https://doi.org/10.5392/JKCA.2018.18.02.147

그래프 스트림에서 슬라이딩 윈도우 기반의 점진적 빈발 패턴 검출 기법

Incremental Frequent Pattern Detection Scheme Based on Sliding Windows in Graph Streams

정재윤*, 서인덕**, 송희섭**, 박재열*, 김민영*, 최도진*, 복경수*, 유재수*
충북대학교 정보통신공학과*, 충북대학교 빅데이터학과**

Jaeyun Jeong(jjjeong@chungbuk.ac.kr)*, Indeok Seo(duckdeock@chungbuk.ac.kr)**, Heesub Song(heeusb@chungbuk.ac.kr)**, Jaeyeol Park(yeols@chungbuk.ac.kr)*, Minyeong Kim(requem_@naver.com)*, Dojin Choi(mycdj91@chungbuk.ac.kr)*, Kyoungsoo Bok(ksbok@chungbuk.ac.kr)*, Jaesoo Yoo(yjs@chungbuk.ac.kr)*

요약

최근 네트워크 기술 발전과 함께 IoT 및 소셜 네트워크 서비스의 활성화로 인해 많은 그래프 스트림 데이터가 생성되고 있다. 이와 같은 그래프 스트림에서 객체들 사이의 관계가 동적으로 변화함에 따라 그래프의 변화를 탐지하거나 분석하기 위한 연구들이 진행되고 있다. 본 논문에서는 그래프 스트림에서 이전 슬라이딩 윈도우에서 검출한 빈발 패턴에 대한 정보를 이용해 빈발 패턴을 점진적으로 검출하는 기법을 제안한다. 제안하는 기법은 이전 슬라이딩 윈도우에서 검출된 패턴이 앞으로 몇 슬라이딩 윈도우동안 빈발할지 또는 빈발하지 않을지를 계산하여 빈발 패턴 관리 테이블에 저장한다. 그리고 이 값을 통해 다음 슬라이딩 윈도우에서는 필요한 계산만 수행함으로써 전체 연산량을 감소시킨다. 또한 패턴 간에 간선을 통해 연결되어 있는 것만 하나의 패턴으로 인식함으로써 더 유의미한 패턴만을 검출한다. 본 논문에서는 제안하는 기법의 우수함을 보이기 위해 여러 성능 평가를 진행하였다. 그래프 데이터의 크기가 커지고 슬라이딩 윈도우의 크기가 커질수록 중복되는 데이터가 증가되기 때문에 기존 기법보다 빠른 처리 속도를 나타낸다.

■ 중심어 : | 그래프 분석 | 그래프 스트림 | 빈발 패턴 검출 | 점진적 처리 | 슬라이딩 윈도우 |

Abstract

Recently, with the advancement of network technologies, and the activation of IoT and social network services, many graph stream data have been generated. As the relationship between objects in the graph streams changes dynamically, studies have been conducting to detect or analyze the change of the graph. In this paper, we propose a scheme to incrementally detect frequent patterns by using frequent patterns information detected in previous sliding windows. The proposed scheme calculates values that represent whether the frequent patterns detected in previous sliding windows will be frequent in how many future silding windows. By using the values, the proposed scheme reduces the overall amount of computation by performing only necessary calculations in the next sliding window. In addition, only the patterns that are connected between the patterns are recognized as one pattern, so that only the more significant patterns are detected. We conduct various performance evaluations in order to show the superiority of the proposed scheme. The proposed scheme is faster than existing similar scheme when the number of duplicated data is large.

■ keyword : | Graph Analysis | Graph Stream | Frequent Pattern Detection | Incremental Processing | Sliding Window |

*본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업 (IITP-2017-2013-0-00881), 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단 (No. 2016R1A2B3007527), 2014년도 산업통상자원부의 재원으로 한국에너지기술평가원(KETEP)의 에너지인력양성사업으로 지원받아 수행한 인력양성 성과입니다. (No. 20164030201330)

접수일자 : 2017년 10월 24일
심사완료일 : 2017년 11월 17일
수정일자 : 2017년 11월 17일
교신저자 : 유재수, e-mail : yjs@chungbuk.ac.kr

I. 서 론

그래프는 객체간의 복잡한 관계를 표현하기 위한 자료구조로 교통망, 생물정보학, 소셜 네트워크 등 다양한 분야에 이용된다[1-6]. 최근 네트워크 기술의 발전과 함께 IoT (Internet of Things), 소셜 네트워크 서비스 등에 대한 활성화로 인해 그래프 데이터가 실시간적으로 변화하는 스트림 데이터 형태로 변화되고 있다[7]. 이와 같이 실시간적으로 변화되는 그래프 데이터를 그래프 스트림 데이터라 한다. 그래프 스트림은 이상 감지, 실시간 트랜드 분석, 이벤트 검출 등 다양한 분야에 사용되고 있다. 예를 들어, 스마트 팩토리에서는 IoT 기반 센서 네트워크를 구축하여 센서간의 데이터 흐름을 그래프로 표현하고 공정 및 공급망 지능화, 이상 감지 등을 수행한다[8]. 그리고 소셜 네트워크 서비스에서도 사용자간의 관계 및 정보 소비 패턴을 그래프로 표현하고 인적 네트워크 분석 및 실시간 트랜드를 분석하는데 활용하고 있다.

그래프에서 객체들 사이의 관계가 동적으로 변화함에 따라 그래프의 변화를 탐지하거나 분석하기 위한 연구들이 진행되고 있다. 빈발 패턴 검출은 그래프 스트림에서 자주 사용되는 분석 기법 중 하나로 특정 기간 동안 자주 발생한 서브그래프를 검출하는 방법이다[9-12]. 예를 들어, 스마트 팩토리에서는 고장 예측을 위해 전조 현상들에 대한 패턴을 감지한다. 소셜 네트워크에서는 사용자간에 친밀 관계를 분석하기 위해서 사용자들 사이에 정보 교류 패턴을 판별한다[13].

