐색하여 초기에 단일 간선 패턴을 찾는다. [그림 4]는 초기 빈발 패턴 알고리즘을 나타낸다. 배치 그래프 G_B의 간선 e는 $(v_{source}, v_{target}, l)$이다. 단일 간선 그래프를 생성한 뒤 리스트의 순서대로 앞에 들어오는 정점은 v_{source} 뒤에 들어오는 정점은 v_{target}으로 수정하고, 간선은 $(v_{source}, v_{target}, l)$로 추가한 뒤, 패턴사전에 저장하게 된다.

Algorithm 1. Initialize Frequent Pattern

Require : Streaming Graph G
Ensure : Pattern Dictionary P

```
for e in E_{G_B}
  single_edge = new Graph()
  single_edge.add_vertex(id=0, label=label[e_src])
  single_edge.add_vertex(id=1, label=label[e_tgt])
  single_edge.add_edge(0,1 label=label[e])
  update_pattern(single_edge)
end for
```

그림 4. 초기 빈발 패턴 알고리즘

2.2 점진적 빈발 패턴 탐색(Incremental Frequent Pattern Search)

점진적 빈발 패턴 탐색(Incremental Frequent Pattern Search)에서는 그래프에서 찾은 패턴에 속한 서브그래프에 대해 반복적으로 수행하게 되며 첫 번째 과정에서는 패턴이 존재하지 않기 때문에 수행되지 않는다.

[그림 5]는 점진적 빈발 패턴 알고리즘을 나타낸다. 이 과정에선 초기 빈발 패턴 탐색에서 나타난 단일 간선 패턴을 점진적으로 업데이트하는 과정을 수행한다. 그래프에서 찾은 서브 그래프 셋에 속한 각 서브 그래프에 대해 확인하고 G(v) -〉 P(v)를 반복 수행하면서 정점을 인덱스로 매칭한 뒤 각 서브 그래프를 정점이 들어오는 순서대로 인덱스로 매칭 시켜 그래프의 크기를 감소시킨다.

[그림 6]과 같이 그래프 스트림에서 $G(v)$와 같이 정점이 순서대로 들어올 때, 패턴 사전에 저장되는 $P(v)$는 들어오는 순서대로 인덱스로 매칭되어 저장되게 된다. $G(v) = [132, 12, 51, 43, 21, 19, 12, 3 \ldots]$인 정점이 들어올 때 그래프의 정점이 들어온 순서대로 $P(v) = [0,1,2,3,4,5,1,6 \ldots]$으로 매칭 된다. 이미 인덱스가 배정된 정점은 기존 인덱스를 사용한다. 그 후, 두 정점이 인접한 정점인지를 확인하고 확장을 진행한다. 제안하는 알고리즘에선 g(e)가 p(e)보다 작을 때 패턴 확장을 진행하게 된다. 케이스를 3가지로 나누어 확장하는데 $g_{src}(v)$가 e에 존재하지 않을 때, $g_{tgt}(v)$가 e에 존재하지 않을 때, 그 이외의 케이스를 나누어 확장한 뒤 패턴을 업데이트한다. $G(v)$와 $P(v)$에 저장된 정점이 패턴에 저장된 간선과 연결되어있는지 연결성을 확인하여 $G(v)$값이 더 높으면 패턴을 확장하는 과정을 수행한다. g(e)의 간선 e가 패턴에 존재하지 않으면 패턴을 확장하고 존재하면 사전에 더한다.

