DOI: 10.3745/KIPSTD.2009.16D.5.673

빈발 패턴 트리 기반 XML 스트림 마이닝

황 정 희†

요 약

웹상에서 데이터 교환과 표현을 위한 표준으로 XML 데이터가 널리 사용되고 있으며 유비쿼터스 환경에서 XML 데이터의 형태는 연속적이다. 이와 관련하여 XML 스트림 데이터에 대한 빈발 구조 추출 및 효율적인 질의처리를 위한 마이닝 방법들이 연구되고 있다. 이 논문에서는 슬라이딩 윈도우 기반으로 하여 XML 스트림 데이터로부터 최근 윈도우 범위에 속하는 데이터에 대한 빈발 패턴 구조를 추출하기 위한 마이닝 방법을 제안한다. 제안된 방법은 XML 스트림 데이터를 트리집합 모델, XFP_tree로 표현하고 이를 이용하여 최근의 데이터에 대한 빈발구조 패턴을 빠르게 추출한다.

키워드 : XML 스트림, XML 마이닝, XML 데이터, 데이터 마이닝

Frequent Patten Tree based XML Stream Mining

Jeong Hee Hwang†

ABSTRACT

XML data are widely used for data representation and exchange on the Web and the data type is an continuous stream in ubiquitous environment. Therefore there are some mining researches related to the extracting of frequent structures and the efficient query processing of XML stream data. In this paper, we propose a mining method to extract frequent structures of XML stream data in recent window based on the sliding window. XML stream data are modeled as a tree set, called XFP_tree and we quickly extract the frequent structures over recent XML data in the XFP_tree.

Keywords : XML Stream, XML Mining, XML Data, Data Mining

1. 서 론

유비쿼터스 환경에서 상황정보 인식 분야를 연구하면서 가장 밑바탕에서 기초가 될 수 있는 것이 인터넷 기술과 XML(Extensible Markup Language)이다. 최근 인터넷의 급격한 발전과 유비쿼터스 컴퓨팅 환경(ubiquitous computing environment) 그리고 센서 네트워크와 같은 많은 정보들의 교환이 이루어지는 환경에서 무한의 연속적으로 전송되는 데이터에 대한 처리가 요구되고 있다. 이러한 무한의 연속 데이터를 스트리밍 데이터라고 한다[1-4]. 그리고 XML은 어떤 서비스든지 그 서비스를 기술하는 방법을 제공하는 유연성이 있으므로 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있다. 현재 유비쿼터스 환경을 위한 기초 데이터로써 XML 스트림 데이터(stream data)의 사용이 일반화되고 있다[4, 7, 8].

전자상거래 사이트의 웹 로그, 네트워크의 트래픽을 감시하는 모니터링 시스템, 유비쿼터스 환경 등에서 사용하는 센서 네트워크를 통해 수집 가능한 스트림 데이터는 매우 빠른 속도로 끊임없이 지속적으로 발생하며 그 양이 방대하다는 특성을 갖는다. 스트림 데이터는 비즈니스 데이터와는 특성이 크게 달라서 스트림 데이터의 처리를 위해 기존의 데이터 저장, 분석 기술을 다음과 같은 특성으로 인해 그대로 사용하기는 어렵다. 첫째, 스트림의 모든 데이터 요소는 한 번에 조사되어야 한다. 둘째, 지속적으로 발생하는 데이터라도 처리할 수 있는 메모리 사용량은 한정되어 있다. 셋째, 스트림 데이터는 가능한 빠르게 처리되어야 한다. 넷째, 데이터 처리 결과에 대한 에러는 가능한 최소화해야 한다. 이러한 스트림 데이터의 처리 요구사항을 만족시키기 위한 데이터 마이닝의 새로운 연구는 스트림 데이터 마이닝이다[2-4, 7].

최근 데이터 스트림에 대한 빈발 패턴을 탐색하기 위한 마이닝 방법들[2, 9, 11]이 제안되고 있다. 그러나 구조적인

※ 이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-331-D00486)
† 종신회원 : 남서울대학교 컴퓨터학과 전임강사
논문접수 : 2009년 2월 4일
수 정 일 : 1차 2009년 4월 21일, 2차 2009년 5월 22일
심사완료 : 2009년 6월 5일

데이터 특성을 갖는 스트리밍 XML 데이터를 마이닝 하기 위한 연구는 미진한 상태이다. 유비쿼터스 환경 및 웹 데이터의 표준인 XML의 사용은 일반적인 기준이 되고 있으므로 기존의 XML 데이터로부터 빈발 구조를 탐색하는 방법[5, 6, 10]을 기반으로 스트리밍 XML 데이터의 특성을 고려하는 빈발 구조의 탐색방법에 대한 연구가 필요하다[4, 8, 9]. 스트리밍 XML 데이터에 대한 빈발 구조 탐색은 스트림 데이터에 대한 효율적인 질의 처리 및 색인구성에 효율적으로 이용될 수 있다.

이 논문에서는 스트리밍 XML 데이터로부터 빈발하게 발생하는 구조를 효율적으로 탐색하기 위하여 슬라이딩 윈도우 기법을 기반으로 하여 현재 윈도우 범위에 속하는 XML 데이터 집합을 트리 모델, XFP_tree로 구성하고 이를 이용한다. XFP_tree는 각 XML 데이터로부터 일정 임계값 이상의 빈발 구조 집합을 표현하는 트리 구조이고, 윈도우 이동 시점에서 삭제되는 트랜잭션의 XML 트리에 해당하는 노드들을 일괄적인 이동에 의해 빈발도를 갱신하므로 현재 윈도우의 최신 데이터에 집중하여 빈발 구조를 탐색할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 스트림 마이닝과 관련된 기존 연구들을 기술한다. 3장에서는 마이닝을 수행하기 위한 기본적인 개념들을 정의하고 4장에서는 스트림 데이터를 표현하는 트리모델, XFP_tree의 구성방법을 기술한다. 5장에서는 슬라이딩 윈도우 기반의 XML 스트림 데이터에 대해 XFP_tree로부터 빈발 구조를 추출하는 방법을 설명한다. 6장에서는 실험을 통해 제안된 방법을 기존 연구와 비교 평가한다. 마지막으로 7장에서 논문의 결론을 맺는다.

