한 번의 데이터베이스 탐색에 의한 빈발항목집합 탐색

채 덕 진[†] · 김 룡[††] · 이 용 미[†††] · 황 부 현[††††] · 류 근 호[†††††]

요 약

본 논문에서는 한 번의 데이터베이스 스캔으로 빈발항목집합들을 생성할 수 있는 효율적인 알고리즘을 제안한다. 제안하는 알고리즘은 빈발항목과 그 빈발항목을 포함하고 있는 트랜잭션과의 관계를 나타내는 이분할 그래프(bipartite graph)를 생성한다. 그리고 생성된 이분할 그래프를 이용하여 후보 항목집합들을 생성하지 않고 빈발 항목집합들을 추출할 수 있다. 이분할 그래프는 빈발항목들을 추출하기위해 대용량의 트랜잭션 데이터베이스를 스캔할 때 생성된다. 이분할 그래프는 빈발항목들과 그들이 속한 트랜잭션들 간의 관계를 엣지(edge)로 연결한 그래프이다. 즉, 본 논문에서의 이분할 그래프는 대용량의 데이터베이스에서 쉽게 발견할 수 없는 빈발항목과 트랜잭션의 관계를 검색하기 쉽게 색인(index)화한 그래프이다. 본 논문에서 제안하는 방법은 한 번의 데이터베이스 스캔만을 수행하고 후보 항목집합들을 생성하지 않기 때문에 기존의 방법들보다 빠른 시간에 빈발 항목집합들을 찾을 수 있다.

키워드 : 데이터마이닝, 연관규칙, 이분할 그래프

Frequent Patterns Mining using only one-time Database Scan

Duck Jin Chai[†] · Long Jin[††] · Yongmi Lee[†††] · Buhyun Hwang[††††] · Keun Ho Ryu[†††††]

ABSTRACT

In this paper, we propose an efficient algorithm using only one-time database scan. The proposed algorithm creates the bipartite graph which indicates relationship of large items and transactions including the large items. And then we can find large itemsets using the bipartite graph. The bipartite graph is generated when database is scanned to find large items. We can't easily find transactions which include large items in the large database. In the bipartite graph, large items and transactions are linked each other. So, we can trace the transactions which include large items through the link information. Therefore the bipartite graph is a indexed database which indicates inclusion relationship of large items and transactions. We can fast find large itemsets because proposed method conducts only one-time database scan and scans indexed the bipartite graph. Also, it don't generate candidate itemsets.

Key Words : Data Mining, Association Rule, Bipartite Graph

1. 서 론

연관규칙 탐사는 데이터베이스를 이루는 트랜잭션들에서 사용자가 미리 정한 최소지지도(minimum support) 보다 많이 발생하는 빈발항목집합(large itemset or frequent itemset)들을 발견하는 문제로 정의할 수 있다. 지금까지 빈발항목집합들을 찾기 위한 다양한 알고리즘들이 제안되었다[1, 2, 3, 4, 5, 7, 8, 9, 10].

기존에 제안된 대부분의 알고리즘들은 Apriori 알고리즘[3]과 같은 휴리스틱(heuristic) 방법을 사용하였다. Apriori와 같은 휴리스틱 알고리즘들은 대용량의 데이터베이스를 반복적으로 스캔(scan)하는 것과 대단히 많은 후보항목집합(candidate itemset)들을 다루는 비용에 대한 문제를 가지고 있다. 따라서 기존의 알고리즘들은 이러한 많은 비용을 필요로 하는 두 가지 문제를 해결하기 위한 노력에 집중되었다. [5]에서는 후보항목집합들을 생성하지 않고 두 번의 데이터베이스 스캔으로 빈발항목집합들을 생성할 수 있는 FP-growth 알고리즘을 제안하였다. FP-growth 알고리즘은 빈발 패턴 트리(Frequent Pattern tree, FP-tree)구조를 이용하여 패턴들을 추출하게 된다. 후보항목집합 생성을 피함으로써 FP-growth 알고리즘은 많은 성능의 개선을 보였다. 그러나 두 번의 데이터베이스 스캔을 통해 FP-tree를 생성하고, 트리의 모든 노드를 검사하여 빈발항목집합들을 생성하는 기본적인 연산 비용 외에 모든 트랜잭션에 대해서 빈발하지 않는 항목들을 전지(pruning)하고 나머지 항목들을 정렬하는데 소요되는 많은 연산 비용이 필요하다.

본 논문에서는 한 번의 데이터베이스 스캔으로 빠르게 빈

※ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단(지방연구중심육성사업/충북BIT)과 BK의 지원으로 수행되었음.
† 준 회 원 : 충북대학교 BK21 PostDoc
†† 정 회 원 : 한국전자통신연구원 연구원
††† 준 회 원 : 충북대학교 전자계산학과 박사과정
†††† 종신회원 : 전남대학교 전산학과 교수
††††† 종신회원 : 충북대학교 전기전자 및 컴퓨터공학부 교수 (교신저자)
논문접수 : 2007년 3월 28일, 심사완료 : 2007년 9월 18일

DOI: 10.3745/KIPSTD.2008.15-D.1.15

발항목집합들을 발견할 수 있는 알고리즘을 제안한다. 제안하는 알고리즘은 이분할 그래프(bipartite graph) 구조를 이용한다. 본 논문에서 생성하는 이분할 그래프는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다. 본 논문에서는 지금부터 이 그래프와 제안하는 알고리즘을 각각 빈발항목 이분할 그래프(Large Items Bipartite graph, LIB-graph) 그리고 ALIB (Algorithm using LIB-graph) 알고리즘이라 명명한다.

제안하는 ALIB 알고리즘은 기존에 제안된 알고리즘과 비교하여 다음과 같은 특징을 갖는다. 첫째, 데이터베이스를 한 번만 스캔한다. ALIB 알고리즘은 처음 데이터베이스를 스캔하여 1-빈발항목들과 LIB-graph를 생성한다. LIB-graph는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다. 즉, 어떤 항목이 어떤 트랜잭션들에 포함되는지를 이 그래프를 통해 확인할 수 있다. 그러므로 대용량의 데이터베이스를 스캔하지 않고 LIB-graph를 통해 빈발항목집합들을 찾아낼 수 있다. 또한 우리의 고려 대상은 데이터베이스를 이루는 전체 항목이 아닌 오직 빈발항목이기 때문에 LIB-graph에서 1-빈발항목에 대한 엣지(edge) 만을 탐색하게 된다. 엣지를 통해 빈발항목이 포함된 트랜잭션들을 색인(index)할 수 있다. 그러므로 알고리즘의 수행이 빠르게 진행된다. 둘째, 후보항목집합들을 생성하지 않는다. 빈발항목집합들을 찾는 것은 어떤 항목집합에 대해서 그 항목집합을 포함하고 있는 트랜잭션의 개수가 미리 정의된 최소지지도를 만족하는지를 확인하는 것이다. LIB-graph에서 각 항목의 엣지를 통하여 어떤 항목집합이 전체 트랜잭션에 몇 번 포함되는지 알 수 있다. 또한, 어떤 항목집합이 같은 트랜잭션에 속해 있는지를 LIB-graph를 통하여 알 수 있다. 그러므로 후보항목집합들을 생성하지 않고도 빈발항목집합들을 찾을 수 있다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로써 연관 규칙 탐사에 대한 정의와 기존에 제안된 알고리즘들에 대해 설명한다. 3장에서는 ALIB 알고리즘의 수행 과정에 대해서 설명한다. 그리고 4장에서는 제안하는 알고리즘들과 기존에 제안된 알고리즘의 성능을 비교 분석한다. 5장에서는 결론 및 향후 연구에 대하여 논의한다.

2. 관련 연구

이 절에서는 연관규칙 탐사의 이해를 위해 알고리즘들 중에서 가장 기본적인 탐사 방법인 Apriori 알고리즘에 대해서 설명하고 후보항목집합을 생성하지 않고 빈발항목집합들을 찾아낼 수 있는 FP-growth 알고리즘에 대해 설명한다.