Algorithm 2. Incremental Frequent Pattern

Require : Pattern Dictionary P

Ensure : Update Pattern Dictionary P'

```
for g in g_B
  count +=1
  G_V = Dict([g(v) :p(v) for p(v), g(v) in g])
  for p(v), g(v) in g
    g(e) = g_B(e)[g_B.incident(g(v))]
    p(e) = p(e)[p.incident(p(v))]
    if g(e)<p(e)
      continue
    for e in g(e)
      g_src(v) = e.src
      g_tgt(v) = e.tgt
      if p_new is None
        p_new = p.copy()
        if g_src(v) not in e
          p_new.add_vertex = p^i_src
          p_new.add_edge = (p^i_src, p^i_tgt, l)
        else if g_tgt(v) not in e
          p_new.add_vertex = p^i_tgt
          p_new.add_edge = (p^i_src, p^i_tgt, l)
        else
          safe_add_edge(p_new, p^i_src, p^i_tgt, l)
        end if
      end if
    end for
    end if
  end for
end for
```

그림 5. 점진적 빈발 패턴 알고리즘

G(v)	132	12	51	43	21	19	12	3
	↓	↓	↓	↓	↓	↓	↓	↓
P(v)	0	1	2	3	4	5	1	6

G(v) : 원본 그래프의 정점, P(v) : 패턴 사전에 저장되는 정점

그림 6. 원본그래프의 정점과 패턴사전에 저장되는 정점

2.3 빈발 패턴 결정(Reference Pattern Extract)

세 번째, 빈발 패턴 결정(Reference Pattern Extract)에서는 앞서 수행된 알고리즘을 바탕으로 최종적으로 서브 그래프 패턴의 빈발도와 간선의 크기를 고려하여 패턴 점수를 결과가 높은 순서대로 정렬한다. 계산 결과가 높을수록 자주 나오고 그래프 연결이 많은 패턴으로 판단하여 기준패턴으로 채택된다. 패턴이 그래프 G_B의 서브그래프 g에 대해 서브 그래프 동형이면 패턴 점수인 RPS를 계산하여 패턴사전에 업데이트한다. 패턴 점수를 계산하는 수식 (1)은 다음과 같다.

$$RPS = (Edge\ Size * \alpha) + (Frequent * \beta) \qquad (1)$$

3. 그래프 관리자

그래프 관리자는 스트림 그래프에서 시간의 변화가 있을 때, 패턴 관리 정책을 사용하여 패턴을 관리하는 역할을 수행한다. 또한, 그래프의 유사도를 판단해 기존과 다른 패턴이 나오면 기준 패턴 생성기를 재 수행시킨다. [그림 7]은 그래프 관리자의 수행 과정을 나타낸다. 그래프 관리자에서 패턴 관리 정책은 기존 패턴과의 일치성과 중요도를 판별하여 결정된다. 그래프 관리자에서는 패턴 유사도 검증(Similarity Verification), 패턴 관리자(Pattern Manager) 두 가지 컴포넌트를 활용하여 그래프를 관리한다. 패턴 유사도 검증에서는 패턴사전에 저장된 기준 패턴들과 시간 t에서 새로 들어온 패턴을 비교하여 유사도를 검증한다. 패턴 유사도 검증에서는 서브 그래프 동형성 검사를 수행하여 유사도를 판별하는 역할을 수행한다. 이때 유사도는 VF2[8] 알고리즘을 수행하여 계산한다. 패턴 관리자에서는 Time-Stamp와 FP-Tree를 활용한 Pruning, 패턴 사전에 저장되는 패턴의 크기 제한을 통해 패턴의 중요도를 결정하고 관리한다.

3.1 패턴 유사도 검증

패턴 유사도 검증은 기존의 기준 패턴과 새로 들어온 패턴의 유사성을 판별하기 위한 단계이다. 서브 그래프 동형검사를 수행하여 기준패턴과 새로 들어온 패턴 G_t의 유사도를 계산한다. 대부분의 압축 알고리즘에서 기준 패턴은 정점과 간선에 공통으로 나타난 그래프를 기준 패턴으로 결정하고, 기준 패턴에서 변경된 사항을 기술하는 방법을 사용한다. 하지만 그래프 패턴이 완전히 일치하는 패턴만 탐색한다면 압축에 적용할 패턴의 개수가 적어져 압축에 적용하기 어렵다. 제안하는 기법

에서는 전체 그래프가 아닌 기준패턴생성기에서 RPS값을 통해 정해진 기준패턴과 새로 들어온 패턴의 유사성을 판별해 비교 연산량을 줄일 수 있다. 그래프 간 유사도 여부를 VF2[8] 알고리즘을 수행하여 계산 한다. VF2는 서브그래프 동형성을 판별하는 알고리즘으로, $G_1 \subseteq G_2$인 패턴들도 동형인 그래프로 판단한다. 따라서, 완전 일치하지 않은 패턴도 동형으로 판단하여 다른 알고리즘보다 유연하게 패턴 유사도를 검증한다.