2. 관련 연구

스트림 데이터는 연속적이며 크기가 무한하고 스트림 데이터 시스템의 제한된 베터리, 무선 통신으로 인한 데이터 손실 등의 특성으로 인하여 잠재적인 에러가 존재한다. 그러므로 스트림 데이터 처리는 기존의 데이터베이스 시스템에서의 데이터 처리와는 다르게 축약 및 요약하여 처리되고 있다. 그리고 스트림 데이터의 크기가 무한한 특성으로 인하여 스트림 데이터의 연관규칙 정보를 탐사하는 마이닝 기법은 한 번의 데이터 스캔만으로 연관규칙 정보를 발견해야 한다[1, 2]. [1]에서는 센서 노드에서 전송되는 스트림 데이터의 크기를 줄이기 위하여 데이터를 집계(aggregation)하는 PGA(Potential Gains Adjustment) 알고리즘을 제안하였다. PGA알고리즘은 스트림 데이터 정보를 트리 형태로 구축하여 평균, 최대값, 최소값 등의 집계 연산을 통하여 요약함으로써 에러를 줄이고 전송되는 메시지 수를 줄일 수 있다. 그러나 PGA 방법은 스트림 데이터에서 직접적인 연관규칙 탐사를 수행하지는 않는다. [2]는 다차원 스트림 데이터의 연관규칙을 탐사할 수 있는 MILE(MIning from muLtiple strEams) 알고리즘을 제안하였다. MILE 알고리즘은 심장박동수, 주식 상승 가격과 같은 이벤트를 하나의 토큰으로 정의한 후 한 번의 스캔을 통하여 다차원 스트림 데이터 간의 연관 규칙을 탐사하는 방법이다. 또한 [7]은 최소 지지도를 만족하는 아이템 셋을 계산하고, 결과 중에는 사용자가 허락하는 범위 내의 오차가 있는 결과도 포함될 수 있도록 하였다. 즉, 오차를 ε, 최소 지지도를 s, 전체 데이터를 N이라고 했을 때 결과 값에서 나타나는 횟수가 sN이상이 되는 것을 모두 찾아주고, 동시에 (s-ε)N 이상이 되는 데이터들 중에서 일부도 찾아준다. 이 연구에서는 버퍼를 많이 쓸수록 더 빠른 수행 속도를 보였다. 그러나 이러한 연구들[1, 2, 7]은 구조적 특성이 있는 XML 스트림 데이터에 직접 적용하기는 어렵다.

XML 스트림 데이터에 대한 구조 발견을 위한 기법들이 [4, 8, 9, 11]에서 제안되고 있다. 많은 질의들이 비슷한 구조를 공유한다는 사실을 기반으로 연속 질의를 점증적으로 그룹핑 하는 방법을 [8]에서 제안하였다. 이 방법은 새로운 데이터가 도착했을 때 실행되는 질의(arrival-based query)나 특정 실행 간격을 지니는 질의(interval-based query) 등의 두 가지 경우 모두를 지원하여 XML 문서 스트림에 대한 다중 연속 질의 최적화에 초점을 두고 있다. [9]는 XML에 대한 빈발 질의 패턴을 발견하기 위해 Apriori 기반의 후보 패턴을 생성한다. 이 연구는 루트화 된 서브트리만을 추출하므로 일반적인 스트림 데이터에 대한 마이닝 기법으로 보기 어렵다. Asai et al.[4]는 XML 스트리밍 데이터로부터 빈발 트리를 분석하기 위해 가상 트리(virtual tree)에서 오른쪽으로 노드를 확장하는 방법(rightmost expansion)을 이용하는 온라인 알고리즘, StreamT을 제안하였다. 그러나 정확한 결과를 보장하기 어렵고 확장 가능한 노드의 경우를 모두 고려하기 때문에 많은 메모리가 필요하다. 이 연구와 유사한 방법을 적용하는 STMer 알고리즘[11]이 최근에 제안되었다. 이 연구에서도 노드의 tail-expansion 방법을 적용하여 스트림으로 입력되는 노드들을 레벨과 레이블에 따라 공통의 prefix tree를 생성하여 노드들을 확장시키고 빈발한 구조패턴을 탐색한다. 그러나 같은 범위의 노드들이 입력될 때까지 버퍼에 노드들을 저장하고 확장 가능한 노드들을 고려하는 것은 StreamT와 유사하다.

새로운 트랜잭션이 지속적으로 추가되는 환경에서 최신의 유용한 정보를 추출하기 위한 방법을 [16]에서 제안하고 있다. 이 연구에서는 새롭게 출현한 항목에 대한 지연추가와 이전 데이터 집합에 출현한 항목들 중에서 중요하지 않은 항목에 대한 전지작업이 병행되는 방법을 제안하였다. 그리고 스트리밍 XML에 대해서 질의에 빠르게 매치되는 것을 찾기 위해 스트리밍 XML 데이터 필터링 기법에 오토마타를 이용하는 방법도 제안하고 있다. [17]은 비결정적 오토마타를 사용하는 YFilter에서 부분 매칭 경로 질의가 증가하면 처리량이 떨어지는 문제를 해결하기 위하여 XPath 질의의 공통된 뒷부분 공유에 기반한 상향식 방식을 사용하는 PoSFilter를 사용하는 방법론을 제시하여 YFilter보다 좋은 처리량을 나타낸다. XML 스트림에 대한 동적인 질의 계획을 제안하는 [18]에서는 관계 데이터 스트림 모델에서 사용하는 시스템의 적응력 있는 동적인 질의 계획을 사용하는

방법의 질의 처리 모델을 제안하였다. 기존의 XML 스트림 데이터를 위한 연구들은 효율적인 질의처리 및 질의 패턴을 발견하는 방법에 중점을 두고 있다. 그러나 유비쿼터스 환경에서 일반적으로 사용되고 있는 XML 스트림 데이터에 대한 마이닝 기법의 연구가 반드시 필요하다.

이 논문에서 제안하는 XFP_tree는 [13]의 DSTree의 개념과 유사한 방식의 트리 구조를 생성한다. [12]에서는 FP-tree[15]를 변형한 DSTree[13]과 유사한 방법으로 트리를 구성하는 방법을 제안하였다. 그러나 [12]는 일괄적으로 처리하는 데이터를 대상으로 점진적으로 마이닝을 하기 위해 제안한 트리 구조이므로 연구 목적이 다르다. 또한 XFP_tree와 DSTree의 트리 구조는 유사하지만 DSTree는 구조가 없는 전형적인 데이터를 대상으로 하기 때문에 이 논문에서 제안하는 XFP_tree를 구성하는 방법과는 차이가 있다. 그리고 [4, 11]에서 현재 시점의 XML 스트림 데이터에 대해 빈발구조를 추출하는 방법을 제시하였으나 우리가 제안하는 윈도우 슬라이드 기반으로 데이터 변화를 유지하는 방법과는 다르다.

3. 기본 정의

이 논문에서는 XML 스트림 데이터에 대한 마이닝 방법을 제안한다. 이를 위해 스트림 데이터의 입력 단위 및 윈도우 사이즈, 그리고 빈발 구조를 추출하기 위한 기본 개념들을 정의한다. 스트림 데이터에서 각 XML를 트랜잭션으로 고려하며 하나의 윈도우에 포함된 일정한 트랜잭션의 수를 윈도우 크기로 고려한다.

[정의 1] 주어진 시간 t 시점에서 일정한 시간 간격 d사이 즉, [t-d+1, t]의 기간에 입력되는 데이터를 포함하고 있는 트랜잭션의 집합을 batch라 한다. 하나의 batch에는 하나 이상의 일정한 수의 트랜잭션으로 구성되어 있다. 이를 표현하면 다음과 같다.

$$B_i = \{t_1, t_2, \ldots, t_n\}, \qquad |B_i| = n$$

여기서 t_n은 트랜잭션을 의미하고 n은 일련의 트랜잭션 순서를 의미한다. 데이터 스트림은 batch 시퀀스 B_i로 나타내며, batch의 시퀀스는 1부터 연속된 번호가 부여된다.

[정의 2] 윈도우 w에는 일련의 batch들이 포함되어 있으며 포함되어 있는 batch의 수를 윈도우 사이즈라 하고 아래와 같이 |w|로 표현한다.

$$w = \{B_1, B_2, \ldots, B_m\}, \qquad |w| = m$$

여기서 윈도우 w에 m개의 batch들이 포함되어 있으므로 윈도우 사이즈는 m이 된다.