2.1 Apriori 알고리즘

연관 규칙 탐사 알고리즘의 가장 전형적인 방법인 Apriori 알고리즘[3]은 AprioriTid, AprioriHybrid[3]로 확장되어 연구되었다. Apriori 알고리즘은 많은 논문에서 인용되고 있으며 많은 응용에 적절히 변형되어 사용되고 있다. Apriori 알고리즘은 k-빈발항목집합 L_k를 구하기 위해서 (k-1)-빈발항목집합 L_{k-1}로부터 k-후보항목집합 C_k를 구하고 C_k의 지지도를 계산하여 최소 지지도 이상을 만족하는 L_k를 구하는 과정을 반복한다. Apriori 알고리즘의 각 단계의 진행은 데이터 항목의 증가에 따라 반복적으로 진행된다. Apriori 알고리즘은 더 이상의 후보항목집합을 생성할 수 없을 때까지 반복되어 빈발항목들을 탐사한다. 후보항목집합의 구성은 전 단계의 빈발항목집합의 조인(join) 연산과 전지 과정을 통해 생성된다. 조인 연산은 두 집합의 곱집합을 구하는 것과 같다. 전지 과정은 조인을 통해 생성된 후보항목집합의 부분 집합이 전 단계의 빈발항목집합의 원소가 아닌 경우, 그 항목은 삭제하는 과정이다. 그 이유는 전 단계에서 빈발하지 못하는 항목은 다음 단계에서도 빈발하지 못하기 때문이다. 전지 과정은 불필요한 후보항목의 수를 줄여 데이터베이스를 읽는 횟수를 감소시키기 위한 것이다.

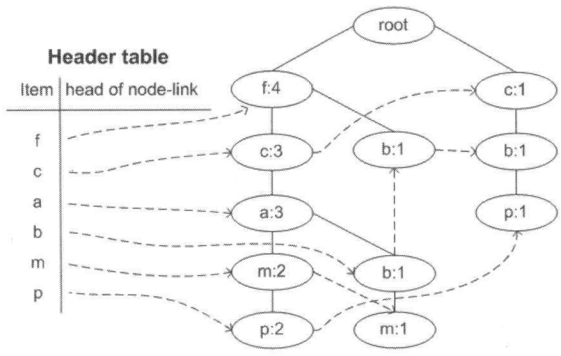
2.2 FP-growth 알고리즘

FP-growth 알고리즘[5]은 빈발 항목들에 대한 정보를 가지고 있는 FP-tree를 생성하고 최소지지도를 만족하는 빈발항목집합들을 생성한다. 또한, 데이터베이스의 반복되는 스캔을 피하고 후보항목집합들의 생성을 회피함으로써 빈발항목집합들을 구하는 비용을 줄였다.

FP-tree 생성은 먼저, 데이터베이스를 스캔하여 1-빈발항목들과 지지도를 구한다. 그리고 1-빈발항목들을 지지도에 따라 정렬하여 목록을 작성한다. 다음은 루트노드인 "null" 노드를 생성하고 데이터베이스의 각 트랜잭션을 스캔한다. FP-tree 생성은 각 트랜잭션에 대해서 1-빈발항목들의 목록에 속하지 않는 항목들을 전지하고 남은 항목인 빈발항목들은 다시 1-빈발항목들의 정렬방법과 같은 방법으로 정렬한다. 이렇게 정렬된 각 트랜잭션은 트리 생성에 이용된다. 이때, 각 트랜잭션의 항목들은 정렬된 상태이므로 먼저 발생한 항목이 부모 노드가 되고 다음 항목은 자식 노드가 되는 식으로 트리가 생성된다. <표 1>의 데이터베이스 예에 대한 FP-tree의 예는 (그림 1)에서 볼 수 있다. 생성된 FP-tree는 연관 규칙들을 생성하는데 이용된다. 생성된 FP-tree의 헤더 테이블(header table)에서 각 항목의 head of node-link에서부터 bottom-up 방식으로 링크를 따라 빈발항목집합들을 생성하게 된다. 생성된 FP-tree에서 각각의 경로는 하나의 트랜잭션을 이루는 항목들의 관계를 나타내고 있기 때문에 각 노드가 가지고 있는 지지도를 검색하여 쉽게 도출할

〈표 1〉 FP-growth 알고리즘에 대한 데이터베이스의 예[5]

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



(그림 1) FP-tree 생성의 예[5]

수 있다.

FP-growth 알고리즘은 두 번의 데이터베이스 스캔을 하지만 후보항목집합들을 생성하지 않고 빈발항목집합을 생성할 수 있는 효율적인 알고리즘이다. 그러나 FP-tree 생성 과정에서 크게 두 가지의 문제점이 있다. 첫 째, 하나의 트랜잭션을 스캔할 때 빈발하지 않는 항목들을 전지하는 연산과 전지되고 남은 빈발항목들을 정렬하는 두 가지의 연산이다. 이러한 연산은 단순히 임의의 항목들을 삽입하고 삭제하는 비교적 가벼운 연산이 아닌 비교 연산을 수행하기 때문에 상당한 시간적 비용을 수반하게 된다. 특히, 대용량 데이터베이스에서의 이러한 작업은 더욱더 큰 비용을 필요로 한다. 둘 째, 트리를 생성하기위해 트랜잭션을 스캔할 때마다 이전에 검색한 트리의 노드들을 계속적으로 반복 검색하며, 경우에 따라 공간적 비용이 커질 수 있다는 것이다. 하나의 트랜잭션을 스캔하여 트리를 갱신(update)할 때마다, 루트의 자식노드부터 비교해 가면서 같은 노드 항목이면 지지도 카운트를 하나 증가시키거나 노드 항목이 다를 때에는 새로운 노드를 생성하게 된다. 이는 트랜잭션을 스캔할 때마다 계속적으로 되풀이되는 과정이다. 또한, 공간상에서도 같은 노드 항목이 존재해도 조상이 틀리면 새로운 노드 항목으로 생성되어져야 하기 때문에 항목들의 분포에 따라 무수히 많은 노드들이 생성될 수 있다.

3. 그래프 구조를 이용한 빈발항목집합 추출

이 절에서는 한 번의 데이터베이스 스캔으로 빈발항목집합들을 찾아낼 수 있는 ALIB 알고리즘을 제안한다. 빈발항목집합을 찾는 문제는 찾고자하는 빈발항목집합이 어떤 트랜잭션 안에 존재하며 그 빈발항목집합을 포함하고 있는 트랜잭션들의 수가 사용자가 정한 최소지지도를 만족하는지를 검사하는 문제이다. 그러므로 데이터베이스를 이루는 각 항목이 어떤 트랜잭션에 존재하는지에 대한 정보를 가지고 있는 자료구조를 생성한다면 그 이후에는 데이터베이스를 스캔하지 않고 생성된 자료구조를 이용해서 빈발항목집합들을 찾아낼 수 있다.

제안하는 알고리즘은 데이터베이스에 존재하는 빈발항목과 그 빈발항목을 포함하고 있는 트랜잭션 정보를 쉽게 찾기 위해 LIB-graph라는 이분할 그래프 구조를 생성하고 이용한다. LIB-graph 는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다.

3.1 이분할 그래프 구조의 생성

이분할 그래프는 그래프 G=(V, E)에서 V(vertices)가 두 그룹 집합 X와 Y(=V-X)로 분할되어, E(edges)의 각 엣지가 X 내의 정점(vertex)과 Y 내의 정점의 쌍으로 나타내어지는 그래프이다.

본 논문에서 생성하는 LIB-graph는 이분할 그래프로서 두 개의 그룹 집합으로 이루어진다. 하나는 데이터베이스를 이루는 전체 항목이고 다른 하나는 전체 트랜잭션 식별자(Transaction IDentification, TID)이다. 두 그룹 사이에 엣지는 한 항목과 그 항목을 포함한 트랜잭션 사이에 관계를 나타낸다. 이 그래프 구조는 다음과 같은 특징을 갖는다. 첫째, LIB-graph의 정점은 두 개의 그룹 집합으로 이루어진다. 하나는 데이터베이스를 이루는 전체 항목들이고 다른 하나는 전체 트랜잭션 식별자들이다. 둘째, LIB-graph의 엣지는 두 그룹 사이의 포함 관계를 나타낸다. 즉, 한 항목과 그 항목을 포함하고 있는 트랜잭션 식별자 사이의 관계를 나타낸다. 셋째, 각 빈발 항목의 엣지의 수는 그 항목의 지지도를 나타낸다. 그래프 구조를 설명하기 위해서 [5]에서 이용되었던 <표 1>의 데이터베이스 예제를 사용하고 최소 지지도는 3으로 설정하였다. 그래프 구조 생성 방법은 다음과 같다.