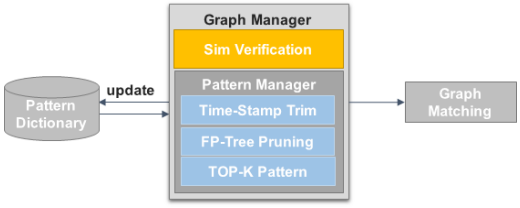

그림 7. 그래프 관리자 수행 과정

3.2 패턴 관리자

패턴 관리자에서는 Time-Stamp와 FP-Tree를 활용한 Pruning, 패턴 사전에 저장되는 패턴의 크기 제한을 통해 패턴의 중요도를 결정하고 관리한다. 최종적으로 세 가지 요소를 적절하게 조합하여 정규화된 패턴 점수(Pattern Score)를 활용하여 점수가 높은 순서대로 패턴을 사전에 유지하게 한다.

스트림 그래프에서는 항상 일정한 패턴이 나오는 것이 아니라 시간에 따라 패턴이 달라지기 때문에, 현재 시점에서 패턴과 시간이 지난 후 패턴이 다를 수 있다. 이러한 패턴을 고려하기 위해 다음과 같은 정책을 사용한다. Time-Stamp Trim에서는 임계값을 정해 임계값인 τ의 Window가 지나면 패턴을 삭제한다. 또한 윈도우에서 시간이 지날 때, Time-Stamp를 업데이트 한다. 그래프는 여러 개의 배치(B_n)가 윈도우(w_n)로 지정되며, 배치의 크기는 사용자가 지정한다. 예를 들어 그림 8과 같이 Batch의 크기가 3으로 지정된 윈도우 W_0가 존재하면 $W_0 = \{B_1, B_2, B_3\}$이다. $\tau = 4$이므로, $W_0 \sim W_3$의 패턴은 유지된다.

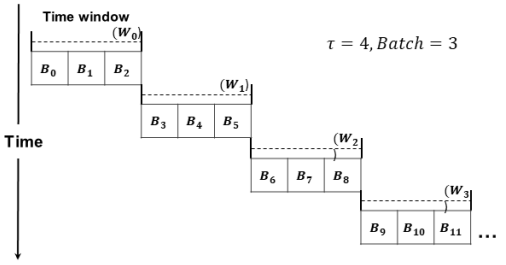

그림 8. 시간이 지남에 따라 변화하는 Time window

제안하는 기법에서는 [그림 9]와 [그림 10]같이 스트림 그래프에서 윈도우와 배치를 이용해 시간을 고려하여 사전을 관리한다. 사전에 저장된 기준 패턴은 업데이트/삭제하며 관리하게 된다. 예를 들어 [그림 9]와 같이 배치의 크기가 3인 윈도우가 존재할 때, 모든 정점이 처음 나온 패턴이라면 Time-stamp를 현재 시점인 T로 유지한다. Window의 크기가 3일 때, Batch 1에서 P_1, P_2, P_3의 패턴이 등장하고 Batch 2는 Batch1에서 등장한 패턴인 P_1과 P_2의 확장패턴인 P_{1+2}가 추가되어 P_1, P_2, P_3, P_{1+2} 총 4개의 패턴이 등장한다. Batch3은 Batch1, Batch2에서 나온 P_1패턴이 제외된 P_2, P_3, P_{1+2}, P_4의 패턴이 등장한다. 따라서 W_0에 패턴은 $P_1, P_2, P_3, P_{1+2}, P_4$가 존재한다. [그림 10]과 같이 $W_0 \rightarrow W_1$로 시간이 변화할 때 패턴은 업데이트 된다. W_1에 W_0과 같은 패턴이 계속 나온다면 Time-Stamp를 현재 시점인 T로 계속 유지하지만, W_1에 나오지 않은 패턴인 P_1과 P_{1+2}는 이전 시점에만 존재하므로, $T-1$으로 Time-Stamp를 수정한다. 이전 시점에서 나오지 않았지만 현재에 추가된 패턴인 P_5와 P_{2+3}은 추가된다. 이러한 과정을 반복 수행하여 Time-Stamp를 수정하고 임계값인 $T = \tau$인 시점 동안 윈도우에서 등장하지 않은 패턴은 삭제된다. 이렇게 패턴을 업데이트/삭제하는 과정을 통해 자주 나오는 패턴은 유지해 압축에 사용하고 자주 나오지 않는 패턴은 삭제하여 관리한다. 이렇게 패턴을 관리하여 그래프를 빠르게 처리할 수 있고, 패턴을 삭제하여 메모리를 확보해 다음 스트림에서 처리속도를 빠르게 할 수 있다.