그러므로 각 batch에 포함되어 있는 트랜잭션의 수는 $|B_i|$로 표현하고, 윈도우에 포함되어 있는 batch의 수는 윈도우 사이즈 |w|로 표현한다.

연속적으로 입력되는 XML 스트림 데이터에 대한 마이닝을 수행하기 위해 이 논문에서는 빈발 구조 항목과, 현재는 빈발 구조가 아니지만 빈발 구조 항목이 될 가능성이 있는 경계구조 항목을 고려한다. 이에 대한 정의는 다음과 같다.

[정의 3] 하나의 batch에 포함되어 있는 트랜잭션의 수 $|B_i|$에 대해 루트부터 임의의 노드까지의 구조 항목 λ를 포함하고 있는 트랜잭션의 수가 주어진 지지도 Θ를 만족하면 λ는 빈발 구조 항목이라 한다. 이를 식으로 표현하면 다음과 같다.

$$\frac{|T_{\ni\lambda}|}{|B_i|} \geq \Theta \quad (0<\Theta\leq 1)$$

$|T_{\ni\lambda}|$는 구조 항목 λ항목을 포함하고 있는 트랜잭션의 수를 의미한다.

XML 데이터를 트리로 표현할 때 트리에 포함되어 있는 단일 노드 항목들의 관계는 순서적 의미를 가지는 연관성 있는 집합 구조이며 이들의 순서는 구조적 측면에서 중요하다. 하나의 XML 데이터 트리를 트랜잭션으로 간주하고 batch에 포함되어 있는 트랜잭션 수에 대한 서브트리를 구성하는 에지를 기반으로 스트림 데이터로부터 빈발 구조 항목을 추출한다.

[정의 4] 현재 빈발하지 않은 구조항목에서 빈발 구조 항목에 포함될 가능성이 잠재되어 있는 항목들을 경계 구조 항목(border항목)이라 한다. 경계 구조항목은 batch에 포함되어 있는 트랜잭션의 수 $|B_i|$에 대해 λ 항목을 포함하고 있는 트랜잭션의 수가 주어진 지지도 Θ를 만족하지 못하는 항목들 중에서 해당 항목을 포함하는 트랜잭션이 추가되면 빈발 지지도를 만족 할 수 있는 빈발 가능 항목이다. 이를 식으로 표현하면 다음과 같다.

$$\frac{|T_{\ni\lambda}|+1}{|B_i|} \geq \Theta \ (0<\Theta\leq 1)$$

경계 구조 항목은 현재는 빈발 구조 항목에 포함되지 않지만 빈발 항목에 포함될 가능성이 잠재되어 있는 항목이다. 이러한 경계구조 항목은 다음의 batch에서 동일 항목들이 추가적으로 발견되면 빈발 구조 항목에 포함될 수 있는 가능성이 있는 항목들로 구성된다. 이러한 경계구조 항목이 일정한 시점에서 빈발 지지도를 만족하면 해당 항목의 빈발도를 추정하여 빈발 구조 항목에 저장한다. 이것은 마이닝 결과에 대한 오차의 범위를 줄이기 위한 방법으로 사용된다.

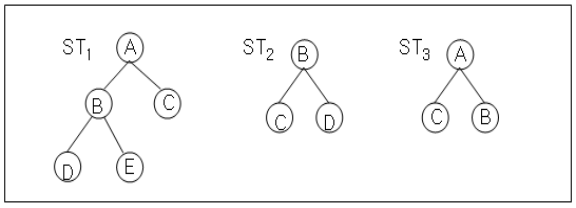
4. 스트림 빈발 패턴 트리

XML 데이터는 일반적으로 노드들에 대한 레이블 및 순서를 부여한 트리(labeled ordered tree)의 구조로 나타낸다. 트리로 구성되는 XML 데이터 스트림은 연속적으로 입력되므로 무한한 노드들의 시퀀스이다. 이 논문에서는 스트림 데이터의 빈발구조를 표현하는 트리를 XFP_tree(XML Frequent Pattern tree)라 하고, 루트를 가진 서브트리의 집합으로 나타낸다. XFP_tree는 XFP_tree= {ST_1, ST_2, .. ST_n}으로 표현되며, 여기서 ST_i는 입력되는 빈발구조이고, n은 가장 최근에 입력된 빈발구조의 식별자를 의미한다. 그리고 이 논문에서 제안하는 XFP_tree는 유사 도메인의 정보를 나타내는 XML 스트림 데이터가 입력된다고 가정한다.

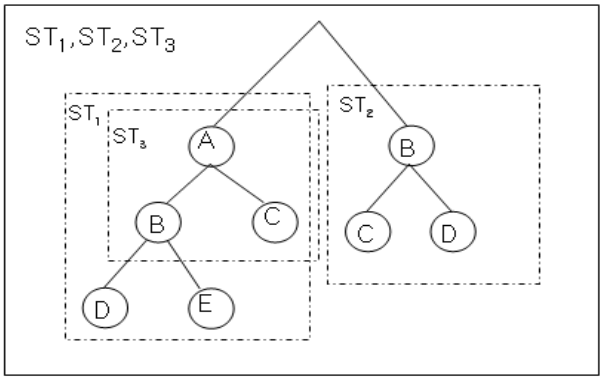
스트림 데이터에 대한 트리 XFP_tree를 구성하는 방법은 첫째, 처음으로 입력되는 루트를 포함하는 XML 데이터에 대한 기초 트리 XFP_tree를 구성한다. 둘째, 새롭게 입력되는 트리의 루트노드와 현재의 XFP_tree의 루트노드가 같지 않으면 새로운 서브트리로 구성한다. XML의 빈발구조는 루트부터 자신의 노드까지의 패스(path)가 구조를 구분하는 중요한 의미를 갖기 때문에 서브 트리의 루트 노드가 같을 때에만 결합한다. 또한 트리를 구성할 때 형제 노드의 순서는 고려하지 않는다. 이것은 빈발 구조의 발견에서 형제노드의 순서는 중요한 의미를 갖지 않기 때문이다.

(그림 1)은 연속적으로 입력되는 XML 스트림 데이터의 빈발구조 ST_1, ST_2, ST_3를 트리로 표현한 것이다.

(그림 2)는 그림 1의 빈발구조를 XFP_tree로 구성한 예를 보여준다. 서브트리 ST_1, ST_3은 루트노드 A가 같기 때문에 같은 루트를 갖는 서브트리로 구성하고, ST_2는 루트 노트 B



(그림 1) XML 스트림 데이터 트리 표현



(그림 2) 스트림 데이터 집합을 표현하는 트리(XFP__tree)

를 갖는 새로운 서브트리로 구성한다.

XML 데이터에 대한 빈발 트리에 대해 [9]에서는 전체 트리 수에 대한 공통 루트를 가진 서브트리(rooted tree)의 포함 정도로써 트리의 빈발도를 나타낸다. 그러나 이 논문에서는 루트를 가진 서브트리의 개념이 아닌 부모노드와 자식노드간의 관계를 나타내는 에지(edge)를 트리의 빈발도를 측정하는 기본 개념으로 사용한다.

XML 스트림 데이터에 대해 XFP_tree를 이용하여 빈발구조를 마이닝 하기 위한 기본 개념은 다음과 같이 정의한다.