최근 그래프 스트림에서 빈발 패턴 검출에 대한 활용이 증감됨에 따라 다양한 연구가 활발히 진행되고 있다[9-17]. [9]에서는 시간에 따라 입력되는 데이터의 양이 달라지는 것을 해결하기 위해 FP-Streaming (Frequent Pattern-Streaming)기법을 제안하였다. [10]에서는 그래프 스트림을 DSTree (Data Stream Tree)라는 구조를 제안함으로써, 빈발 패턴 검출을 할 때 효율적으로 데이터를 메모리에 저장한다. 그래프 스트림 데이터가 입력되면 DSTree를 구축한 후, 이 DSTree를 이용해 FP-Tree (Frequent Pattern Tree)를 구축하여 빈발 패턴을 검출한다. 하지만 DSTree를 구축할 때, 트리를 구성하기 위해 많은 포인터가 사용되고, 또 트리를 구축하는데 오랜 시간이 걸린다는 문제점이 있다. [11]에서는 [10]에서 제안한 DSTree 보다 더 효율적으로 그래프를 저장할 수 있는 DSMatrix (Data Stream Matrix)를 제안하였다. DSMatrix는 2차원 배열이기 때문에 DSTree보다 적은 비용으로 구축할 수 있다. 또한, 새로운 빈발 패턴 검출 기법을 통해 더 효과적으로 빈발 패턴을 찾는다. 하지만 슬라이딩 윈도우로 빈발 패턴을 검출할 때, 윈도우가 겹쳐서 처리하는 부분에서 중복으로 처리하여 성능이 저하된다. 또한, 이 두 개의 연구에서는 위에서 제시한 고려할 점 중에 세 번째인 연결성을 고려하지 않기 때문에, 서로 관련이 없는 두 개의 패턴을 하나의 패턴으로 보는 문제점이 있다. [12]는 이런 문제를 해결하기 위해 연결성을 고려한 빈발 패턴 검출 기법을 제안하였다. 이웃하는 간선 정보를 테이블로 관리함으로써 연결된 패턴만을 검출한다. 하지만 슬라이딩 윈도우로 인해 발생하는 중복 처리 문제를 여전히 해결하지 못했다.

본 논문에서는 실시간으로 입력되는 그래프 스트림 데이터에서 점진적으로 빈발 패턴을 검출하는 기법을 제안한다. 제안하는 기법은 윈도우가 이동할 때 이전 윈도우에서 분석한 결과를 재사용하여 연산량을 감소시킨다. 제안하는 기법은 입력되는 그래프 스트림을 DSMatrix에 저장 한 후, 간단한 AND 연산을 통해 빈발 패턴을 검출할 수 있다. 이 때, 검출된 패턴은 앞으로 몇 슬라이딩 윈도우동안 빈발할지 또는 빈발하지 않을지를 계산하여 따로 테이블에 관리한다. 이렇게 계산된 값을 통해 다음 슬라이딩 윈도우에서는 필요한 계산만 수행함으로써 전체 연산량을 감소시킨다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 본 논문과 관련된 연구들을 분석하고 Ⅲ장에서는 제안하는 빈발 패턴 검출 기법에 대해 설명한다. Ⅳ장에서는 제안하는 기법의 우수성을 확인하기 위해 성능 평가 결과를 보여준다. 마지막 Ⅴ장에서는 논문의 결과와 향후 연구 내용에 대해 기술한다.

II. 관련 연구

최근 그래프 스트림에서 빈발 패턴을 검출하기 위한 연구가 다양하게 진행되고 있다. 그래프 스트림에서 빈발 패턴을 검출하기 위한 첫 단계는 그래프를 분석하기 편리한 형태로 저장하는 것이다. 또한 그래프 스트림은 연속적으로 입력되기 때문에 빠르게 저장하고 적은 연산으로 다음 데이터를 입력받을 수 있어야한다. [10]에서는 그래프 스트림을 메모리에 저장하기 위한 DSTree라는 구조를 제안한다. 그래프 스트림이 입력되면, 각 엣지를 순서대로 정렬한 후 트리를 구축한다. 이 DSTree의 각 노드는 간선에 대한 정보를 배치별로 유지하기 때문에, 윈도우 슬라이드가 이동했을 때에도 해당 정보만 바꿔주면서 DSTree를 유지한다. [11]에서는 DSMatrix라는 구조를 제안한다. DSMatrix는 2차원 배열로 그래프의 각 간선에 대해 발생했는지 여부를 1 또는 0으로 표현하여 적은 메모리 공간에 그래프 스트림 데이터를 저장할 수 있다.

빈발 패턴을 검출하기 위한 방법은 크게 유사 알고리즘과 정확성 알고리즘이 있다. 유사 알고리즘은 정확성보다 빠르게 검출하는 것을 목적으로 하는 기법으로 대표적으로 FP-streaming 알고리즘[9]이 있다. 이 알고리즘에서는 그래프 스트림 데이터가 매시간마다 입력데이터의 양이 달라지는 것을 해결하기 위해 입력 시간별로 빈발 패턴을 검출한다.