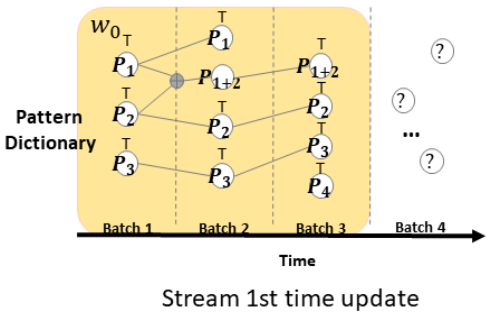

그림 9. 첫 번째 스트림에서 시간을 고려한 패턴 업데이트

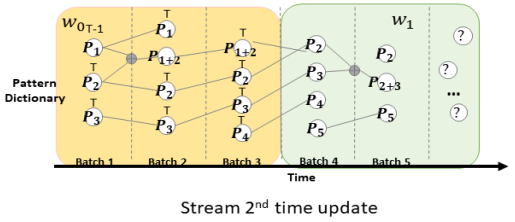

그림 10. 두 번째 스트림에서 시간을 고려한 패턴 업데이트

제안하는 기법에서는 한정적인 저장 공간을 효율적으로 사용하기 위해 사전에 저장된 패턴을 FP-Growth 알고리즘에 적용해 자주 나오지 않는 패턴을 pruning 한다. 이러한 정책을 수행하여 데이터의 크기를 감소시켜 인-메모리 환경에 많은 그래프 데이터를 유지 및 처리 수행할 수 있고, 자주 사용되지 않는 패턴들을 패턴 사전에서 업데이트/삭제해 최신 패턴을 유지하여 실시간으로 변화하는 스트림 그래프에서 압축 효율 향상시킨다. [그림 11]은 FP-Growth알고리즘을 적용한 Pruning 과정의 예시이다. 패턴P에는 $(pid, B, T, Frequent)$의 정보가 저장된다. [그림 11]에 적용된 Frequent Threshold =3, Time Threshold = 2 일 때, 리프노드가 일정 임계값 이하의 값이 나오는 패턴은 자주 나오지 않는 패턴으로 판단하고 Pruning한다. 윈도우 w_0에서 임계값을 만족하지 못하는 패턴 P_1, P_{1+2}는 Pruning된다. 또한, 윈도우 w_1에서 임계값을 만족하지 못하는 패턴 P_5도 Pruning된다. 이러한 과정을 통해 패턴을 유지하는 비용을 줄이고 다음 스트림에 들어오는 데이터를 유지할 수 있는 메모리상 공간을 만든다.