[정의 5] 빈발 XML 스트림 데이터를 표현하는 트리는 XFP_tree=<E_1, E_2,..E_n>의 에지들로 구성되며, 각각의 에지는 부모노드와 자식노드의 순서관계 E_i = <P_i, C_i>로써 이루어진다.

XFP_tree는 루트노드가 같고 부모노드와 자식노드 관계가 같으면 공통의 에지로 표현한다. 즉, 그림 2에서 ST_1, ST_3의 루트노드 A가 동일하므로 <A, B>, <A, C>는 공통의 에지로써 하나의 에지로 표현된다. 그러나 ST_1, ST_2는 루트 노드가 다르므로 분리된 서브트리로 표현한다. XML 데이터에서 에지는 데이터의 구조를 구성하는 기초단위로써, XML의 구조 정보를 나타내는 중요한 요소이며 루트부터 특정 노드까지의 패스(path) 정보는 연속된 에지관계로써 XML의 구조를 나타내는 중요한 의미를 지닌다.

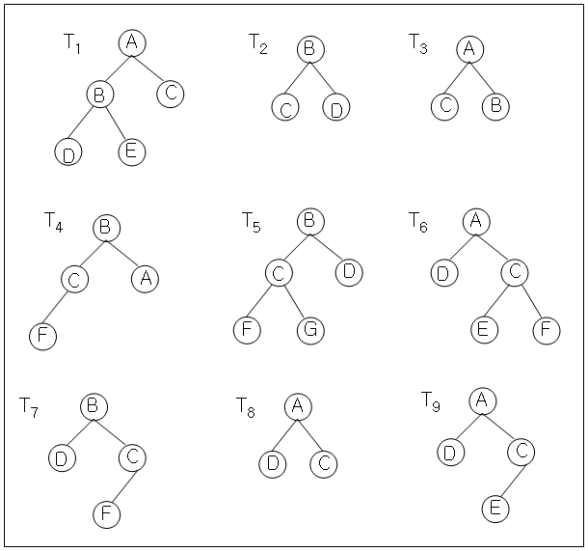
[정의 6] XFP_tree에서 루트 노드 n_1부터 임의의 노드 n_i까지의 연속된 에지들의 구조정보를 XFPath라 하고, XFPath=<n_1, n_2, ... , n_i>는 노드들의 레벨 순서에 의해 구성된다.

XFPath는 부모노드와 자식노드와의 관계를 나타내는 에지를 확장하여 루트노드부터 특정노드까지의 구조정보를 표현하므로 패스를 구성하는 에지와 노드들의 레벨정보를 포함하는 빈발 구조의 중요한 요소이다.

[정의 7] XML 스트림 데이터에 대한 구조의 빈발도 XSup(E_i)는 XML 트리를 구성하는 XFP_tree에 포함되어 있는 에지 E_i=<P_i, C_i>의 발생 빈도이다. 이에 대한 식은 다음과 같이 나타낸다.

$$XSup(E_i) = \frac{|E_i|}{|XFP\_tree|}$$

여기서 |XFP_tree|는 XML 스트림 데이터 트리 XFP_tree에 포함되어 있는 개별 XML 트리의 수 즉, 트랜잭션 수를 의미하며, |E_i|는 에지의 발생빈도를 의미한다. 에지는 각 XML 데이터를 나타내는 트리에서 유일한 노드의 관계로 구성되며, 주어진 사용자 지지도(0<min_sup ≤1)를 만족하면 XSup(E_i) ≥ min_sup 이므로 에지 E_i는 빈발하다는 것을 의미한다. 예를 들면, 그림 2에서 에지 <A, B>와 <A, C>의 빈발도는 2/3(0.66)이다.



(그림 3) 스트리밍 XML 데이터의 트리 예

5. XML 스트림 데이터의 빈발 패턴 마이닝

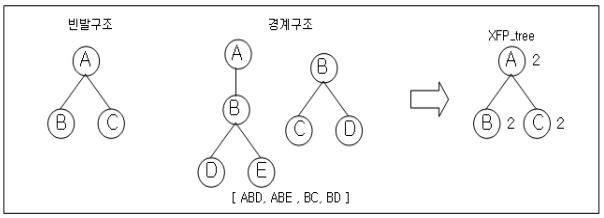
XML 데이터는 구조적인 순서가 중요한 요소이기 때문에 구조 정보를 유지하면서 트리를 구성해야 한다. 이 논문에서는 슬라이딩 윈도우 기반의 스트리밍 XML 데이터에서 빈발 구조를 추출하기 위하여 FP-tree를 변형한 XFP_tree를 이용한다. XFP_tree는 XML의 구조를 고려한 빈발 트리로써 XML의 빈발 구조 및 빈발도를 유지한다. 이 장에서는 트랜잭션의 삽입과 삭제에 따른 XFP_tree의 변화과정과 XFP_tree로부터 빈발 구조를 마이닝하는 과정에 대해 설명한다.

5.1 트랜잭션 삽입

이 논문에서는 트랜잭션들로 이루어진 스트림이 고정된 크기로 일괄(batch)처리되어 입력된다고 가정한다. 즉, 하나의 batch에는 일정한 수의 트랜잭션으로 구성되어 있다. XFP_tree의 빈발도 유지에 대한 예를 보이기 위해 각 batch에 포함되어 있는 트랜잭션을 B_1(t1, t2, t3), B_2(t4, t5, t6), B_3(t7, t8, t9)라 하고, 윈도우 사이즈를 2 batch라 가정한다.

입력되는 XML 데이터로부터 빈발한 구조들로 구성되는 트리 XFP_tree는 XML의 구조를 구성하는 엘리먼트들을 트리의 노드로 구성하여 항목들을 나타내고 각 항목에 대한 빈발도를 유지한다. 그리고 같은 경로를 갖는 노드들은 경로를 공유하는 구조로 나타낸다. (그림 3)은 시간에 따라 입력되는 XML 스트림 데이터로부터 XFP_tree의 구조를 구성하는 예를 보이기 위한 입력 트리이다.

(그림 4)는 순서대로 입력된 그림 3의 T_1, T_2, T_3으로 구성된 B_1에서 min_sup를 0.5로 하여 빈발 구조와 경계구조(border) 그리고 XFP_tree를 구성한 결과이다. 즉, XFP_tree는 빈발 구조를 나타낸 트리이다. XFP_tree를 구성할 때 각



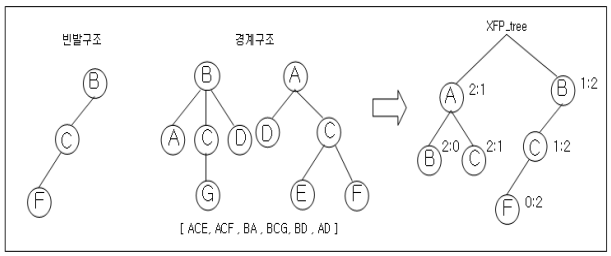
(그림 4) B_1의 빈발구조, 경계구조, XFP_tree

서브트리들의 루트가 다르면 분리된 구조의 서브트리로 구성된다. B_1에서의 경계구조는 현재 B_1에서는 min_sup를 만족하지 못하여 빈발구조에 속하지 않는 구조이나 지속적으로 입력되는 스트림 데이터에서 빈발할 가능성이 있는 잠재적인 빈발 구조이다.