데이터베이스를 스캔하여 빈발 항목들을 찾아낸다. 이때, 발생하는 빈발 항목들은 {(a, 3), (b, 3), (c, 4), (f, 4), (m, 3), (p, 3)}이다. 여기서의 숫자는 괄호안의 항목에 대한 지지도를 나타내고 표기는 편의상 알파벳 순서로 한다. 빈발 항목들을 발견하기 위해 데이터베이스를 스캔할 때 데이터베이스의 트랜잭션들이 차례대로 스캔된다. 이때, 하나의 트랜잭션을 스캔할 때마다 두 그룹 사이의 엣지가 형성된다.

그래프 생성 알고리즘에 의해서 첫 번째 트랜잭션 레코드를 스캔한다. 이때, 트랜잭션에 포함된 항목들은 {f, a, c, d, g, i, m, p}이다. 이들은 모두 트랜잭션 식별자가 100인 트랜잭션에 포함되어 있으므로 트랜잭션 식별자 그룹의 100이라는 정점으로 연결된다. 두 번째 트랜잭션에 포함된 항목들은 {a, b, c, f, l, m, o}이다. 이들은 같은 방법으로 모두 정점 200에 연결된다. 모든 트랜잭션이 스캔되면 (그림 2)와 같은 하나의 그래프 구조가 생성되고 동시에 빈발항목들이 추출된다. 그림 2에 표현된 항목들은 빈발항목들이다. 빈발항목집합 발견 과정에서 관심대상은 오직 빈발항목들이기 때문에 빈발하지 않는 항목과 그 항목의 엣지는 생략하였다.

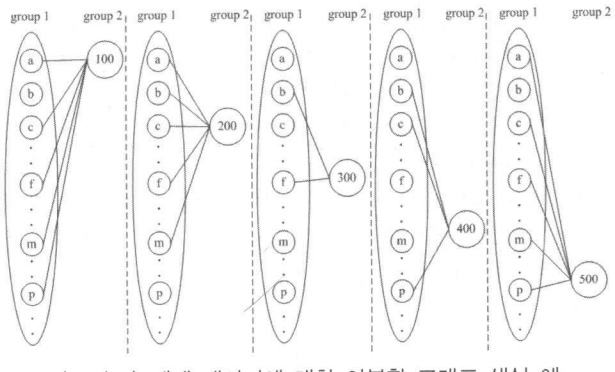
그래프 생성 알고리즘에서 알 수 있듯이, 정확히 데이터베이스를 한 번 스캔한다. 한 번의 데이터베이스 스캔으로 1-빈발항목들과 LIB-graph를 생성한다. 이 과정에서 별도의 연산을 필요로 하지 않는다. 그러므로 그래프 생성 알고리즘을 수행하는데 드는 비용은 O(|D|) 이다. 여기서 |D|는 데이터베이스를 이루는 전체 트랜잭션들의 수이다. 또한, 이 그래프는 빈발항목들과 이 빈발항목들이 포함된 트랜잭션들과의 관계를 표현하고 있기 때문에 이후의 빈발항목집합 추

출과정은 데이터베이스의 스캔 없이 그래프를 가지고 수행된다. 따라서 LIB-graph 는 데이터베이스안에 존재하는 빈발항목들과 트랜잭션들의 포함관계를 엣지를 이용하여 연결한 구조이다.

3.2 LIB-graph 를 이용한 빈발 항목집합 추출

LIB-graph에서 두 가지 중요한 특성을 발견할 수 있다. 첫 번째는 빈발 항목과 트랜잭션과의 포함관계이다. group 1은 빈발항목들의 집합이고 group 2는 Group 1에 속한 각 빈발항목을 포함하고 있는 트랜잭션 식별자들의 집합이다. Group 1에서 빈발 항목의 엣지는 그 항목을 포함하고 있는 트랜잭션 식별자에 링크되고 트랜잭션 식별자들의 엣지는 그 트랜잭션에 존재하는 항목들로 링크된다. 그러므로 어느 한 빈발항목의 엣지에 연결된 트랜잭션 식별자 엣지들을 추적하면 그 빈발항목을 포함하고 있는 모든 트랜잭션들과 그 트랜잭션에 포함된 다른 빈발항목들을 알 수 있다. 나머지 과정은 발견된 트랜잭션에서 빈발항목집합들을 찾아내는 것이다. 두 번째는 그래프의 group 1에 속하는 항목들이 정렬되어 있다는 것이다. group 1의 항목들에 대한 정렬은 지지도에 의한 것이 아니라 처음 그래프 구조를 설정할 때 미리 만들어지므로 별도의 연산 비용을 필요로 하지 않는다. 대용량의 트랜잭션 데이터베이스에서 빈발항목집합을 추출하고자 할 때, 이러한 정렬에 대한 비용은 알고리즘의 성능에 큰 영향을 준다. 따라서 본 논문에서 제안하는 알고리즘은 전체 빈발항목집합 탐사과정에서 큰 연산비용을 줄이게 된다.

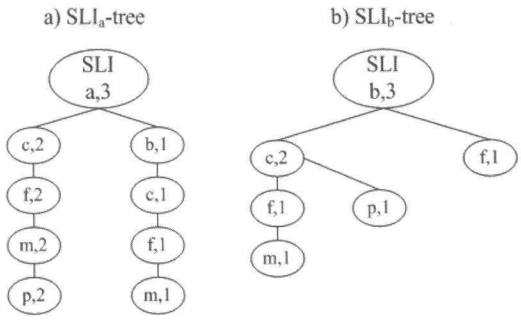
LIB-graph 탐사는 group 1에 포함된 빈발항목으로부터 시작된다. 정점 a에 대해서, 빈발 항목 a로부터 시작되는 엣지의 수는 3이다. 즉, 빈발 항목 a의 지지도는 3이라는 것을 나타내고 그 엣지로 연결된 트랜잭션은 {100, 200, 500}이다. 만약 a로 시작되는 빈발 항목집합이 존재한다면 a가 속한 트랜잭션마다 나머지 항목집합이 최소지지도 이상 존재해야만 된다. 그러므로 (그림 2)의 예에서 100, 200, 500의 트랜잭션에서 a와 함께 모두 같이 존재하는 항목들을 찾아내면 그 항목들은 a로 시작되는 빈발항목집합이 된다. (그림 2)에 보이는 것과 같이 트랜잭션 100에 링크되어 있는 엣지의 수는 a를 제외한 4개이다. 그림 상에서는 트랜잭션 100의 엣지의 수는 5개이지만 실제적으로는 더 많은 엣지를 갖는다. 그러나 우리의 관심 대상은 빈발항목들에 해당하므로 나머지 엣지들은 전지시킨다. 트랜잭션 100에 존재하는 엣지의 수는 5개이고 이들의 엣지를 따라가 보면 {a, c, f, m, p} 집합을 얻을 수 있다. 이 집합은 하나의 트랜잭션에 존재하는 모든 빈발항목들을 나타낸다. 여기의 예에서, 빈발항목 a를 포함하고 있는 여러 개의 트랜잭션 중에서 하나의 트랜잭션은 100이라는 식별자를 갖는 트랜잭션이고 그 트랜잭션에 포함된 모든 빈발항목은 a, c, f, m, p라는 것을 알 수 있다. 같은 방법으로, a가 포함된 두 번째 트랜잭션 200과 세 번째 트랜잭션 500의 엣지에서 각각 {a, b, c, f, m}과 {a, c, f, m, p}를 얻을 수 있다.