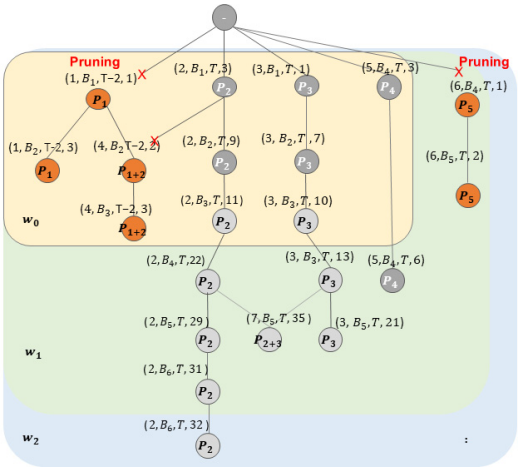

그림 11. FP-Growth알고리즘을 적용한 Pruning 과정

4. 패턴 사전 (Pattern Dictionary)

모든 패턴은 사전이라 불리는 메모리 내 공간에 저장된다. 패턴사전에는 기준패턴 생성기에서 생성된 패턴에 관련된 모든 정보들이 저장된다. 그래프 패턴은 패턴 사전에 저장될 때 그 정점과 간선이 그래프에서 얼마나 자주 나온 패턴인지를 고려하여 빈도수가 같이 저장된다. 기준 패턴은 패턴의 정보를 식별할 수 있는 패턴 ID, Time-Stamp, 기준패턴 점수 등의 패턴의 정보인 프로버넌스 데이터(Provenance Data)가 함께 저장된다. 프로버넌스 데이터를 활용하여 그래프 패턴 마이닝으로 추출된 패턴에 프로버넌스 정보를 활용해 그래프 변화 식별 및 이력에 대한 기록을 활용할 수 있다. 데이터는 사전 인코딩(dictionary encoding)을 수행하여 그래프 데이터를 변환해 저장한다. 이렇게 변환된 그래프 패턴을 다시 인코딩하면 그래프로 패턴을 그냥 저장하는 것보다 작은 용량을 가지게 되어 저장 공간을 효율적으로 사용할 수 있다. [그림 12]는 패턴사전에 저장되는 정보이다. 패턴 ID, 패턴, 빈발도, 간선의 크기, 기준패턴 점수, Time-Stamp, Prov, Batch와 같은 패턴의 정보와 프로버넌스 데이터(Provenance Data)가 함께 저장된다. 기준패턴생성기와 그래프 매니저에서는 패턴사전에 저장되는 내용을 활용하여 기준패턴을 정하고, 어떤 패턴을 자주 나오는 패턴으로 판단할지를 결정 한다.

P_id	Pattern	Frequent	Edge size	RPS (Score)	Time Stamp	Prov	Batch
1	v:01 v:119 e:013	5	3	15	T	I_1	B_1
2	v:01 v:12 e:012	5	3	15	T	I_1	B_1
3	v:02 v:13 v:23 e:0125	2	4	8	T	I_1	B_1
4	v:027 v:113 e:016	2	3	6	T	I_1	B_2
5	v:02 v:12 e:014	2	3	6	T	I_1	B_2

그림 12. 제안된 기법에서 패턴 사전에 저장되는 정보

Ⅳ. 성능평가

1. 성능평가 환경

본 논문에서는 제안하는 기법의 우수성을 보이기 기존 기법[1]과 비교를 통해 성능평가를 수행하였다. [1]은 사전 기반 압축 방식을 사용하여 그래프 스트림에서 최대 압축 패턴을 발견하는 기법으로 그래프 스트림에서 이전에 확인한 패턴을 적용하고, 이전 패턴에서 확장된 새로운 패턴을 저장하여 압축률이 높은 패턴 사전을 만든다.

[표 1]은 성능평가 환경을 보여준다. 성능평가 환경은 Intel(R) Core(TM) i7-9700K 3.60GHz 프로세서, 32GB 메모리, 256GB 디스크 환경으로 구성되고 Python igraph[15]를 사용해 성능평가를 수행하였다. igraph는 그래프 네트워크 구조를 생성해주는 툴로 정점과 간선 레이블로 구성된다.