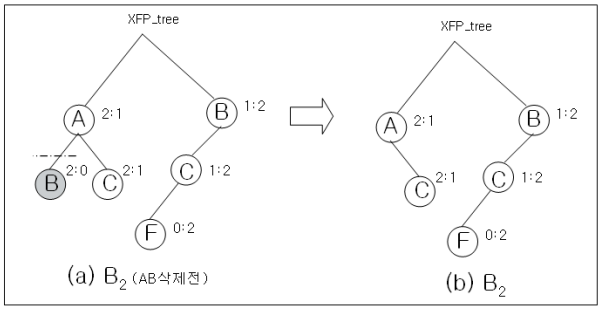
(그림 3)의 T_4, T_5, T_6으로 구성된 B_2의 데이터가 입력되었을 때의 B_2에서의 빈발 구조와 경계 구조를 나타낸 것이 (그림 5)이고 XFP_tree는 (그림 4)의 XFP_tree에 추가적으로 삽입된 B_2의 빈발구조를 포함한 결과를 보여준다. 즉, (그림 4)의 XFP_tree에 입력된 B_2의 빈발 구조, XFPath B-C-F를 XFP_tree에 삽입한다. 여기서 B-C-F는 현재 XFP_tree에 포함되어 있지 않은 새로운 구조이다. 새로운 빈발 구조를 삽입할 때는 이 구조가 경계구조에 포함되어 있었던 항목인지의 여부에 따라 빈발도를 추정하여 빈발도를 삽입한다.

빈발도를 추정하는 방법은 빈발구조가 경계구조에 존재하는 경우의 추정 빈발도는 ($|B_{i-1}|$* min_sub) -1이고, 경계구조에 존재하지 않을 때의 추정 빈발도는 ($|B_{i-1}|$* min_sub)-2에 의한다(추정 빈발도 계산은 윈도우 사이즈 및 배치에 포함된 트랜잭션 수에 따라 달라질 수 있다). 그러므로 빈발 구조 B-C-F의 추정 빈발도는 구조의 기초단위가 되는 에지 단위의 구조를 고려할 때 에지 B-C는 경계구조에 존재하므로 1이 되고 B-C-F는 경계구조에 존재하지 않으므로 추정 빈발도는 0이 된다. (그림 5)의 XFP_tree는 추정된 빈발도를 포함하여 구성한 것이다.

또한 XFP_tree를 유지함에 있어 계속적으로 입력되는 batch로 인해 더 이상 빈발도를 만족하지 않는 구조가 있는지 점검하여 삭제하는 과정이 필요하다. 삭제될 구조의 최소 빈발도는 현재까지 입력된 batch에 포함된 트랜잭션의 수에 대한 지지도를 고려하여 $\sum_{i=1}^{n}|B_i|$* min_sub를 만족하는



(그림 5) B_2의 빈발구조, 경계구조, XFP_tree



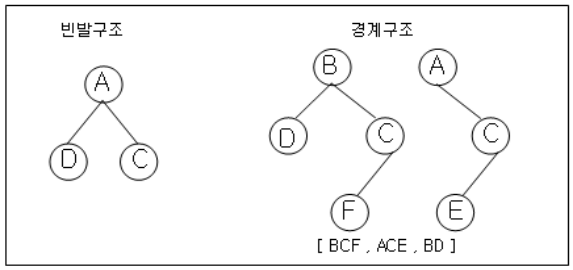
(그림 6) B_2의 입력에 따른 삽입 및 삭제 후의 XFP_tree

지 여부를 판단한다. (그림 6)의 (a)에서 빈발도를 만족하지 않는 구조는 에지 A-B이므로 이를 삭제한다. 이에 대한 XFP_tree의 결과를 (그림 6)의 (b)에서 보여준다. 그러나 현재 입력된 batch에서 발견된 빈발 구조는 $\sum_{i=1}^{n}|B_i|*$ min_sub를 만족하지 않아도 제거하지 않는다(예, B-C-F). 이것은 현재의 데이터에 대한 중요도를 우선적으로 고려하기 위해서이다.

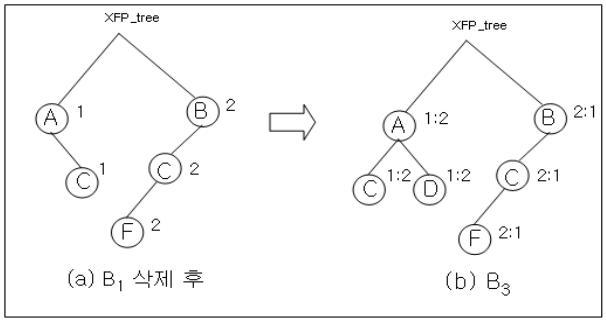
그리고 XFP_tree에서 빈발도를 만족하지 못하는 구조의 삭제와 더불어 경계 구조에 대한 관리도 추가적으로 이루어진다. B_1의 입력상태에서의 경계구조 집합은 {ABD, ABE, BC, BD}이었고 B_2의 입력으로 인한 추가적인 경계구조항목은 {ACE, ACF, BA, BCG, BD, AD}이다. 스트림 데이터에서는 최근에 입력된 데이터를 중요하게 고려한다. 그러므로 경계 구조항목의 유지에서도 현재 batch에서의 경계 구조항목은 경계항목에 포함하여 유지하고 이전 배치에서의 경계항목이 최근에 입력된 배치에 빈발항목이나 경계항목에 포함되지 않는다면 경계 구조항목에 대한 지지도를 만족하지 못하므로 경계 구조항목에서 제거한다. 즉, B_1에 대한 경계구조 집합의 항목들이 B_2에서의 입력 구조항목에 포함되어 있지 않은 {ABD, ABE, BC}항목들은 제거하고 B_2에서 경계항목인 {BD}항목은 유지한다. 그리고 B_2에서의 경계구조 항목들인 {ACE, ACF, BCG, BA, AD}도 유지한다. XFP_tree에서 삭제된 A-B구조는 빈발도에 의해 경계구조항목으로 삽입된다. 결과적으로 B_2의 삽입에 따른 경계구조 항목으로는 {ACE, ACF, BCG, BA, AD, AB, BD}으로 유지된다.

5.2 트랜잭션 제거

윈도우 사이즈를 2라 가정할 때 B_3이 입력되면 윈도우



(그림 7) B_3의 빈발구조와 경계구조



(그림 8) B_3의 입력에 따른 B_1 삭제 후의 XFP_tree

사이즈를 고려하여 가장 오래전에 입력된 B_1의 정보를 XFP_tree로부터 삭제하고 최근 데이터인 B_2와 B_3의 빈발구조정보를 XFP_tree에 유지한다. B_1의 삭제는 XFP_tree의 각 노드정보를 일괄적으로 이동하여 간단히 삭제하고 B_3의 빈발 구조를 XFP_tree에 추가한다. (그림 7)은 B_3에 포함되어 있는 빈발 구조와 경계구조 정보를 보여주고, (그림 8)의 (a)는 B_1의 정보를 삭제한 후의 결과를 보여주며 (b)는 B_3의 빈발 구조와 경계구조 정보를 삽입한 결과를 보인다.