정확성 알고리즘은 모든 빈발 패턴을 정확하게 검출해 내는 것을 목적으로 한다. 따라서 유사 알고리즘에 비해 느리게 검출하지만 정확한 빈발 패턴만을 검출한다. [11]에서는 두 가지 정확성 빈발 패턴 검출 기법을 제안한다. 첫 번째는 그래프 스트림을 DSMatrix에 저장한 후, 모든 간선에 대해 FPTree를 구축한다. 이렇게 구축된 FPTree에서 재귀적으로 FPTree를 구축함으로써, 모든 빈발 패턴 검출이 가능하다. 이 기법은 여러 FPTree를 구축함으로써 원하는 데이터를 추출하기 쉽다. 두 번째 기법은 각 간선에 대해 FPTree를 구축한 후, 깊이 우선 탐색 방법으로 각 노드를 방문하며 빈발 패턴의 발생 횟수를 세는 방법이다. 이 기법은 각 간선별로만 FPTree를 구축하기 때문에 더 적은 비용으로 빈발 패턴 검출이 가능하다. [12]에서는 빈발 패턴 검출을 할 때 FPTree를 구축하지 않고 간단한 AND연산을 통해 빈발 패턴을 찾아내는 기법을 제안한다. 또한 간선간의 연결성을 고려하여 더 의미있는 빈발 패턴을 검출하는 기법을 제안하였다. 여기서는 두가지 빈발 패턴 검출 기법을 제안한다. 첫 번째 기법은 모든 빈발 패턴을 AND연산을 통해 검출한 후, 연결되지 않은 패턴을 제외시키는 방법이다. 그리고 두 번째 기법은 AND연산을 하기 전에 서로 연결 여부를 확인하고 연산을 수행한다.

[9]에서는 시간별로 빈발 패턴을 검출하지만 그래프 스트림에서 빈발 패턴을 검출할 때에는 이전 시간에서 빈발 유무가 영향을 미친다. 그렇기 때문에 빈발 패턴을 정확히 검출할 수 없다. [10]에서는 그래프의 구조가 크게 변할 경우 DSTree 구조가 바뀌되기 때문에 DSTree를 재구축하는데 많은 시간이 소요된다는 단점이 있다. 또한 DSTree를 구축할 때 많은 포인터를 사용하게 됨으로써 메모리에서 이 트리를 유지하기 위한 관리비용이 커진다는 문제점이 있다. 이러한 문제를 해결하기 위해 [11]에서는 DSMatrix라는 2차원 배열을 제안하여 효과적으로 해결한다. 하지만 빈발 패턴 검출 과정에서 많은 FPTree를 구축하면서 많은 연산과 메모리가 필요하다. 또한 [11][12]은 슬라이딩 윈도우 기법을 이용하여 발생하는 문제점인 중복 계산을 해결하지 못하며 성능 저하가 발생한다.

Ⅲ. 제안하는 점진적 빈발 패턴 검출 기법

1. 제안하는 기법의 특징

그래프 스트림 데이터에서 빈발 패턴을 검출할 때 세 가지 고려할 점이 있다. 첫 번째는 한정적인 저장 공간을 효율적으로 사용하고, 빠르게 빈발 패턴 검출을 해야 한다. 스트림 데이터는 연속적으로 입력되며, 끝이 어디인지 알 수 없기 때문에, 분석해야할 데이터가 실시간으로 변화된다. 따라서 어느 정도 데이터가 입력되었을 때, 분석한 후 삭제하여 메모리를 확보해야 다음에 입력될 데이터에서 분석이 가능하다. 두 번째는 시간에 따라 입력되는 데이터가 다르다는 점이다. 즉, 현재의 빈발 패턴이 나중엔 빈발 패턴이 아닐 수 있고, 그

반대의 경우도 발생한다. 마지막으론 그래프의 연결성을 고려하여 패턴을 검출해야한다. 여기서 연결성이란 검출된 패턴들이 서로 연결되어 있는 것을 말한다. 예를 들어, [그림 1]에서 패턴 A, B, F가 검출되었을 때, A와 B는 연결되었기 때문에 패턴 AB라고 볼 수 있고, F는 연결되어 있지 않기 때문에 독립적인 패턴이다.

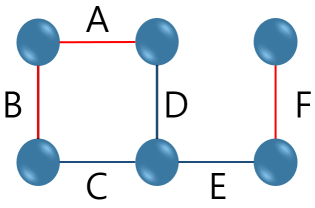


그림 1. 연결성을 고려한 패턴

본 논문에서는 한정적인 저장 공간을 효율적으로 사용하기 위해 DSMatrix를 사용한다. DSMatrix는 boolean 형식의 2차원 배열로 작은 크기로 많은 그래프를 저장할 수 있다. 그리고 빈발 패턴을 검출할 때, 간단한 AND 연산을 이용하므로 빠르게 빈발 패턴 검출이 가능하다. 그리고 제안하는 기법은 시간에 따라 입력되는 데이터가 다른 특성을 고려하기 위해 슬라이딩 윈도우를 사용한다. 이 방법을 이용함으로써 여러 시간 동안의 그래프의 변화를 고려한 빈발 패턴 검출이 가능하다. 또한 간단한 연산을 통해 연결성을 고려하여 더 의미 있는 패턴을 검출한다.