표 1. 성능 평가 환경

구 분	내 용
프로세서	Intel(R) Core(TM) i7-9700K 3.60GHz
메모리	32 GB
디스크	256 GB
프로그램 언어	Python 3.6.13 igraph 0.9.1

수식 (2)는 원본 그래프 대비 압축된 그래프의 크기를 비교하는 식으로 원본 데이터의 크기에 비례하여 데이터가 얼마나 압축되었는지 확인할 수 있다. 성능 평가는 데이터를 스트림 환경에 처리속도를 빠르게 하고 최대로 압축하는 기준패턴을 확인하기 위해 패턴사전의 크기에 따른 수행시간과 압축률의 변화에 대해 평가한다. 또, 한 번에 처리되는 데이터 크기인 배치의 변화에 따른 압축률과 수행평가 비교를 수행한다. 마지막으로 그래프 데이터 크기에 따른 압축률 및 수행시간의 비교를 수행한다.

$$Compression\ rate = \frac{Size\ of\ Comressed\ Graph}{Size\ of\ Original\ Graph} \times 100 \quad (2)$$

2. 성능평가 결과

그래프 스트림 환경에서 패턴 사전에 저장되는 데이터가 적을 경우, 그래프와 비교할 수 있는 데이터가 부족해 압축률이 떨어질 수 있다. 반면에 패턴사전에 저장되는 데이터가 많을 경우, 스트림 환경에서 데이터 처리시간이 길어져 오버로드가 발생할 수 있다. 패턴 사전 크기에 따른 처리 속도에 차이가 생기기 때문 값을 변경하여 실험을 수행한다. [그림 13]은 패턴 사전 크기에 따른 수행시간과 압축률을 비교한다. 실험 평가에 사용되는 데이터는 간선의 크기가 10만개인 가상 데이터를 사용하였다. 배치의 크기가 20일 때, 최적의 수행시간과 압축률을 갖는 패턴사전의 크기를 비교한다. 이 실험에서는 패턴사전의 크기에 따라 압축률은 크게 변하지 않지만, 150이상의 패턴사전 크기를 적용하면 수행시간이 약 50%까지 감소하는 것을 볼 수 있다. 패턴 사전 크기가 150이상부터는 비슷한 수행시간을 보인다. 이러한 이유는 처음에는 사전에 저장되는 패턴이 많지만, 일정 크기 이상 패턴이 저장되면 패턴사전에 저장되는 패턴이 크게 증가하지 않기 때문이다.

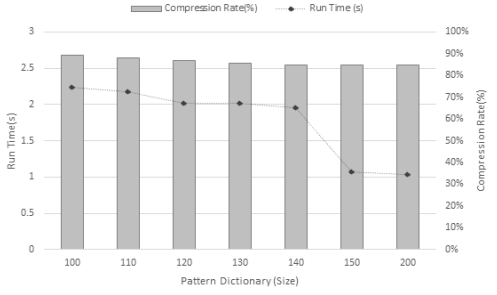

그림 13. 패턴 사전 크기에 따른 수행시간과 압축률 비교

만약 그래프 스트림의 입력속도가 처리속도보다 빠르다면, 검출하기 이전에 새로운 그래프가 입력되어 유실되는 빈발 패턴이 발생할 수 있다. 따라서 응용에 맞는 적절한 배치 크기를 설정해 최적의 배치크기를 찾는 것이 중요하다. [그림 14]는 배치 크기에 따른 제안 기법과 기존기법인 GraphZip의 처리속도를 비교한다. 실험 평가에 사용되는 데이터는 간선의 크기가 10만개인 가상 데이터를 사용 하였다. 배치는 10개의 그래프로 구성된다. 또한 패턴 사전의 크기는 150일 때, 배치크기에 따른 처리속도를 비교한다. 제안하는 기법에서는 배치크기가 10이하일 때 처리속도가 가장 빨랐으며, 20이후 에는 비슷한 처리 속도를 보임 보인다. 기존 기법인 GraphZip과 비교 했을 때, 처리속도가 평균 2.3배 빨랐으며 배치크기가 10이하 일 때 처리속도 차이가 가장 컸다. 이 결과를 통해 배치의 개수가 증가할수록 제안하는 기법의 성능이 좋아진다는 것을 확인할 수 있다. 빠른 처리가 요구되는 스트림 환경에는 제안하는 기법이 더 적합할 것으로 판단할 수 있다.