가장 오래된 B_1를 삭제하고 B_3의 빈발구조 {AD, AC}를 XFP_tree에 삽입한다. 이 때 에지 A-D는 새롭게 XFP_tree에 삽입되는 구조이므로 현재 batch에서의 빈발도 및 추정 빈발도를 계산하여 빈발도를 유지한다. 에지 A-D구조의 추정 빈발도는 B_2 상태에서 경계구조에 존재하므로 ($|B_{i-1}|*$ min_sub)-1에 의해 1이 된다. 그리고 XFPath B-C-F는 현재의 B_3에서 경계구조이고 XFP_tree에 이미 존재하여 빈발도를 만족하므로 현재의 빈발도 유지한다.

이와 같이 윈도우 슬라이딩을 기반으로 연속적인 데이터에 대한 빈발구조를 반영하는 XFP_tree의 유지과정은 다음과 같은 알고리즘에 의해 수행된다.

```
1. While Bi is inputted // 데이터 스트림의 batch 입력
  2. Find the frequent structures and border structures in Bi
    2.1 If i of batch (Bi) = 1   //첫번째 B1 입력
      2.1.1 XFP_tree 초기화  // 빈발 구조로 기초 트리 구성
      2.1.2 경계 구조항목 초기화
    2.2 Else if  i of batch (Bi) ≤ |w| //  batch 수가 윈도우 사이
        즈보다 작을 경우
      2.2.1 XFP_tree에 새로운 빈발 구조 항목 추가
      2.2.2 기존의 빈발구조 항목의 빈발도 갱신
        // XFP_tree에 이미 존재하는 빈발구조
      2.2.3 XFP_tree와 경계 구조 항목 유지
        //최소 지지도를 만족하지 않는 구조항목 제거 및
        경계구조 항목에 삽입해야 할 구조항목 분류
    2.3 Else  //i of batch (Bi) < |w| //  batch 수가 윈도우 사이
        즈보다 큰 경우
      2.3.1 XFP_tree에서 가장 오래된 batch 일괄 제거
      2.3.2 XFP_tree에 새로운 빈발 구조 항목 추가
      2.3.3 기존의 빈발구조 항목의 빈발도 갱신
      2.3.4 XFP_tree와 경계 구조 항목 유지
```

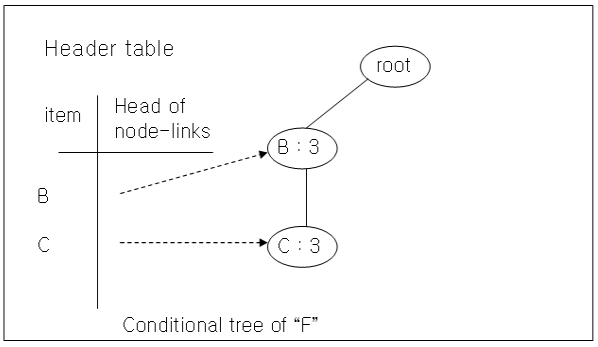
5.3 빈발 구조 마이닝

빈발구조 탐색은 현재 윈도우 범위에 속하는 스트림 XML 데이터들로 구성된 트랜잭션으로부터 빈발 구조를 마이닝 한다. 즉, 마이닝 요청시에 XFP_tree로부터 주어진 임계치 이상의 빈발 구조를 추출한다.

스트림 데이터가 입력된 임의의 시간 t에 대해 빈발 구조 집합을 나타내는 XFP_tree 상태를 (그림 8)의 (b)라 할 때, XFP_tree로부터 일정 임계치 이상의 빈발 구조만을 추출하고자 하는 최소 지지도를 min_support =3라 하면, 이를 만족하는 항목 및 이들의 빈발 지지도는 {A:3, B:3, C:6, D:3, F:3, AC:3, AD:3, BC:3, BF:3, CF:3, BCF:3}이다. 즉, 항목 C의 지지도는 에지 A-C에서의 1:2와 에지 B-C에서 2:1의 합을 나타낸다. 여기서 A-C의 1:2 의미는 B_2에서의 A-C빈발 지지도가 1이고 B_3에서는 2인 것을 나타낸다. 그러므로 현재 시점에서의 AC 빈발 지지도는 1과 2의 합인 3이 된다. 같은 방법으로 에지 B-C의 빈발 지지도는 3이다.

XFP_tree에서의 빈발구조 추출은 FP-tree와 같은 방법으로 주어진 지지도를 만족하는 트리의 노드들에 대한 부분 데이터(projected database)를 생성하여 빈발 지지도를 얻을 수 있다. 예를 들어, 노드 F에 대한 조건부 트리는 루트 노드 B로부터의 패스 B-C-F(2:1)에서(B 2:1, C 2:1)를 기초로 (그림 9)와 같이 구축된다.

XFP_tree로부터 주어진 지지도를 만족하는 경로 및 에지를 마이닝 할 때는 FP-growth와 같은 방법으로 XFP_tree 트리의 노드들에 대한 조건부 트리를 구성하고, 이들 조건부 트리로부터 빈발한 경로 및 에지를 발견한다. 즉, (그림 9)로부터 노드 F에 대한 경로 및 에지의 빈발도는 BCF :3 , BF :3, CF : 3임을 알 수 있다. (그림 10)은 XFP_tree를 구성하는 각 항목에 대한 조건부 패턴을 찾아가는 과정을 반복적으로 수행한 결과를 보여준다. 이와 같이 윈도우 이동에 따른 XFP_tree를 구성하고, 트리를 구성하는 노드를 중심으로 조건부 부분 트리를 생성하여 노드간의 에지 및 노드의 경로에 대한 빈발도를 유지하여 사용자에게 마이닝 결과를 빠르게 응답할 수 있다.



(그림 9) 노드 F에 대한 조건부 트리

item	Conditional pattern base	Conditional XFR_tree
C	{A:3), {B:3}	{(A:3), (B:3)}|C
D	{A:3}	{(A:3)}|D
F	{B:3, C:3}	{(B:3, C:3)}|F
A	Φ	Φ
B	Φ	Φ