제안하는 기법의 전체 처리 과정은 그림 2와 같다. 전처리 단계는 그래프 스트림이 입력될 때, 입력된 데이터를 메모리에 효율적으로 저장하기 위한 단계이다. 이 때 DSMatrix라는 2차원 배열을 이용하여 적은 공간에 많은 그래프 데이터를 저장할 수 있다. 빈발 패턴 검출 단계에서는 실제 빈발 패턴을 검출하기 위한 연산을 수행한다. 빈발 패턴을 검출할 때에는 패턴의 발생 횟수를 합하여 빈발 여부를 확인하고, 발생된 두 패턴을 AND 연산을 하여 다시 발생횟수를 합함으로써, 여러 간선으로 이루어진 빈발 패턴 검출이 가능하다. 이 때, 빈발 패턴 관리 테이블을 통해 이전 윈도우 슬라이드에서 검출한 빈발 패턴에 대한 정보를 이용한다. 빈발 패턴 관리 테이블에는 기존에 검출된 패턴이 앞으로 몇 슬라이딩 윈도우동안 빈발할지 또는 빈발하지 않을지를 계산하여 저장하고, 이 값을 통해 다음 슬라이딩 윈도우에서는 필요한 계산만 수행함으로써 전체 연산량을 감소시킨다. 마지막으로 검출한 빈발 패턴을 사용자에게 전달함과 동시에 다시 빈발 패턴 관리 테이블에 저장하여, 다음 윈도우 슬라이드에서 활용한다.

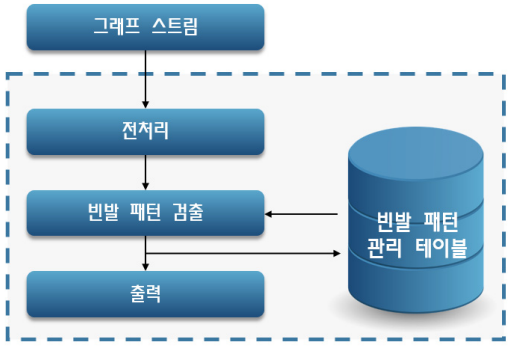


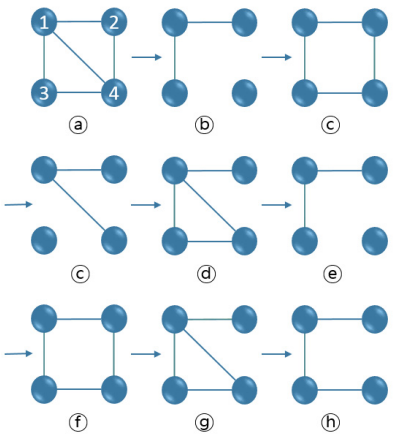
그림 2. 전체 처리 과정

2. 전처리

전처리는 그래프 스트림이 입력되면 메모리에 저장하는 단계이다. 그래프 스트림 데이터는 연속적으로 입력되며, 끝이 어디인지 알 수 없기 때문에, 분석해야할 데이터가 실시간으로 변화된다. 따라서 어느 정도 데이터가 입력되었을 때, 분석한 후 삭제하여 메모리를 확보해야 다음에 입력될 데이터에서 분석이 가능하다. 이러한 특성을 고려하여, 제안하는 기법의 전처리 단계는 DSMatrix이라는 2차원 배열구조를 이용한다. DSMatrix는 boolean 형식의 배열이기 때문에 기존 기법[10]에서 사용한 DSTree보다 적은 공간에 많은 데이터를 저장할 수 있다. 그리고 그래프에서 간선의 유무를 1, 0으로 표현하기 때문에 빠른 추가, 삭제가 가능하여 슬라이딩 윈도우 방식에 적합하다.

DSMatrix는 하나의 윈도우 슬라이드는 사용자가 정한 수만큼의 배치로 이루어진다. [그림 3](a)와 같이 그래프가 입력될 때, DSMatrix는 [그림 3](b)와 같이 구성된다. 여기서 1개 배치는 3개 그래프이고 3개의 배치는 1개 슬라이딩 윈도우로 이루어져있다. 각 간선은 연결된 두 개의 정점의 이름으로 표현한다. 즉, [그림 3](a)에서 정점 1과 정점 2에 연결된 간선은 [그림 3](b)

의 1열처럼 1/2로 표현한다. 이렇게 표현된 간선들은] 그림 3](b)의 contents 부분에 발생유무를 1 또는 0으로 표현한다. 예를 들어, [그림 3](a)에서 ⓒ는 1/2, 1/3, 2/4, 3/4를 가진 그래프인데, 이 때 DSMatrix는 4.(b)의 4번째 열로 표현할 수 있다. 이렇게 구성된 DSMatrix를 이용해 다음 단계에서 빈발 패턴을 검출한다. 그리고 검출 후에 새로운 슬라이딩 윈도우가 입력되면, 가장 앞의 배치를 삭제하고 뒤에 새로운 배치를 추가하는 방법으로 빠르게 다음 슬라이딩 윈도우의 그래프들을 저장한다.



(a) 그래프 스트림 데이터

row	Contents			Contents			Contents		
1/2	1	1	1	1	1	1	1	1	1
1/3	1	1	1	0	1	1	1	1	1
1/4	1	0	0	1	1	0	0	1	0
2/4	1	0	1	0	0	0	1	0	0
3/4	1	0	1	0	1	0	1	1	1

(b) DSMatrix

그림 3. DSMatrix 생성 과정

3. 빈발 패턴 검출

3.1 기본적인 빈발 패턴 검출

빈발 패턴을 검출할 때, 제안하는 기법은 두 가지를 고려하였다. 첫째, 스트림 데이터는 연속적으로 입력되기 때문에 어느 정도 데이터가 입력되었을 때, 분석한 후 삭제하여 메모리를 확보해야 다음에 입력될 데이터에서 분석이 가능하다. 이를 위해 빠르게 빈발 패턴을 검출하는 방법이 요구된다. 본 논문에서는 패턴의 발생 횟수를 합하여 빈발 여부를 확인하고 발생된 두 패턴을 AND 연산을 하여 다시 발생 횟수를 합하여 연속적으로 여러 간선으로 이루어진 빈발 패턴 검출한다. 여기서 AND연산과 발생횟수를 합하는 연산은 간단한 연산이기 때문에 빠른 패턴 검출이 가능하다. 두 번째로는 연결성을 고려하였다. 만약 두 개의 패턴이 떨어져있다면 이 두 패턴이 동시에 발생해도 하나의 패턴으로 보기 어렵다. 따라서 제안하는 기법에서는 연결성을 고려하여 더 의미 있는 빈발 패턴을 검출한다.