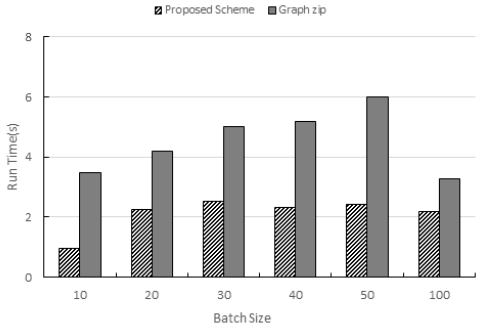

그림 14. 배치 크기에 따른 처리 시간 비교

Ⅴ. 결론

본 논문에서는 그래프 스트림 처리를 위한 점진적 빈발 패턴 기반 압축 기법을 제안하였다. 그래프 패턴 마이닝으로 빈발 패턴을 추출하고 점수를 계산하여 기준 패턴을 찾는다. 또한, 패턴과 프로버넌스 정보를 활용해 그래프 변화 식별 및 이력에 대한 기록을 활용한다. 이렇게 함으로써 기존 기법보다 빠르게 패턴을 검출하고 패턴을 압축에 적용할 수 있다. 제안한 압축 기법은 실시간으로 변화하는 스트림 그래프에서 적용할 수 있다. 그래프 데이터의 크기를 감소하여 인-메모리 환경에 많은 그래프 데이터를 유지하여 빠른 처리를 수행할 수 있게 한다. 또한, 최신 패턴을 유지하여 실시간으로 변화하는 스트림 그래프에서 압축 효율 및 처리속도를 향상시킨다. 성능평가 결과 제안하는 기법과 기존 기법을 비교하였을 때 데이터의 크기와 임계값의 변경에 따라 평균 2.3배 ~ 3.2배의 속도 차이를 보였다. 또한 150이상의 패턴사전 크기를 적용하면 수행시간이 약 50%까지 감소하는 것을 볼 수 있다. 실험 평가를 통해 배치의 개수, 패턴사전의 크기에 따라 제안하는 기법과 기존 기법의 압축률은 비슷하지만 처리 속도는 좋아진다는 것을 확인할 수 있다. 빠른 처리가 요구되는 스트림 환경에는 제안하는 기법이 더 적합할 것으로 판단할 수 있다. 하지만 그래프 스트림의 입력속도가 처리속도보다 빠르다면 패턴 검출 이전에 새로운 그래프가 입력되어 유실되는 패턴이 발생할 수 있다. 또한, 기존의 기법과 비교하였을 때 압축률은 큰 차이를 보이지 않았다. 따라서 향후 연구에서는 이러한 문제점을 해결하기 위해 압축률을 향상시킬수 있도록 시스템을 구현하고 추가적인 기존 기법과의 비교를 통해 성능을 향상시킬 예정이다.

참 고 문 헌

[1] P. Charles and H. B. Lawrence, “GraphZip: Mining graph streams using dictionary-based compression,” in Proc. SIGKDD Workshop Mining Learn. Graphs, 2017.

[2] S. Maneth and F. Peternek, “Grammar-based graph compression,” Information Systems, Vol.76, pp.19-45, 2019.

[3] R. A. Rossi and R. Zhou, “GraphZIP: A clique-based sparse graph compression method,” J. Big Data, Vol.5, No.1, pp.1-14, 2018.

[4] B. Dolgorsuren, K. Khan, M. K. Rasel, and Y. Lee, “StarZIP: Streaming Graph Compression Technique for Data Archiving,” IEEE Access, pp.38020-38034, 2019.

[5] K. S. Bok, J. E. Han, J. T. Lim, and J. S. Yoo,

"Provenance compression scheme based on graph patterns for large RDF documents," The Journal of Supercomputing, Vol.76, No.8, pp.6376-6398, 2020.

[6] J. H. Lee and F. Liu "An Efficient Graph Compressor Based on Adaptive Prefix Encoding," SSDBM '19, pp.85-96, 2019.