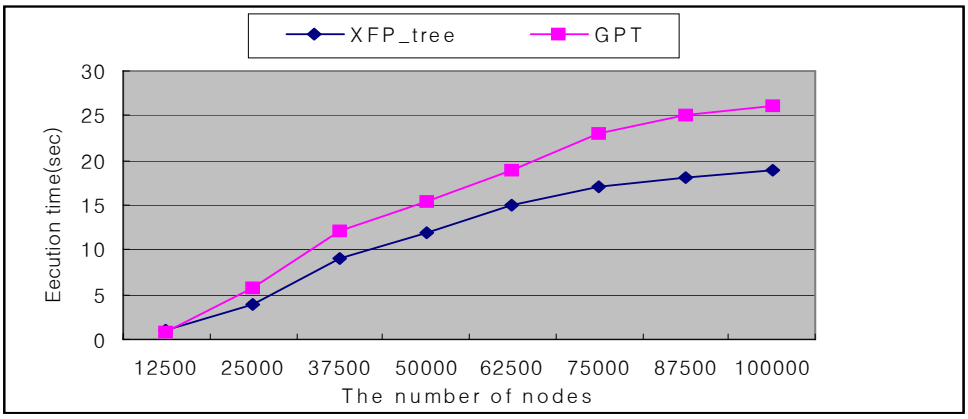
(그림 10) 각 노드항목에 대한 조건부 패턴

6. 실험 결과

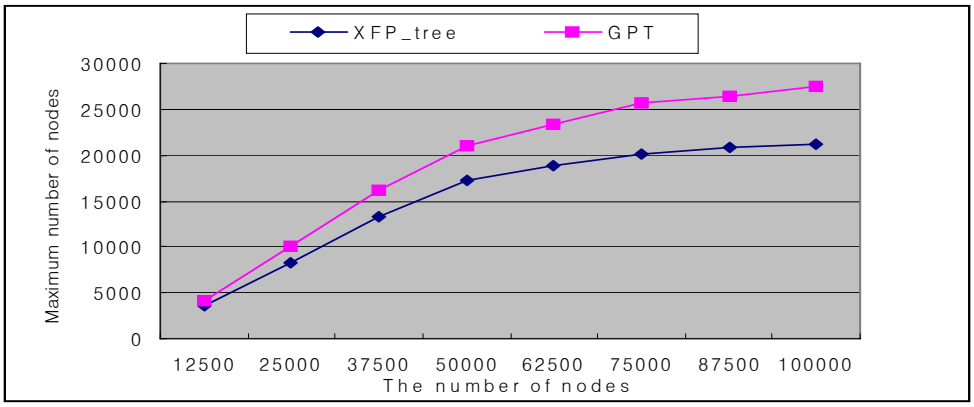
이 논문에서는 XML 스트림 데이터로부터 빈발 구조를 추출하기 위하여 후보 빈발 구조들을 표현하는 XFP_tree를 구성하고, 이를 이용하여 빠르게 빈발 구조를 추출하는 방법을 제안하였다. 제안한 XFP_tree에 대한 효율성을 측정하기 위하여 http://dblp.uni-trier.de/xml 사이트의 데이터와 XML 데이터 뱅크[19]의 DTD를 이용하여 IBM's AlpaWords XML generator에 의해 실험 데이터를 생성하였다. 생성된 3000개의 XML 데이터를 스트림 데이터가 입력되는 형태로 순차적으로 입력하여 실험하였고, 생성된 XML 데이터의 평균적인 트리 깊이는 5.2이다. XFP_tree의 실험에 필요한 조건으로 윈도우 사이즈 ω=4, 그리고 각 batch에 포함되는 트랜잭션의 수는 100개로 하였으며, 이 논문에서 제시하는 방법의 효율성을 평가하기 위해 가장 유사한 연구인 [11]과 비교 실험하였다. [11]에서는 공통의 prefix tree로 구성되는 GPT(Global Prefix Tree)를 생성하고 이를 이용하여 빈발구조를 마이닝 한다. 첫 번째 실험에서는 마이닝 성능에 많은 영향을 미치는 XFP_tree 와 GPT를 구성하는 데 소요되는 시간을 측정하여 비교하였다. (그림 11)은 입력된 XML 스트림 데이터에 대한 노드 수의 증가에 따라 트리 구성에 소요되는 시간을 측정한 결과를 보여준다.

(그림 11)의 결과에서 노드의 수가 적은 초기의 GPT 구성은 XFP_tree보다 근소한 차이의 적은 시간이 소요되지만 노드의 수가 많아질수록 XFP_tree보다 더 많은 시간이 소요되는 것을 알 수 있다. GPT는 입력되는 스트림 데이터에 대해 확장 가능한 노드를 중심으로 서브트리 구성을 지연시킨 후 일정 범위의 노드가 최종적으로 입력되면 서브트리를 만들고 이것을 GPT로 병합시키는 과정을 거친다. 그리고 XFP_tree는 스트림 데이터에서 일정한 크기의 트랜잭션 단위인 batch로 구분하고 각 batch에서 빈발구조를 찾는 전처리 과정을 통해 빈발 구조로 트리를 구성한다. 노드의 수가 증가할수록 GPT의 트리 구성 시간이 더 많이 소요되는 것은 입력되는 트리의 노드들에 대해 확장 가능한 노드들을 모두 고려하여 서브트리를 구성하고 이를 GPT에 병합하는 과정에서 많은 시간이 소요되는 것으로 생각된다. 반면 XFP_tree은 각 batch의 트리에서 빈발 구조를 필터링하는 전처리 과정을 거쳐 XFP_tree를 구성하므로 서브 트리의 모든 노드들을 GPT에 병합하는 것에 비해 트리를 구성하는 시간이 적게 소요되는 것임을 알 수 있다.

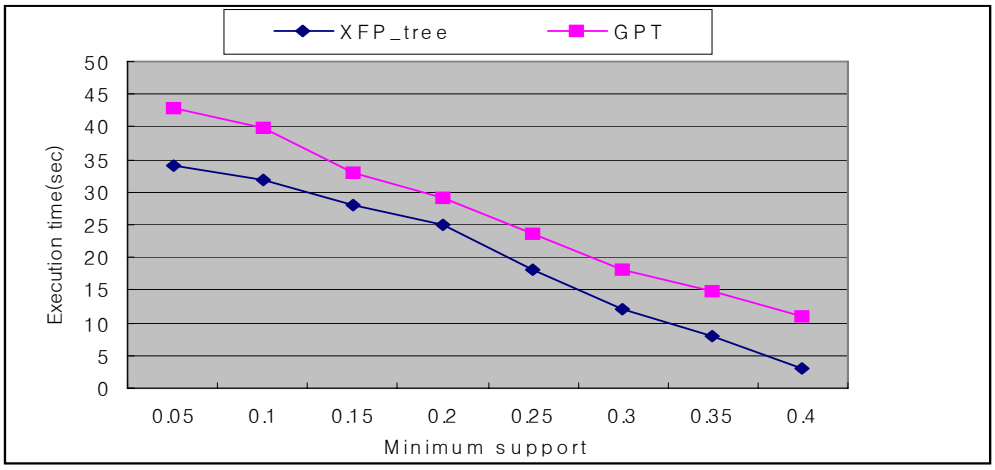
두 번째 실험에서는 트리의 구성에 있어 입력된 노드의



(그림 11) 노드 수의 변화에 따른 수행시간



(그림 12) 노드 수의 변화에 따른 메모리 사용량



(그림 13) 최소 지지도 변화에 따른 수행시간

수의 변화에 따른 메모리 사용량의 변화를 비교하기 위하여 입력된 전체의 노드 수에 대해 각 트리에서 유지되는 최대 노드 수의 비율에 대한 비교를 수행하였고 이에 대한 결과를 (그림 12)에서 보여준다.

GPT는 스트림 데이터를 표현하는 서브트리(Augment Prefix Tree)들을 병합하고 나서, 최소 지지도를 만족하지 않는 노드들에 대한 가지치기를 수행한다. 그러므로 가지치기를 수행하기 전의 트리 사이즈는 가지치기 하고 난 후의 트리 사이즈의 최고 두 배에 이르는 메모리가 필요하였다. 반면에 XFP_tree는 지정된 윈도우 사이즈에 포함되어 있는 일정한 수의 트리에서 빈발 구조만을 선별하는 전처리과정을 통해 빈발구조만을 XFP_tree에 포함한다. 우리는 정당한

메모리 사용량의 비교를 위해 그림 12에서 나타난 GPT의 메모리 사용량은 가지치기를 수행 한 후의 결과를 가지고 비교한 것이다. 실험 결과로부터 XFP_tree가 GPT보다 더 적은 메모리를 사용하는 것으로 나타난다. 이것은 XFP_tree는 윈도우 슬라이드를 기반으로 하여 주기적으로 오래된 데이터를 일괄 삭제하는 반면에 GPT는 계속적인 데이터의 입력에 대해 빈발도를 만족하지 않는 노드들만을 삭제하여 트리를 유지하므로 XFP_tree에 비해 더 많은 수의 노드를 유지하고 있기 때문이라 생각된다. 지지도의 변화에 따라 메모리 사용량에 대한 차이를 감안하더라도 트리 유지를 위한 메모리 사용은 XFP_tree가 GPT보다 평균적으로 20-30% 적게 사용하는 결과를 알 수 있었다.