제안하는 기법에서 빈발 패턴을 검출하기 위해서는 먼저 단일 간선에 대해 빈발 여부를 확인한다. 전처리 단계에서 만들어진 DSMatrix에서 각 간선 별로 하나의 배치에서의 발생 횟수를 계산하여 [표 1]의 빈발 패턴 관리 테이블의 FiB (Frequency in Batch)에 입력한다. FiB는 하나의 배치에서 해당 간선이 발생한 횟수이다. 그리고 FiS (Frequency In Sliding window)는 앞에서 계산한 FiB를 모두 더한 값으로, 해당 간선이 현재 슬라이딩 윈도우에서 빈발했는지를 확인 할 수 있다. 예를 들어 [표 1]의 1/2간선은 하나의 배치마다 3회씩 발생하였고, 이번 윈도우 슬라이드에서 총 9번을 발생하였다. 여기서 만약 τ=5라면 1/2 간선은 9회 발생했기 때문에 빈발 패턴이다.

표 1. 빈발 패턴 관리 테이블

row	SlideNum	FiB	FiB	FiB	FiS
1/2	1	3	3	3	9
1/3	1	3	2	3	8
1/4	0	1	2	1	4
2/4	-1	2	0	1	3
3/4	0	2	1	3	6

단일 간선이 다음 몇 번의 슬라이드 동안 빈발할지 혹은 빈발하지 않을지를 계산하여 slideNum에 입력한다. 예를 들어, [표 1]에서 다음 윈도우 슬라이드에서 1/2간선이 모두 0이 입력되었다고 해도, 빈발 횟수는 여전히 6으로 빈발하는 간선이다. 하지만 그 다음 윈도우

슬라이드에서 다시 모두 0이 입력되면 총 3번 발생하였으므로 빈발 간선이 아니다. 따라서 slideNum은 1이 된다. 이 slideNum을 계산하는 방법은 [그림 4]와 같다. slideNum은 FiS가 threshold 보다 클 때와 작을 때의 경우로 나누어 계산한다. 먼저 클 때에는 BatchCount에 현재 슬라이딩 윈도우에서 가장 처음 배치를 삭제했을 때 남은 수를 계산하고 그 다음 새로 입력되는 배치가 모두 0이라고 가정하여 threshold를 넘는지 확인한다. 이 과정을 반복하여 몇 회 반복했을 때 BatchCount가 threshold보다 작아지는지 계산하여 결과를 반환한다. 만약 FiS가 threshold보다 작다면, 다음에 입력될 배치가 모두 1이라고 가정하여 threshold를 넘지 않는지 확인하고, 횟수를 반환한다. 빈발하는 단일 간선을 모두 검출한 후에는 여러 간선들로 이루어진 패턴을 검출한다. 검출된 두 개의 패턴에 AND 연산을 하면, 두 패턴이 동시에 발생한 빈발 횟수를 알 수 있다. 이를 이용하여 계속적으로 패턴을 확장한다. 즉, 두 개의 간선으로 이루어진 패턴을 검출한다면 단일 빈발 간선을 이용한다. 앞의 예제에서 단일 간선 패턴 중 1/2와 1/3으로 AND 연산을 하게 되면 111;011;111 이므로 발생 횟수는 8이 되기 때문에 1/2/3 역시 빈발하는 패턴임을 알 수 있다.

Algorithm 1. slideNum

```
Input :
 FiB[]  - array of FiB which is the number of edge
          in a batch
 FiS  - The number of the edge in sliding window
 th - threshold
Output : slideNum
slideNum ← 0
if FiS >= th then
    BatchCount ← FiS - FiB[0]
    while BatchCount >= th and
    slideNum < SlidingWindowSize do
        BatchCount ← BatchCount - FiB[slideNum+1]
        slideNum ← slideNum+1
    end while
    return slideNum
else
    BatchCount ← FiS - FiB[0]+BatchSize
    while BatchCount < th and
    slideNum < SlidingWindowSize do
        BatchCount ← BatchCount - B[SlideNum+1]
                    + BatchSize
        slideNum ← slideNum + 1
    end while
    return -slideNum
end if
```

그림 4. slideNum 계산 알고리즘

추가적으로 여러 간선으로 이루어진 패턴을 검출할 때에는 연결성을 고려해야 한다. 두 개의 패턴이 서로 연결되어 있는지를 판단할 때에는 간선 이름을 이용한다. 즉, 패턴의 이름에 중복되는 정점의 id가 있다면 두 패턴은 서로 연결되어 있는 것을 알 수 있다. 만일 두 개의 패턴이 서로 연결되어 있지 않으면, 빈발 패턴에 포함시키지 않는다. 즉, 예제에서 1/2와 3/4는 중복되는 정점 id가 없기 때문에 AND 연산을 수행하지 않는다. 이렇게 검출한 빈발 패턴은 [표 2]과 같이 빈발 패턴 관리 테이블에 저장한다. 여기서 FiB, FiS, SlideNum은 단일 간선에서의 계산 방법과 같다.