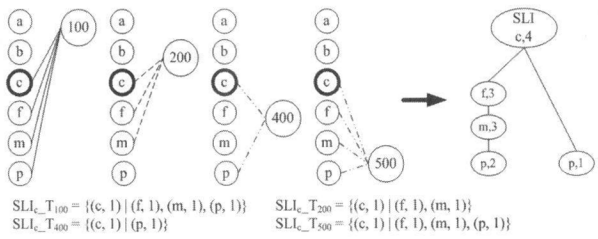
(그림 2) 예제 데이터에 대한 이분할 그래프 생성 예

추출된 빈발항목들의 집합으로부터 원하는 빈발항목집합들을 추출하기 위해 FP-tree를 응용한 트리 구조를 사용한다. [5]에서의 FP-tree는 전체 빈발항목들을 대상으로 하지만 본 논문에서의 트리는 하나의 빈발항목에 대해서 트리를 생성한다. 본 논문에서는 이 트리의 루트노드로 빈발항목을 갖는다는 의미에서 SLI_{item}-*tree(SLI, Start Large Item)*라 한다. 이때, SLI_{item}의 item은 하나의 빈발항목을 나타내고 이 항목은 트리의 루트 노드가 된다. SLI_{item}-*tree*는 위의 빈발항목 a에 대한 예와 같이 하나의 빈발항목에 대한 트랜잭션의 역 엣지를 스캔할 때 생성된다. 위의 빈발항목 a에 대한 SLI_a-*tree*의 생성은 처음, SLI_a-*tree*의 루트 노드를 생성한다. 트리의 각 노드는 항목과 그 항목의 지지도 값을 갖는다. SLI_a-*tree*의 루트 노드는 항목 a가 되고 지지도 값은 0으로 초기화된다. 빈발 항목 a와 첫 번째 엣지로 연결된 트랜잭션 100의 역 엣지를 스캔하면 트리의 첫 번째 가지(branch)가 생성된다. 그것은 루트 노드로부터 시작하는 <(a,1), (c,1), (f,1), (m,1), (p,1)>이다. 두 번째 트랜잭션 200에 대해서, 그것의 리스트는 {a, b, c, f, m}이다. 루트 노드 a를 제외한 나머지 리스트는 존재하는 패스(path)와 같지 않으므로 두 번째 가지가 생성된다. 이때, a의 지지도는 1 증가되고 생성된 가지, <(b,1), (c,1), (f,1), (m,1)>는 루트 노드 (a, 2)에 링크된다. 세 번째 트랜잭션 500에 포함된 빈발항목 리스트는 {a, c, f, m, p}이다. 이들은 첫 번째 가지와 동일한 패스이기에 그 패스를 따라서 각 노드의 지지도를 1씩 증가시키면 된다.

트리의 탐색은 루트노드로부터 생성된 자손들을 비교하면 쉽게 구할 수 있다. 이 트리는 두 개의 패스 <(a,2), (c,2), (f,2), (m,2), (p,2)>와 <(a,1), (b,1), (c,1), (f,1), (m,1)>을 생성한다. 첫 번째 패스는 전체 데이터베이스에서 두 번 나타난다는 것을 의미한다. a 항목은 데이터베이스에서 세 번 나타나지만 {c, f, m, p}와는 전체 데이터베이스에서 두 번 나타난다는 것을 의미한다. 두 번째 패스는 전체 데이터베이스에서 한 번 나타난다는 것을 의미한다. SLI_a-*tree*에서의 빈발항목집합 탐색은 전체 데이터베이스에서 빈발항목 a를 포함하는 모든 빈발항목집합들을 찾는 것이다. 두 개의 가지에서 최소지지도를 만족하는 최대 길이의 집합은 {a, c, f, m}이고 이 집합의 지지도는 3이다. 빈발항목집합의 부분집합은 역시 빈발항목집합이므로 빈발항목 a를 포함하고 있는 모든 빈발항목집합들은 {(ac), 3}, {(af), 3}, {(am), 3}, {(acf), 3}, {(acm), 3}, {(afm), 3}, {(acfm), 3}이 된다. (그림 3)은



(그림 3) *SLIa-tree*와 *SLIb-tree*의 예



(그림 4) 빈발항목 c에 대한 빈발항목집합 탐색의 예

빈발항목 a와 빈발항목 b에 대해서 LIB-graph를 탐사한 후의 생성된 트리를 보여준다. *SLI$_b$-tree*는 단말 노드로부터 세 개의 패스가 존재하지만 모두 최소지지도를 만족하지 못하기 때문에 그림에서 볼 수 있듯이 빈발항목 b로부터 생성될 수 있는 빈발항목집합은 존재하지 않는다.

정점 c에 대한 빈발 항목집합 추출 과정이 (그림 4)에 나타나있다. 빈발 항목 c는 모두 4개의 트랜잭션 {100, 200, 400, 500}에 존재한다. 그래프에서 빈발 항목 c를 포함하고 있는 트랜잭션을 스캔하여 *SLI$_c$-tree*를 생성한다. 그림 4에서 볼 수 있듯이, 각 트랜잭션에 빈발항목 a와 b에 대한 엣지가 존재하지만 이전에 이 두 개의 빈발항목에 대한 탐사를 완료했기 때문에 빈발항목 c에 대한 탐사에서는 빈발항목 a와 b에 대한 엣지에 대한 탐사를 할 필요가 없다. 그러므로 (그림 4)에서 보여 지듯이 빈발항목 c를 포함하고 빈발항목집합들은 {(cp), 3}, {(cf), 3}, {(cm), 3}, {(cfm), 3}이다. 이렇듯 빈발항목탐사 과정이 진행됨에 따라 연산해야할 항목의 수가 줄어들기 때문에 탐사과정의 후반으로 갈수록 연산은 더욱더 빠르게 진행된다. ALIB 알고리즘은 <표 2>에 기술되어진다.

4. 성능 평가

이 절에서는 본 논문에서 제안하는 ALIB 알고리즘에 대해서 최근에 제안된 FP-growth 알고리즘과의 성능을 비교한다. 성능 평가는 시간 복잡도 이론을 이용하여 두 알고리즘의 성능을 분석하고 시뮬레이션 실험을 통해 성능 측정을 수행한다.

4.1 시뮬레이션 실험 환경

모든 실험은 Microsoft Windows XP 운영체제, 1 GB 메인 메모리, 2GHz Pentium PC를 가지고 수행되었다. 프로그램은 Microsoft/Visual C++6.0을 사용하여 작성되었다. 실험에서 기록한 시간은 전체 실행 시간을 의미한다. 즉, 데이터를 입력 받을 때부터 최종 알고리즘 수행이 끝나고 결과를 출력할 때까지를 기록하였다. 그리고 실험에서 사용한 데이터는 기존의 알고리즘들에서 성능 평가를 위해 사용하였던 데이터 생성기를 사용하여 생성하였다.

실험은 두 개의 데이터 집합을 가지고 수행한다. 첫 번째 데이터 집합은 전체 항목들의 개수가 1000개인 T25.I10.D10K이다. 이 데이터 집합은 D_1으로 표시된다. 이 데이터 집합은 평균 트랜잭션 크기가 25이고 잠재적으로 가능한 최대 빈발항목집합의 크기가 10이라는 것이다. D10K는 전체 트랜잭션의 개수가 10000개라는 것을 나타낸다. 두 번째 데이터 집합은 전체 항목들의 개수가 10000개이고 전체 트랜잭션의 개수가 100000개인 T25.I20.D100K 이다. 이 데이터 집합은 D_2로 표시된다.

〈표 2〉 ALIB(mining Algorithm using LIB-graph) 알고리즘

```
Input : transaction database and minimum support(ε)
Output : frequent patterns

Procedure ALIB(transaction database, ε) {
    for each transaction do
        scan transaction
    create LIB-graph and large items
    for each large item do
        for each large item do
            scan edges of large item
            create SLI_item-tree
        if child node of SLI_item-tree IS NOT NULL then
            create path from root node of SLI_item-tree
            create combination of SLI_item-large-branch
          for each combination do
                    if MIN(support of combination) >= ε then
                                store the combination and
                            MIN(support of combination)
}
```

4.2 시간 복잡도에 의한 성능 분석

시간 복잡도 이론을 이용하여 ALIB 알고리즘과 FP-growth 알고리즘의 성능 분석을 수행한다. 시간 복잡도에 의한 성능 분석은 크게 데이터베이스를 스캔하는 비용과 새로운 데이터 구조를 생성하고 생성된 데이터 구조를 검색하는 비용으로 나눠볼 수 있다. 시간 복잡도 분석을 위해 전체 데이터베이스를 이루는 트랜잭션의 개수를 D이라 하고 각 트랜잭션에 존재하는 항목들의 평균 개수를 M개, 그리고 빈발 항목의 개수를 L개라고 가정했을 때, 두 알고리즘의 시간 복잡도는 다음과 같다.

4.2.1 FP-growth 알고리즘의 시간 복잡도

FP-growth 알고리즘의 수행은 크게 네 가지 연산으로 나누어 볼 수 있다. 첫째, 1-빈발항목들을 발견하기 위해 데이터베이스를 스캔한다. 둘째, 트리를 생성하기 위해 미리 트랜잭션을 스캔하여 빈발하지 않는 항목들을 전지하고 나머지 빈발한 항목들을 지지도에 따라 정렬시킨다. 셋째, 정렬된 각각의 트랜잭션을 스캔하여 FP-tree를 생성시킨다.

넷째, 생성된 트리를 검색하여 빈발 항목집합들을 발견한다. 이 4가지 연산에 대한 시간 복잡도는 <표 3>과 같다.