다음은 지지도의 변화에 따라 XFP_tree와 GPT를 구성하고 유지하는 데 소요되는 시간에 대한 비교를 하였고 그림 13에서 결과를 보여준다. 지지도가 커질수록 XFP_tree와 GPT의 구성에 소요되는 시간이 급격히 줄어드는 것을 볼 수 있다. 이것은 빈발 지지도의 임계값이 커질수록 이를 만족하는 빈발 구조의 노드 수가 감소하기 때문에 수행시간이 적게 소요되는 것임을 알 수 있다. 그러나 지지도 임계값이 작아지면 더 많은 빈발 구조의 노드들을 XFP_tree와 GPT에서 유지하게 되어 마이닝 결과의 정확도에는 더 좋은 결과를 유도할 수 있다. 그러므로 지지도 임계값은 마이닝의 수행시간과 정확도에 미치는 영향을 고려하여 반복적인 실험을 통해 적절한 임계치를 설정하는 것이 중요하다.

또한 마이닝 결과의 정확도를 알아보기 위해 실험 데이터 전체에 대한 일괄적인 마이닝을 수행하여 추출된 빈발구조의 결과와 비교하였다. 비교 결과에서 GPT와 XFP_tree 모두 일괄적인 마이닝 수행 결과와 비교하여 98%이상의 정확도를 보였다.

GPT는 오차 파라미터를 이용하여 트리의 가지치기를 수행하기 때문에 오차 허용범위의 임계값과 추정 빈발도 방식이 마이닝 결과의 정확도에 영향을 미친다. 그리고 XFP_tree는 최소 지지도의 임계값에 의한 경계구조와 추정 빈발도 계산에 따라 마이닝 결과의 정확도에 영향을 준다. 이러한 마이닝 결과의 정확성에 영향을 주는 요소들에 대한 비교 및 이로 인한 마이닝 결과의 정확도 변화에 대한 상세분석이 향후 연구에서 지속적으로 수행될 것이다.

7. 결 론

XML은 어떤 서비스든지 그 서비스를 기술하는 방법을 제공하는 유연성이 있으므로 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있다. 이 논문에서는 XML 스트림 데이터로부터 최신의 데이터에 대한 빈발구조를 빠르게 추출하기 위한 방법으로 경계구조 항목을 고려하고, 빈발 구조집합을 나타내는 XFP_tree를 구성하여 마이닝하는 방법을 제안하였다. XFP_tree는 지속적으로 입력되는 XML 스트림 데이터에서 각 XML의 빈발 구조를 결합하여 생성하는 빈발 구조의 트리 집합으로써 슬라이딩 윈도우 기반으로 하여 최신의 데이터에 대한 빈발 구조 데이터를 표현한다. 그러므로 최신 데이터에 대한 특정 임계값 이상의 빈발 구조를 추출하거나 사용자의 빈발 질의 구조 분석 및 결과를 제공하고자 할 때 XFP_tree로부터 결과를 빠르게 얻을 수 있다. 또한 제안한 방법에 대한 성능 평가를 위해 기존 연구와의 트리 구성 소요 시간 및 메모리 사용량에 대한 비교 실험을 수행하였다. 그러나 윈도우 사이즈에 따른 성능평가 및 오차 범위를 고려한 추정 빈발도의 마이닝 정확도에 미치는 영향 등에 대한 분석이 추가적으로 수행되어야 할 필요가 있다.

참 고 문 헌

[1] A. Deligiannakis, Y. Kotidis, and Roussopoulos, "Hierarchical In-Network Data Aggregation with Quality Guarantees," LNCS(EDBT 2004), 2004.

[2] G. Chen, X. Wu, and X. Zhu, "Mining Sequential Patterns Across Data Streams," Univ. of Vermont Computer Science Technical Report(CS-05-04), 2005.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," Invited paper in Proc. of PODS, 2002.

[4] T.Asai, K.Abe, S. Kawasoe, H.Sakamoto, et al., "Online algorithms for mining semi-structured data stream," In.Proc. ICDM, 2002.

[5] T. Dalamagas, T. Cheng, K. J. Winkel, and T. Sellis, "Clustering XML Document by Structure," The 3rd Helenic Conference on AL. SETN, 2004.

[6] M. Zaki, "Efficiently Mining Frequent Tree in a Forest," Proceedings of the ACM SIGKDD International Conference, 2002.

[7] G. S. Manku, R. Motwani, "Approximate Frequency Counts over Data Streams," VLDB 2002.

[8] J. Chen, D. J. DeWitt, F. Tian, U. Wang, " A Scalable Continuous Query System for Internet Database," ACM SIGMOD, 2000.

[9] L.H. Yang, M.L. Lee, W. Hsu, "Finding hot query patterns over an XQuery stream," VLDB Journal Special Issue on Data Stream Processing, 2004.

[10] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi, "A Tool for Extracting XML Association Rules from XML Documents," Proceedings of IEEE-ICTAI 2002, USA, November 2002.

[11] M.C. Hsieh, Y.H. Wu, A.L. Chen, "Discovering Frequent Tree Patterns over Data Stream," In Proc of SIAM International Conference on Data Mining, 2006.

[12] C. K. S. Leung Q. I. Khan, T. Hoque, "CanTree:A Tree Structure for Efficient Incremental Mining of Frequent Pattern Sets," In proc. ICDM 2005.

[13] C. K. S. Leung Q. I. Khan, "DSTree:A Tree Structure for

the Mining of Frequent Sets from Data Streams," In proc. ICDM 2006.
[14] J. Li. D. Maier, "Semantics and Evaluation Techniques for Window Aggregates in Data Streams," In Proc. of ACM SIGMOD International Conference on the Management of Data, 2005.
[15] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," In Proc. of ACM SIGMOD International Conference on the Management of Data, 2000.
[16] 장중혁, 이원석, "데이터 스트림에서 개방 데이터 마이닝 기반의 빈발 항목 탐색,"정보처리학회논문지D, 제10-D권 제3호, 2003.
[17] 박석, 김영수, "부분매칭 경로 질의를 위한 포스트픽스 공유에 기반한 스트리밍 XML 데이터 필터링 기법," 정보과학회 제33권 제 1호, 2006.
[18] 김영현, 강현철, "XML 스트림 데이터에 대한 적응력 있는 질의 처리 시스템,"정보과학회, 제33권 제 3호, 2006.
[19] NIAGARA query engine. http://www.cs.wisc.edu/niagara/data.html

황 정 희

e-mail : jhhwang@nsu.ac.kr
1991년 충북대학교 전산통계학과(이학사)
2001년 충북대학교 전자계산학과(이학석사)
2005년 충북대학교 전자계산학과(이학박사)
2001년~2005년 정우씨스템(주) 연구소장
2006년~현 재 남서울대학교 컴퓨터학과 전임강사
관심분야 : XML 및 웹 데이터베이스, 스트림 데이터 관리, 데이터 마이닝, 유비쿼터스 컴퓨팅, 시공간 데이터베이스, RFID 보안