표 2. 패턴이 추가된 빈발 패턴 관리 테이블

row	SlideNum	FiB	FiB	FiB	FiS
1/2	1	3	3	3	9
1/3	1	3	2	3	8
1/4	0	1	2	1	4
2/4	–1	2	0	1	3
3/4	0	2	1	3	6
1/2/3	1	3	2	3	8
1/3/4	0	2	1	2	5

3.2 점진적 빈발 패턴 검출

본 논문에서 빈발 패턴을 검출할 때 슬라이딩 윈도우 방식을 사용한다. 하지만 슬라이딩 윈도우는 데이터가 중복되기 때문에 [11][12]과 같이 성능저하가 발생한다. 이 문제를 해결하기 위해 이전에 검출하였던 빈발패턴

정보를 빈발 패턴 관리테이블에 저장하고, 이를 활용하여 새로운 빈발패턴을 검출하는 방법을 제안한다. 빈발 패턴 관리 테이블에는 기존에 검출된 패턴이 앞으로 몇 슬라이딩 윈도우동안 빈발할지 또는 빈발하지 않을지를 계산하여 저장하고, 이 값을 통해 다음 슬라이딩 윈도우에서는 필요한 계산만 수행함으로써 전체 연산량을 감소시킨다.

[그림 5]과 같이 새로운 그래프가 입력되었을 때, DSMatrix는 [그림 6]의 빨간 상자 부분의 배치가 추가된다. 새 배치가 추가되면 기본적인 빈발 패턴 검출과 같이 FiB와 FiS를 계산하여 입력한다. 하지만 slideNum이 0이 아니라면 slideNum을 감소시킨 후, FiB와 FiS를 계산하지 않는다. 즉, [표 3]과 같이 1/2, 1/3와 2/4는 FiB와 FiS를 계산하지 않는다.

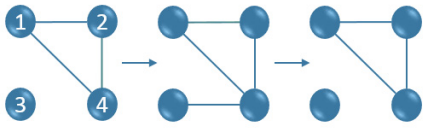


그림 5. 새로 입력된 그래프 스트림

row	Contents			Contents			Contents		
1/2	1	1	1	1	1	1	1	1	1
1/3	0	1	1	1	1	1	0	0	0
1/4	1	1	0	0	1	0	1	1	1
2/4	0	0	0	1	0	0	1	1	1
3/4	0	1	0	1	1	1	0	1	0

그림 6. 윈도우 슬라이드가 이동한 후 DSMatrix

여러 간선들로 이루어진 빈발 패턴을 검출할 때에도 앞의 과정과 유사한 방법을 이용한다. [표 3]에서 1/2/3 패턴은 slideNum이 1이었기 때문에 FiB, FiS는 계산하지 않아도 빈발 패턴이라는 것을 알 수 있다. 그러므로 slideNum을 0으로 변경한 후 빈발 패턴으로 유지한다. 1/3/4 패턴 같은 경우에는 slideNum=0 이었기 때문에 1/3과 3/4의 새로 입력된 배치 부분만 AND 연산을 하여 FiB와 FiS를 계산한다. 즉, 1/3은 000, 3/4는 010이므로 000 & 010 = 000이 되고, 따라서 새로 추가된 부분의 FiB=0, FiS=3이 된다. 이 때, 만약 FiS가 threshold 값보다 크다면 slideNum을 계산한다.

표 3. 윈도우 슬라이드가 이동한 후 빈발 패턴 검출

row	SlideNum	FiB	FiB	FiB	FiS
1/2	1 → 0	3	3		
1/3	1 → 0	2	3		
1/4	0	2	1	3	6
2/4	-1 → 0	0	1		
3/4	0	1	3	1	5
1/2/3	1 → 0	2	3		
1/3/4	0	1	2	0	3

Ⅳ. 실험 평가

본 논문에서는 제안하는 기법의 우수성을 보이기 기존 기법[11][12]과의 성능평가를 수행하였다. [표 4]는 성능평가 환경을 보여준다. 임의의 그래프 스트림 데이터를 생성하고 배치의 크기, 윈도우 슬라이드의 크기, threshold 값 등 여러 가지 경우에 대해 평가하였다. 성능평가 환경은 Intel(R) Core(TM) i5-4440 3.10GHz 프로세서, 8G 메모리, 1TB디스크 환경으로 구성되고 Java로 구현하여 성능평가를 수행하였다.

표 4. 성능 평가 환경

구분	내용
프로세서	Intel(R) Core(TM) i5-4440 3.10GHz
메모리	8 GB
디스크	1 TB
프로그램 언어	Java

[그림 7]은 데이터 크기, 즉 간선의 수가 증가함에 따라 데이터의 처리 속도를 기존 기법과 비교한 실험 결과이다. 이 실험에서는 그래프의 간선의 개수를 변화시키며 성능평가를 하였고, 배치는 100개의 그래프로 구성되며, 윈도우 슬라이드는 5개의 배치로 이루어져있

다. 또한 Threshold는 80%로 윈도우 슬라이드에서 400회 이상 등장한 패턴을 검출한다. 이 실험에서 간선의 수가 적을 때에는 처리 시간이 크게 차이 나지 않지만, 간선의 수가 증가할수록 제안하는 기법의 처리 시간이 최대 60%까지 줄어드는 것을 볼 수 있다. 이러한 이유는 간선의 수가 증가함에 따라 중복되는 처리결과도 같이 증가하는데 기존 기법은 이런 중복되는 처리를 계속 수행하기 때문이다.