4.2.2 ALIB 알고리즘의 시간 복잡도

ALIB 알고리즘의 수행은 크게 세 가지 연산으로 요약해 볼 수 있다. 첫째, 데이터베이스를 스캔하고 그래프를 생성하는 연산이다. 이 연산에는 데이터베이스를 스캔하는 연산과 1-빈발항목들을 발견하는 연산, 그리고 그래프를 생성하는 연산이 포함된다. 이들 각각의 연산이 독립적으로 실행될 시에는 각 연산에 대한 별도의 연산비용이 필요할 것이다. 그러나 1-빈발항목들을 발견하는 것과 그래프를 생성하는 것은 미리 생성된 자료 구조에 트랜잭션들을 스캔하여 읽어진 항목들을 링크 시키면 되기 때문에 별도의 연산 비용이 필요 없는 연산들이다. 그러므로 데이터베이스를 스캔하는 시간 복잡도만으로 그래프 생성 연산까지 해결이 가능하다. 둘째, 그래프를 검색하여 트리를 생성하는 연산이다. 셋째, 생성된 트리를 검색하여 한 항목에 대한 빈발 항목집합들을 발견하는 것이다. 이 세 가지 연산에 대한 시간 복잡도는 <표 4>와 같다.

<표 3>과 <표 4>를 비교했을 때, FP-growth 알고리즘과 ALIB 알고리즘은 생성하는 데이터 구조나 방법은 조금 다르지만 새로운 데이터 구조를 생성하고 그 데이터 구조를 이용하는 등의 비슷한 연산과정을 수행한다는 것을 알 수 있다. 그러나 ALIB 알고리즘은 최소한 FP-growth 알고리즘에서 수행하는 빈발하지 않는 항목들을 전지하고 난 후 나머지 빈발항목들을 정렬하는 연산을 수행하지 않는다. 그러므로 각 트랜잭션에서 빈발항목들을 지지도에 따라 정렬해야하는 연산 비용만큼 좋은 성능을 보인다는 것을 알 수 있다.

4.3 시뮬레이션 실험에 의한 성능 측정

FP-growth 알고리즘은 최근에 제안된 효과적인 알고리즘이다. FP-growth 알고리즘은 데이터베이스의 스캔횟수를 줄이기 위해 FP-tree라는 새로운 구조를 제안하였다. 데이터베이스를 최소한 스캔하여 FP-tree를 생성하고 이후의 빈발 항목집합 추출과정은 이 트리를 이용함으로써 데이터베이스 스캔을 최소화 하였다.

실험 결과에서 알 수 있듯이 FP-growth 알고리즘과 본 논문에서 제안하는 ALIB 알고리즘이 빈발 항목집합을 추출하는데 매우 효과적이다. (그림 5)와 (그림 6)은 각각 D_1과 D_2데이터집합을 가지고 수행한 성능 평가의 결과를 보여준다. 그림에서 볼 수 있듯이 FP-growth 알고리즘과 ALIB 알고리즘이 좋은 성능의 알고리즘이라는 것을 보여준다. 그러나 제안하는 ALIB 알고리즘의 성능이 더 좋다는 것을 그림에서 볼 수 있다.

두 알고리즘의 성능상의 차이는 크게 데이터베이스 스캔횟수와 새로운 구조를 생성하는데 드는 연산 비용이 FP-growth 알고리즘보다 ALIB 알고리즘이 더 적기 때문일 것이다. FP-growth 알고리즘 수행에서 주로 차지하는 컴퓨팅 비용은 데이터베이스를 스캔하고 빈발 항목들의 지지도에 따라 트랜잭션을 정렬하는 것과 정렬된 트랜잭션을 다시 스캔하여 FP-tree를 생성해 나가는 것이다. FP-growth 알고리즘이 후보 항목집합을 생성하지 않고 데이터베이스를 적게 스캔하지만 대용

〈표 3〉 FP-growth 알고리즘 시간 복잡도

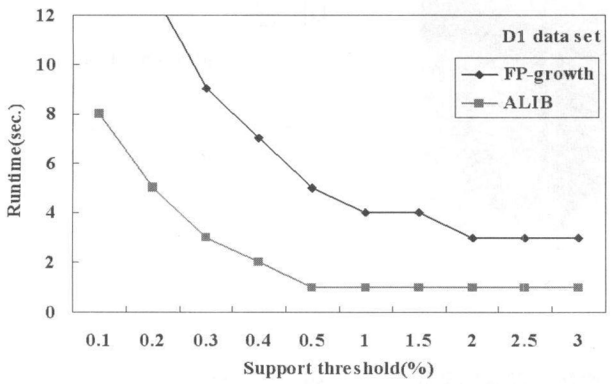
트랜잭션의 개수 : D, 각 트랜잭션의 평균 항목 수 : M, 빈발항목의 수 : N

main operation	sub operation	description	time complexity
데이터베이스 스캔	database scan	빈발항목 발견 연산을 위한 데이터베이스 스캔	O(D × M)
항목 전지 및 정렬	database scan	전지와 정렬을 위한 데이터베이스 스캔	O(D × M)
	non-frequent item pruning	빈발하지 않는 항목 전지 연산	O(D × M)
	frequent item sorting	전지되고 남은 빈발항목 정렬 연산	O(D× LlogL)
트리 생성	FP-tree generation	FP-tree 생성 연산	O(D × M)
빈발항목집합 추출	FP-tree scan	빈발항목집합을 발견하기 위한 FP-tree 스캔	O(L × logL)
	frequent pattern mining	추출된 빈발 시퀀스로부터 빈발항목집합 추출	O(L2)

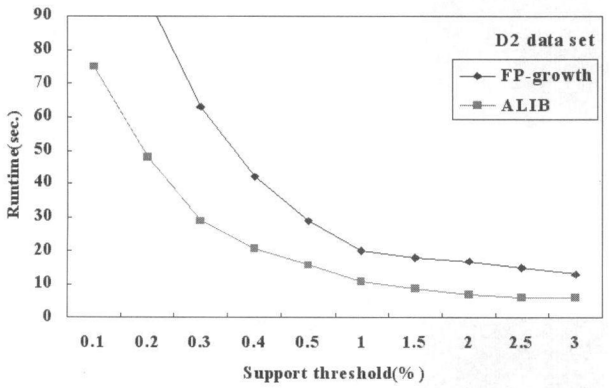
〈표 4〉 ALIB 알고리즘 시간 복잡도

트랜잭션의 개수 : D, 각 트랜잭션의 평균 항목 수 : M, 빈발항목의 수 : N

main operation	sub operation	description	time complexity
데이터베이스 스캔	database scan	빈발항목 발견과 그래프 생성을 위한 데이터베이스 스캔	O(D × M)
트리 생성	graph scan	트리 생성을 위한 그래프 스캔	O(L × D)
	non-frequent edge pruning	빈발하지 않는 항목에 대한 엣지들의 전지 연산	O(D × M)
	tree generation	SLI-tree의 생성	O(L × M)
빈발항목집합 추출	SLI-tree scan	빈발항목집합을 발견하기 위한 SLI-tree 스캔	O(L × logL)
	frequent pattern mining	추출된 빈발 시퀀스로부터 빈발항목집합 추출	O(L2)



(그림 5) FP-growth와 ALIB의 성능 비교



(그림 6) FP-growth 와 ALIB의 성능 비교

량의 데이터베이스에서 빈발하지 않는 항목들을 전지하고 빈발한 항목들을 정렬하는데 드는 비용은 여전히 많은 비용을 차지한다.

5. 결론 및 향후 연구 방향

임의의 k-빈발항목집합은 하나의 트랜잭션에 존재하며 빈발항목의 개수가 k 개인 항목들의 집합을 의미한다. 우리는 빈발항목집합을 생성하기 위해 데이터베이스를 이루는 모든 트랜잭션들을 검사한다. 빈발항목집합들을 발견하는 문제는 데이터베이스에서 어떤 항목집합이 최소지지도 이상으로 트랜잭션에 존재하는가를 찾는 문제라 정의할 수 있다.

빈발항목집합들을 발견하는 과정에서 데이터베이스를 스캔하는 것은 필수적이다. 그러나 대용량의 데이터베이스의 스캔 횟수가 많으면 많을수록 빈발항목집합을 찾는 연산 비용은 증가하게 된다. 그러므로 기존의 알고리즘들은 대용량의 데이터베이스를 스캔하는 횟수를 줄이는데 초점을 맞추고 있다.