[7] Sebastian Maneth and Fabian Peternek, "Applying Grammar-Based Compression to RDF," ESWC 2021, pp.93-108, 2021.

[8] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "TimeCrunch: Interpretable dynamic graph summarization," in Proc. 21th ACMSIGKDD Int. Conf. Knowl. Discovery Data Mining, pp.10551064, 2015.

[9] W. Henecka and M. Roughan, "Lossy compression of dynamic, weighted graphs," in Proc. 3rd Int. Conf. Future Internet Things Cloud, pp.427434, Aug. 2015.

[10] L. Dhulipala, I. Kabiljo, G. Ottaviano, S. Pupyrev, and A. Shalita, "Compressing graphs and indexes with recursive graph bisection," in Proc.22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, pp.15351544, 2016.

[11] Y. Lim, U. Kang, and C. Faloutsos, "SlashBurn: Graph compression and mining beyond caveman communities," IEEE Trans. Knowl. Data Eng., Vol.26, No.12, pp.30773089, Dec. 2014.

[12] J. Han, J, Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," in Proc. ACM SIGMOD Int. Conf. Manage. Data, pp.1-12, 2000.

[13] C Borgelt, "An implementation of the FP-growth algorithm," Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, 2005.

[14] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining frequent patterns in data streams at multiple time granularities," Next generation data mining, Vol.212, pp.191-212, 2003.

[12] M. ZARROUK, "Frequent Patterns mining in time-sensitive Data Stream," International Journal of Computer Science Issues, Vol.9, No.4, pp.1467-1470, 2012.

[13] 복경수, 한지은, 노연우, 육미선, 임종태, 이석희, 유재수, "RDF 그래프 패턴을 고려한 프로버넌스 압축 기법," 한국콘텐츠학회논문지, 제16권, 제2호, pp.374-386, 2016.

[14] 정재윤, 서인덕, 송희섭, 박재열, 김민영, 최도진, 복경수, 유재수, "그래프 스트림에서 슬라이딩 윈도우 기반의 점진적 빈발 패턴 검출 기법," 한국콘텐츠학회논문지, 제18권, 제2호, pp.147-157, 2018.

[15] G. Csardi and T. Nepusz, "The igraph software package for complex network research," InterJournal, complex systems, Vol.1695, No.5, pp.1-9, 2005.

저 자 소 개

이 현 병(Hyeonbyeong Lee) 정회원

- 2016년 8월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2018년 8월 : 한국교통대학교 컴퓨터공학과(공학석사)
- 2019년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

〈관심분야〉 : 빅 데이터, 그래프 스트림, 그래프 마이닝, 데이터베이스 시스템

신 보 경(Bokyoung Shin) 준회원

- 2016년 2월 : 충북대학교 정보통신공학부(공학사)
- 2019년 3월 ~ 현재 : 충북대학교 정보통신공학과(석사과정)

〈관심분야〉 : 빅 데이터, 그래프 스트림, 그래프 마이닝, 데이터베이스 시스템

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 차장
- 2011년 3월 ~ 2019년 8월 : 충북대학교 전자정보대학 정보통신공학부 초빙교수
- 2011년 9월 ~ 현재 : 원광대학교 SW융합학과 조교수

〈관심분야〉 : 데이터베이스 시스템, 이동 객체 데이터베이스, 이동 P2P 네트워크, 소셜 네트워크 서비스, 빅데이터 처리 등

유 재 수(Jaesoo Yoo) **종신회원**

- 1995년 2월 : KAIST 전산학과(공학박사)
- 1995년 2월 ~ 1996년 8월 : 목포대학교 전산통계학과 전임강사
- 1996년 8월 ~ 현재 : 충북대학교 전자정보대학 정보통신공학부 정교수
- 2009년 8월 ~ 2010년 2월, 2019년 9월 ~ 2020년 8월 : California State University, Fullerton 방문교수

〈관심분야〉 : 데이터베이스시스템, 인텔리전트 DB, 소셜 미디어, 빅데이터 등