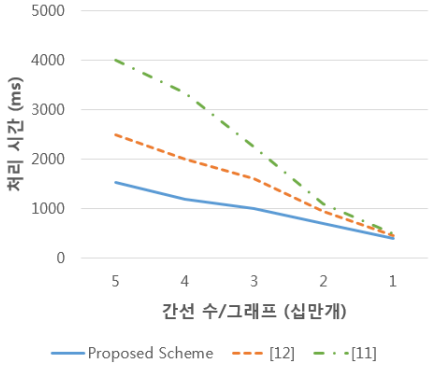


그림 7. 간선 수에 따른 처리 시간

[그림 8]은 윈도우 슬라이드의 크기에 따른 제안 기법의 처리 속도의 차이를 보여준다. 각 그래프는 30만개의 간선으로 구성된다. 그림에서 하나의 슬라이딩 윈도우에 배치가 20개가 있을 때, [11]의 처리시간의 63%, [12]의 처리시간의 50%가 감소하는 것을 볼 수 있다. 이 결과를 통해 슬라이딩 윈도우의 개수가 증가할수록 제안하는 기법의 성능이 좋아진다는 것을 확인할 수 있다. 그 이유는 점진적 빈발 패턴 검출 기법을 사용할 때, slideNum의 값에 따라 앞으로 몇 번의 입력동안 계산하지 않아도 되는지가 결정되기 때문이다. 그렇기 때문에 슬라이딩 윈도우의 크기가 커지면 계산하지 않아도 되는 패턴이 증가하게 됨으로써 전체 연산량이 감소하게 된다.

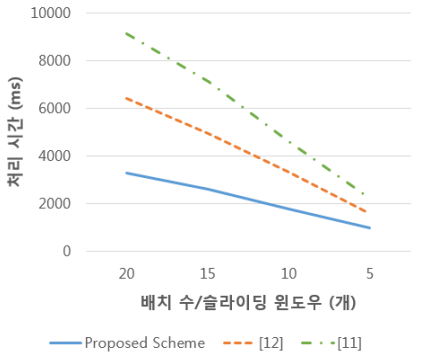


그림 8. 슬라이딩 윈도우 크기에 따른 처리시간

[그림 9]는 threshold값에 따른 처리 시간을 보여준다. 이 성능평가에서 그래프는 30만개로 이루어져 있고 1개 배치는 100개 그래프, 1개 슬라이딩 윈도우는 5개의 배치로 구성되어있다. 그림에서 보다시피 threshold가 작아지면 처리속도가 급격하게 감소한다. 그 이유는 두 개 이상의 빈발 패턴을 검출할 때 고려해야 하는 경우의 수가 지수함수 형태로 증가하기 때문이다. 하지만 threshold가 80% 일 때에는 [12]에 비해 약 60%, [11]에 비해 55%의 처리속도를 보이며 제안하는 기법의 처리속도가 빨라진다. 그 이유는 threshold 값 역시 slideNum의 영향을 미치기 때문이다. threshold 값이 클수록 slideNum이 커질 가능성이 높아지므로, 성능의 향상을 보인다. 따라서 데이터와 응용 분야에 맞는 적절한 threshold값을 선택하는 것이 중요하다.

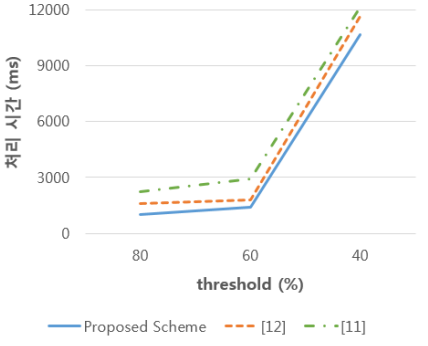


그림 9. threshold에 따른 처리시간

V. 결론

본 논문에서는 그래프 스트림에서 빈발 패턴 검출을 위한 점진적인 처리 기법을 제안하였다. 이렇게 함으로써 기존의 기법보다 빠르게 빈발 패턴 검출이 가능하기 때문에 더 많은 양의 데이터에 대해 분석이 가능하다. 이런 빈발 패턴 검출은 센서 네트워크에서 이상감지, 소셜 네트워크에서 사용자간의 관계 및 정보를 이용한 인적 네트워크 분석 및 실시간 트랜드를 분석 등 다양한 분야에서 활용할 수 있다. 제안하는 기법은 이전 슬라이딩 윈도우에서 발견한 빈발 패턴들을 빈발 패턴 관리 테이블에 관리하여 다음 빈발 패턴 검출할 때 활용함으로써 처리속도를 감소시킨다. 또한 연결성을 고려하여 유의미한 빈발 패턴을 검출하였다. 성능평가 결과 제안하는 기법은 그래프의 크기가 커지고, 슬라이딩 윈도우의 크기가 커질수록 슬라이딩 윈도우에서 중복된 부분이 많아지기 때문에 이런 중복 연산을 감소시킴으로써 기존 기법들보다 평균 55%정도 처리 시간이 감소하는 것을 확인하였다. 하지만 threshold 값이 작아지면서 빈발 패턴의 수가 지수 형태로 증가하게 되고, 그 결과로 성능이 저하되는 것이 발견되었다. 본 논문에서는 빈발 패턴을 테이블에 관리하기 때문에 빈발 패턴의 수가 많아질 때, 이를 관리하기 위하여 많은 메모리 공간이 필요하고, 스캔하는데 많은 시간이 소요된다는 한계가 있다. 따라서 향후 연구로 관리해야하는 패턴의 수가 증가되었을 때 인덱스를 이용하여 스캔하는 비용을 줄이고, 필요한 패턴에 직접 접근하기 위한 인덱스를 사용하는 기법에 대해 연구할 예정이다. 또한 임의로 생성한 데이터를 사용하여 실험평가를 진행하였기 때문에, 데이터의 신뢰도를 향상시키기 위하여 실제 데이터를 이용하여 더 다양한 실험 평가를 진행할 예정이다.

참 고 문 헌

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

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

[3] A. Cuzzocrea, F. Furfaro, G. M. Mazzeo, and D. Saccà, “A grid framework for approximate aggregate query answering on summarized sensor network readings,” Proc. OTM Workshops, pp.144-153, 2004.