데이터베이스의 스캔횟수를 줄이고 후보항목집합을 생성하지 않음으로써 많은 성능 개선을 보인 대표적인 알고리즘이 FP-growth 알고리즘이다. 이 알고리즘은 데이터베이스를 두 번 스캔한다. 그리고 FP-tree라는 데이터베이스를 압축한 형태의 자료구조를 생성한다. 이후의 빈발항목집합 추출과정은 이 FP-tree를 가지고 수행함으로써 알고리즘 수행이 빠르다. 그러나 FP-tree를 생성하기 위해서 데이터베이스의 각 트랜잭션을 조작하는데 많은 컴퓨팅 비용을 요구한다.

본 논문에서는 대용량 데이터베이스를 스캔횟수를 줄이고, 빈발항목집합을 빠르게 생성하기 위한 ALIB 알고리즘을 제안하였다. ALIB 알고리즘은 데이터베이스를 단지 한 번만 스캔함으로써 원하는 빈발항목집합을 찾아낼 수 있다. 이러한 연산은 데이터베이스를 한 번 스캔하여 생성된 LIB-graph를 가지고 가능해진다. LIB-graph는 빈발항목과 트랜잭션과의 관계를 알 수 있는 그래프 구조이다. 데이터베이스에서 우리가 찾고자 하는 것은 빈발항목집합이다. 이 빈발항목집합은 빈발항목으로 이루어져있다. 데이터베이스의 각 트랜잭션을 스캔하고 그 트랜잭션을 이루는 항목들이 빈발한지를 확인하기 위한 연산에 많은 비용을 소비한다. ALIB 알고리즘은 빈발항목에 대한 정보를 가지고 있는 LIB-graph를 이용함으로써 빈발하지 않는 연산에 불필요한 항목들에 대한 연산 비용을 줄일 수 있다. 또한, ALIB 알고리즘은 후보항목집합을 생성하지 않는다. 기존의 알고리즘에서 빈발항목집합 탐사과정의 가장 많은 연산 비용을 차지했던 부분이 후보항목집합의 생성과 그들이 빈발한지 안하는지를 확인하는 작업이었다. ALIB 알고리즘은 후보항목집합을 생성하지 않으므로 빈발항목집합을 찾아내기 위한 연산 비용을 크게 줄일 수가 있었다. 시뮬레이션 실험을 통하여 제안한 알고리즘이 FP-growth 알고리즘 보다 우수한 성능을 갖는다는 것을 보였다.

ALIB 알고리즘이 시간상에서는 우수하다고 할 수 있으나 FP-growth보다 공간상의 문제를 가질 수 있다. FP-growth 알고리즘에서 생성하는 FP-tree의 최대 깊이는 M이고 자식 노드의 최대 개수는 L이므로 FP-growth 알고리즘의 공간복잡도는 $O(LM^2)$이다. ALIB 알고리즘에서, LIB-graph를 생성하는 것의 공간복잡도는 전체 트랜잭션의 개수 D와 빈발항목의 개수 L에 대한 O(D+L)이다. 그리고 하나의 SLI-tree를 생성하는데 공간복잡도는 트랜잭션을 이루는 항목들의 평균 개수가 M개 이므로 $O(M^2)$이다. 하나의 SLI-tree는 휘발성으로, 빈발항목집합을 찾는 일이 종료되면 메모리에서 사라지기 때문에 SLI-tree에 대한 전체 공간복잡도는 $O(M^2)$이다. 따라서, D의 개수와 L의 개수가 커지면 커질수록 ALIB 알고리즘이 FP-growth보다 효율적이지 못할 수 있다. 향후엔 더 빠른 성능의 알고리즘에 대한 연구와 메모리의 효율성을 높일 수 있도록 연구가 진행되어야 할 것이다.

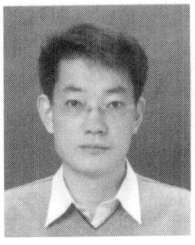
참 고 문 헌

[1] R. Agrawal, C. Aggarwal and V. V. V. Prasad, "A tree projection algorithm for generation of frequent itemsets," In Journal of Parallel and Distributed Computing, Volume 61, Issue 3, pp.350-371, March, 2001.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," In Proceedings of the ACM SIGMOD, Washington D.C., pp.207-216, May, 1993.

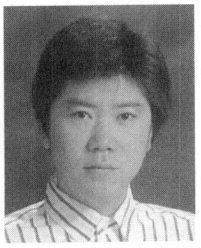
[3] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," In Proceedings of the VLDB, Santiago, Chile, pp.487-499, September, 1994.

[4] G. Grahne, L. Lakshmanan and X. Wang, "Efficient mining of constrained correlated sets," In Proceedings of the ICDE, pp.512-521, February, 2000.
[5] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," In Proceedings of the ACM SIGMOD, pp.1-12, June, 2000.
[6] M. Klemettinen, h. Mannila, P. Ronkainen, h. Toivonen and A.I. Verkamo, "Finding interesting rules from large sets of discovered association rules," In Proceedings of the CIKM, pp.401-408, November, 1994.
[7] B. Lent, A. Swami and J. Widom, "Clustering association rules," In Proceedings of the ICDE, pp.220-231, April, 1997.
[8] B. Liu, W. Hsu and Y. Ma, "Mining association rules with multiple minimum supports," In Proceedings of the ACM SIGKDD, pp.337-341, August, 1999.
[9] R. Ng, L. V. S. Lakshmanan, J. Han and A. Pang, "Exploratory mining and pruning optimizations of constrained associations rules," In Proceedings of the ACM SIGMOD, pp.13-24, June, 1998.
[10] J.S. Park, M.-S. Chen, and P.S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," In Proceedings of the ACM SIGMOD, pp.175-186, May, 1995.
[11] S. Sarawagi, S. Thomas and R. Agrawal, "Integrating association rule mining with relational database systems: Alternatives and implications," In Proceedings of the ACM SIGMOD, pp. 343-354, June, 1998.
[12] A. Savasere, E. Omiecinski and S. Navathe, "An efficient algorithm for mining association rules in large databases," In Proceedings of the VLDB, pp.432-444, September, 1995.
[13] R. Srikant, Q. Vu and R. Agrawal, "Mining association rules with item constraints," In Proceedings of the Knowledge Discovery and Data Mining, pp.67-73, August, 1997.



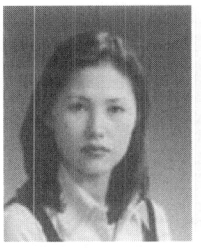
채 덕 진

e-mail : djchai2520@hotmail.com
1999년 동신대학교 컴퓨터학과(학사)
2001년 전남대학교 대학원 전산통계학과(석사)
2006년 전남대학교 대학원 전산학과(박사)
2006년~현 재 충북대학교 BK21 PostDoc
관심분야 : 데이터 마이닝, 스트림 데이터 마이닝, 멀티미디어 데이터베이스



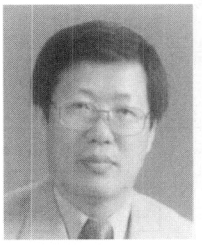
김 룡

e-mail : kimlyong@dblab.chungbuk.ac.kr
2000년 연변과학기술대학교 전자전산학과(학사)
2003년 충북대학교 대학원 전자계산학전공(석사)
2007년 충북대학교 대학원 전자계산학전공(박사)
2007년~현 재 한국전자통신연구원 연구원
관심분야 : 스트림 데이터 마이닝, 시공간 데이터 마이닝, 시공간 데이터 베이스, 스트림 데이터 처리, 센서 데이터 처리



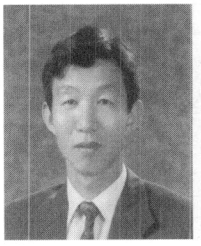
이 용 미

e-mail : ymlee@dblab.chungbuk.ac.kr
2002년 충북대학교 컴퓨터과학과(학사)
2005년 충북대학교 전자계산학과(석사)
2005~현 재 충북대학교 전자계산학과 박사과정
관심분야 : 시공간 데이터베이스, 스트림 질의 처리, 스트림 데이터 마이닝



황 부 현

e-mail : bhhwang@chonnam.chonnam.ac.kr
1978년 숭실대학교 전산학과(학사)
1980년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1980~현 재 전남대학교 전산학과 교수
관심분야 : 분산시스템, 분산 데이터베이스 보안, 객체지향 시스템, 전자상거래, 스트림 데이터 마이닝



류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr
1976년 숭실대학교 전산학과(이학사)
1980년 연세대학교 산업대학원 전산전공(공학석사)
1988년 연세대학교 대학원 전산전공(공학박사)
1976~1986년 육군군수 지원사 전산실(ROTC 장교), 한국전자통신연구원(연구원), 한국방송통신대학교 전산학과(조교수) 근무
1989년~1991년 Univ. of Arizona Research Staff (TempIS 연구원, Temporal DB)
1986년~현 재 충북대학교 전기전자 및 컴퓨터공학부 교수
관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 유비쿼터스 컴퓨팅 및 스트림 데이터 처리, 지식기반 정보검색 시스템, 데이터베이스 보안, 데이터 마이닝, 바이오인포메틱스

스트리밍 XML 데이터의 빈발 구조 마이닝

황 정 희†

요 약

유비쿼터스 환경에서 상황정보 인식 분야를 연구하면서 가장 밑바탕에서 기초가 될 수 있는 것은 인터넷 기술과 XML(Extensible Markup Language)이다. 인터넷을 통한 통신에서 XML 데이터의 사용이 일반화되고 있으며 데이터의 형태는 연속적이다. 그리고 XML 스트림 데이터에 대한 질의를 처리하기 위한 방안들이 제시되고 있다. 이 논문에서는 스트림 데이터에 대한 질의처리를 효율적으로 수행하기 위한 기반연구로써 XML을 레이블의 순서화된 트리로 모델링하여 온라인 환경에서 빈발한 구조를 추출하는 마이닝 방법을 제안한다. 즉, 지속적으로 입력되는 XML 데이터의 구조를 트리로 모델링하고 각각의 트리를 하나의 트리 집합의 구조로 표현하여 현재 윈도우 시점에서 빈발한 구조를 정확하고 빠르게 추출하는 방법을 제시한다. 제시하는 방법은 XML의 질의 처리 및 색인 구성의 기초 자료로 활용될 수 있다.

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

Mining of Frequent Structures over Streaming XML Data

Jeong Hee Hwang†

ABSTRACT

The basic research of context aware in ubiquitous environment is an internet technique and XML. The XML data of continuous stream type are popular in network application through the internet. And also there are researches related to query processing for streaming XML data. As a basic research to efficiently query, we propose not only a labeled ordered tree model representing the XML but also a mining method to extract frequent structures from streaming XML data. That is, XML data to continuously be input are modeled by a stream tree which is called by XFP_tree and we exactly extract the frequent structures from the XFP_tree of current window to mine recent data. The proposed method can be applied to the basis of the query processing and index method for XML stream data.

Key Words : Stream Data, XML Data, Frequent Structure Extraction, XML Mining

1. 서 론

유비쿼터스는 문자 그대로 해석하면 모든 곳에 존재한다는 의미이다. 유비쿼터스 환경에서 상황정보 인식 분야를 연구하면서 가장 밑바탕에서 기초가 될 수 있는 것이 인터넷 기술과 XML(Extensible Markup Language)이다. 현재의 인터넷 서비스는 제한적으로 그것에 접근 할 수 있는 단말기와 모뎀만 있다면 언제 어디서든 원하는 서비스를 얻을 수 있는 충분한 인프라를 갖추고 있기 때문이다. 또한 서로 다른 기반의 서비스 간에도 통신이 가능하다는 장점이 있다[1, 2, 3].

XML은 HTML의 개념에서 한 단계 더 나아가 미래의 웹 서비스를 주도할 중요한 도구로서 유동적으로 정보의 포맷을 만들고 그것과 함께 해당 데이터를 공유할 수 있게 한다. 예를 들면 자동차의 생산자가 자동차에 대한 정보를 어떤 정해진 형식에 맞추어서 생성하면 그것이 어떤 "형식"에 부합하므로 어느 곳에서나 그 정보를 유효한 값으로 받아 볼 수 있고 또 다른 자동차와도 쉽게 비교가 가능해 질 수 있다. 다시 말해 XML은 마크업 심볼을 임의로 정의할 수도 있으며, 단순이 이미지나 텍스트를 보여주는 것이 아니라 XML 자체가 데이터가 되어 다른 개체에 저장되거나 연산을 수행할 수 있다. XML은 위의 예와 같이 어떤 서비스든지 그 서비스를 기술하는 방법을 제공하는 유연성이 있으므로 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있다. 그리고 현재 유비쿼터스 환경을 위한 기초 데이터로써 XML의 사용은 일반화되고 있고 데이터의 입력은 연속적인 형태(stream data)이다[4, 5, 6].

스트림 데이터(stream data)는 전자상거래 사이트의 웹로그, 네트워크의 트래픽을 감시하는 모니터링 시스템, 유비쿼터스 환경 등에서 사용하는 센서 네트워크를 통해 수집 가능한 데이터로써, 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 그대로

※이 논문은 2007학년도 남서울대학교 학술 연구비 지원에 의하여 연구되었음
† 정 회 원 : 남서울대학교 컴퓨터학과 전임강사
논문접수 : 2007년 5월 18일, 심사완료 : 2007년 9월 24일

DOI: 10.3745/KIPSTD.2008.15-D.1.23

사용하기는 힘들다. 최근 들어 많은 연구자들이 스트림 데이터의 처리 기술 개발에 힘써 왔고, 다양한 시스템들이 개발되고 있다.

기존의 응용 프로그램에서는 데이터 원본들이 주로 디스크나 테잎에 데이터가 들어있다는 관점하에 이들에 대한 순차 접근이나 임의 접근이 가능하다고 보았다. 하지만 최근에 이르러 많은 응용분야에 있어서 이러한 관점이 더 이상 유효하지 않게 되었다. 따라서 기존의 데이터 원본에 대한 관점이 유효하지 않은 이러한 분야에서는 질의 처리나 데이터 마이닝에 관한 새로운 많은 이슈(issue)들이 발생하게 되었는데, 특히 데이터 마이닝과 관련된 이러한 이슈들에 대한 연구를 스트림 데이터 마이닝 이라 부른다. 이러한 응용분야의 예로 네트워킹 분야를 들 수 있는데 이 분야는 라우터와 같은 굉장히 많은 네트워크 구성 요소들과 관련된 프로토콜 및 데이터 교환 트래픽들로 구성되며, 통행량의 조절, 장애처리, 용량 모니터 및 조절, 침입탐지, 주문형 회계와 청구서 발송 등의 작업을 요구한다[8, 9, 10].

이러한 응용 분야에서도 데이터가 계속해서 끊임없이 도착하는데, 이러한 데이터의 특징은 데이터 도착 속도가 가변적이며 많은 소스들이 데이터를 전송하고 또한 데이터의 사이즈가 아주 거대해서 무한한 데이터 스트림(infinite data stream)을 형성하기 때문에 디스크나 테잎 같은 곳에 저장을 할 수 없다는 것이다. 이러한 스트림 데이터를 처리하기 위하여 새로운 스트림 데이터 모델이 요구된다. 특히, 웹 데이터의 표준인 XML의 사용은 일반적인 기준이 되고 있다. 그러므로 XML 스트림 데이터에 대한 질의처리 및 이를 위한 색인기법에 대한 연구가 수행되고 있다[5, 7, 8]. 그러나 빠르고 효율적인 질의 처리 및 색인구성을 위해서는 XML 데이터의 구조 분석이 반드시 필요하다. 이 논문에서는 슬라이딩 윈도우 기법을 기반으로 스트리밍 XML 데이터로부터의 빈발하게 발생하는 구조를 효율적으로 추출하는 마이닝 방법을 제안한다. 개별적인 XML 데이터 트리로부터 빈발한 구조를 추출하지 않고 현재 윈도우 시점에서의 XML 트리들을 하나의 트리로 표현하여 빈발구조를 추출하므로 윈도우 이동시점에서 삭제되는 트랜잭션의 XML 트리의 구조에 해당하는 노드들을 별도로 찾아서 빈발도를 갱신하는 과정을 거치지 않고 단지 윈도우 이동에 따른 트리노드의 빈발도를 일괄적인 이동에 의해 갱신을 수행하므로 정확하고 빠른 마이닝 결과를 얻을 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 데이터 스트림에 대한 관련 연구를 기술하고 3장에서는 입력되는 스트리밍 XML 데이터의 표현모델을 설명한다. 그리고 4장에서는 이 논문에서 제안하는 스트리밍 XML 데이터의 마이닝 방법을 기술한다. 5장에서는 제안모델에 대한 실험 결과를 보이고 6장에서 결론을 맺는다.