[4] A. Fariha, C. F. Ahmed, C. K. Leung, S. M. Abdullah, and L. Cao, “Mining frequent patterns from human interactions in meetings using directed acyclic graphs,” Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, pp.38-49, 2013.

[5] F. Jiang and C. K. Leung, “Mining interesting “following” patterns from social networks,” Proc. International Conference on Data Warehousing and Knowledge Discovery, Springer, pp.308-319, 2014.

[6] S. K. Tanbeer, F. Jiang, C. K. Leung, R. K. MacKinnon, and I. J. M. Medina, “Finding groups of friends who are significant across multiple domains in social networks,” Proc. International Conference on Computational Aspects of Social Networks, pp.21-26, 2013.

[7] 한진수, 조중권, 최도진, 임종태, 복경수, 유재수, “부하 분산을 위한 정점 절단 기반의 그래프 스트림 분할 기법,” 한국정보과학회 학술발표논문, pp.206-208, 2017.

[8] 강필성, “사물인터넷과 빅데이터 분석 기반의 스마트공장 구현 사례 및 시사점,” 한국정보화진흥원, Near & Future, Vol.20, pp.25-35, 2016.

[9] 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, pp.191-212, 2003.

[10] C. K. Leung and Q. I. Khan, “DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams,” Proc. International Conference

on Data Mining, pp.928-932, 2006.

[11] P. Braun, J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Effectively and Efficiently Mining Frequent Patterns from Dense Graph Streams on Disk," Proc. International Conference in Knowledge Based and Intelligent Information and Engineering Systems, pp.338-347, 2014.

[12] A. Cuzzocrea, Z. Han, F. Jiang, C. K. Leung, and H. Zhang, "Edge-based Mining of Frequent Subgraphs from Graph Streams," International Conference in Knowledge Based and Intelligent Information and Engineering Systems, pp.573-582, 2015.

[13] S. K. Tanbeer, C. K. Leung, and J. J. Cameron, "Interactive Mining of Strong Friends from Social Networks and its Applications in E-Commerce," Journal of Organizational Computing and Electronic Commerce, Vol.24, No.2-3, pp.157-173, 2014.

[14] 서복일, 김재인, 황부현, "스트림 데이터 환경에서 배치 가중치를 이용하여 사용자 특성을 반영한 빈발항목 집합 탐사," 한국콘텐츠학회논문지, 제11권, 제1호, pp.56-64, 2011.

[15] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin, "On dense pattern mining in graph streams," Proceedings of the VLDB Endowment, Vol.3, No.1-2, pp.975-984, 2010.

[16] A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, "Mining frequent closed graphs on evolving data streams," Proc. ACM SIGKDD international conference on Knowledge discovery and data mining, pp.591-599, 2011.

[17] E. Valari, M. Kontaki, and A. N. Papadopoulos, "Discovery of top-k dense subgraphs in dynamic graph collections," Proc. International Conference on Scientific and Statistical Database Management, Springer, pp.213-230, 2012.

저 자 소 개

정 재 윤(Jaeyun Jeong) 준회원

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

<관심분야> : 그래프 분석, 빈발 패턴 마이닝, 데이터베이스 시스템, 분산 컴퓨팅 등

서 인 덕(Indeok Seo) 준회원

- 2016년 2월 : 청주대학교 통계학과(이학사)
- 2016년 3월 ~ 현재 : 충북대학교 빅데이터협동과정(석사과정)

<관심분야> : 소셜 네트워크, 빅데이터, 신뢰도 분석

송 희 섭(Heesub Song) 준회원

- 2016년 2월 : 한림대학교 전자물리학과(이학사)
- 2016년 3월 ~ 현재 : 충북대학교 빅데이터 협동과정(석사과정)

<관심분야> : 소셜 미디어, 소셜 네트워크, 빅데이터, 데이터베이스 시스템 등

박 재 열(Jaeyeol Park) 준회원

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

<관심분야> : 데이터베이스 시스템, RDF, 실체화 뷰, 빅데이터 등

김 민 영(Minyeong Kim) 준회원

- 2017년 2월 : 충북대학교 정보통신공학과(공학사)
- 2017년 3월 ~ 현재 : 충북대학교 정보통신공학과(석사과정)

<관심분야> : 데이터베이스 시스템, 분산 컴퓨팅, 그래프 분석

최 도 진(Dojin Choi) 정회원

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

<관심분야> : 연속 질의 처리, 그래프 스트림

복 경 수(Kyungsoo Bok) 종신회원

- 1998년 2월 : 충북대학교 수학과(이학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2005년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2005년 3월 ~ 2008년 2월 : 한국과학기술원 전산학과 Postdoc
- 2008년 3월 ~ 2011년 2월 : ㈜가인정보기술 연구소
- 2011년 3월 ~ 현재 : 충북대학교 정보통신공학과 초빙교수

<관심분야> : 데이터베이스 시스템, 이동객체 데이터베이스, 소셜네트워크, 빅데이터 등

유 재 수(Jaesoo Yoo) 종신회원

- 1989년 2월 : 전북대학교 컴퓨터공학과(공학사)
- 1991년 2월 : KAIST 전산학과(공학석사)
- 1995년 2월 : KAIST 전산학과(공학박사)
- 1995년 3월 ~ 1996년 8월 : 목포대학교 전산통계학과(전임강사)
- 1996년 8월 ~ 현재 : 충북대학교 정보통신공학부 및 컴퓨터정보통신연구소 교수

<관심분야> : 데이터베이스 시스템, 분산시스템, 바이오정보처리, 이동객체 데이터베이스, 소셜네트워크, 빅데이터 등