2. 관련연구

최근 인터넷의 급격한 발달과 유비쿼터스 컴퓨팅 환경 그리고 센서 네트워크와 같은 많은 정보들의 교환이 이루어지는 환경에서 무한의 연속적인 데이터의 전송 처리가 요구된다. 스트림 데이터 마이닝과 관련하여 군집화 알고리즘, 분류 알고리즘, 연관 규칙 알고리즘 등이 최근에 발표되었다[5, 6, 7, 9, 11]. [9]는 최소 지지도를 만족하는 아이템 셋을 계산하되 결과 중에는 사용자가 허락하는 범위 내의 오차가 있는 결과도 포함될 수 있도록 하였다. 즉, 이 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.

[7]에서는 NiagaraCQ에 대하여 언급한다. 많은 질의들이 비슷한 구조를 공유한다는 사실에 기반하여 연속 질의를 점증적으로 그룹핑 한다. 또한 새로운 데이터가 도착했을 때 실행되는 질의(arrival-based query)나 특정 실행 간격을 지니는 질의(interval-based query) 등의 두 가지 경우 모두를 지원한다. 그러나 XML 문서 스트림에 대한 다중 연속 질의 최적화에 초점을 두었다. 또한 이와 관련하여 [10]은 XML 스트리밍 데이터로부터 빈발 트리를 분석하기 위해 가상 트리(virtual tree)에서 오른쪽으로 노드를 확장하는 방법(rightmost expansion)을 이용하는 온라인 알고리즘인 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.

[8]은 XML에 대한 빈발 질의 패턴을 발견하기 위해 Apriori 기반의 후보 패턴을 생성한다. 그러나 루트화된 서브트리만을 추출하므로 일반적인 스트림 데이터에 대한 마이닝 기법으로 보기 어렵다. [12]는 FP-tree를 변형한 DSTree [13]와 유사한 방법으로 트리를 구성하는 방법을 제안하였다. 그러나 [12]는 점진적 마이닝을 위해 제안한 트리 구조이며 DSTree는 스트림 데이터를 마이닝 하기 위해 트리를 구성하기 때문에 대상으로 하는 데이터가 다르므로 트리의 구성 및 항목에 대한 빈발도 유지 방법도 다르다.

데이터 스트림에 대한 연구는 국내에서도 꾸준히 진행되고 있다. [15]에서는 새로운 트랜잭션이 지속적으로 추가되는 환경에서 최신의 유용한 정보를 추출하기 위한 방법을 제안하고 있다. 이를 위해 새롭게 출현한 항목에 대한 지연 추가와 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 기법을 오토마타를 이용하는 방법들도 제안되고 있다.

비순서화된 스트림에서 입력 튜플을 효율적으로 정렬하고

저장하기 위한 방법 및 슬라이딩 윈도우의 생성 시기를 결정하기 위한 평균-기반 추정 방식을 [16]에서 제안하였다. 또한 XML 스트림에 대한 동적인 질의 계획을 제안하는 [17]에서는 관계 데이터 모델을 사용하는 시스템의 적응력있는 질의 처리 모델을 적용하는 방법을 제안하였다.

이렇게 XML 스트림 데이터을 처리하기 위한 질의 계획 및 빈발 질의 패턴을 발견하기 위한 연구가 진행되고 있는 반면에 스트리밍 XML 데이터로부터 빈발한 패턴을 탐색하기 위한 마이닝 기법은 미진한 상태이다. 그러므로 스트리밍 XML 데이터에 대한 빈발 구조의 분석 즉, 유비쿼터스 환경에서 기초가 되는 XML 스트림 데이터에 대한 마이닝 기법의 연구가 반드시 필요하다. 이 논문에서 제안하는 트리구조를 이용한 마이닝 방법은 DSTree와 유사한 방법이지만 DSTree는 구조없는 스트림 데이터를 대상으로 마이닝하기 위한 트리 구조를 나타내고 있으며, 이 논문에서 제안하는 XFP_tree는 스트리밍 XML 데이터에 대한 마이닝을 위해 제안된 트리이다.

3. XML 스트림 데이터의 트리 모델

구조적 특징을 갖는 XML 데이터는 일반적으로 트리 모델로 표현한다. 즉, XML 데이터를 트리로 표현하여 노드들에 대한 레이블 및 순서를 부여한 트리(labeled ordered tree)의 구조로 나타낸다. 트리로 구성되는 XML 데이터 스트림은 연속적으로 입력되므로 무한한 노드들의 시퀀스이다.

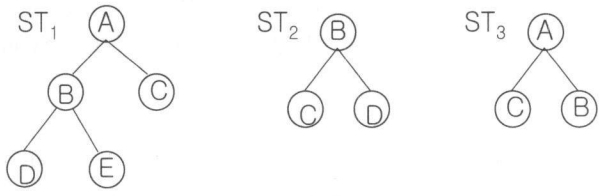
XML 데이터를 트리로 모델링하여 마이닝을 수행할 때 일반적으로 각 XML 데이터에 대한 트리에서 빈발한 패스(path) 정보를 발견한다. 이 논문에서도 연속적으로 입력되는 XML 스트림 데이터를 트리구조의 집합을 표현하는 하나의 트리를 구성하여 빈발한 패턴을 발견한다. 이 논문에서 제안하는 스트림 데이터의 트리를 XFP_tree(XML Frequent Pattern tree)라 하고, XFP_tree는 루트를 가진 서브트리들을 포함하는 하나의 트리집합이다. 즉, XFP_tree는 지속적으로 입력되는 시퀀스 트리들을 하나의 트리로 표현하는 XFP_tree= {ST_1, ST_2, .. ST_n}의 의미를 포함한다. 여기서 ST_i는 입력되는 XML 트리를 의미하며, n은 가장 최근에 생성된 트리의 식별자를 의미한다.

스트림 데이터에 대한 트리 XFP_tree를 구성하는 방법은 다음과 같다.

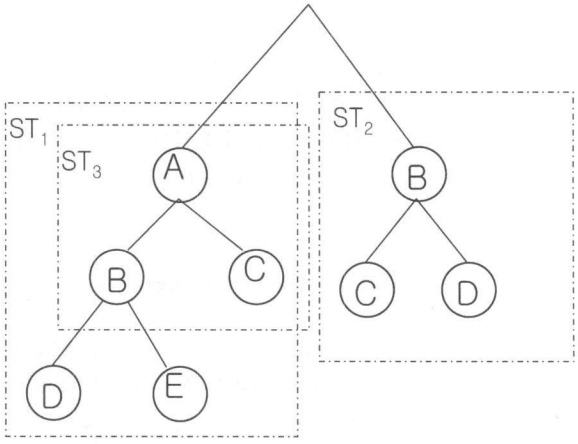
첫째, 처음으로 입력되는 루트를 포함하는 XML 데이터에 대한 기초 트리 XFP_tree를 구성한다.

둘째, 새롭게 입력되는 트리의 루트와, 같은 루트를 포함하는 서브트리가 존재하면 기존의 서브트리에 결합하지 않고 개별적인 서브트리를 구성한다. 이것은 XML는 루트부터 자신의 노드까지의 패스(path)가 구조를 구분하는 중요한 의미를 갖기 때문이다. 그리고 트리를 구성할 때 형제간의 노드 사이에는 순서를 고려하지 않는다. 이것은 빈발 구조의 발견에서 형제노드의 순서는 중요하지 않기 때문이다.

(그림 1)은 연속적으로 입력되는 XML 스트림 데이터



(그림 1) XML 스트림 데이터 트리 표현



(그림 2) 스트림 데이터 집합을 표현하는 트리(XFP__tree)

ST_1, ST_2, ST_3을 트리로 표현한 것이다.

(그림 2)는 (그림 1)에 있는 스트림 데이터를 하나의 트리(XFP_tree)로 구성한 예를 보여준다. ST_1, ST_3은 루트노드 A가 같기 때문에 같은 루트를 갖는 서브트리로 구성되며, ST_2는 루트 노트 B를 갖는 서브트리로 구성된다.

XML 데이터에 대한 빈발 트리를 [8]에서는 루트를 가진 서브트리(rooted tree)가 전체 트리에 대한 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. 즉, XML 데이터에서 에지는 데이터를 구성하는 기초단위로써, XML의 구조 정보를 나타내는 중요한 요소이며 루트부터 특정 노드까지의 path 정보는 에지를 확장한 XML의 구조를 나타내는 중요한 의미를 지닌다.

XML 스트림 데이터에 대해 마이닝을 수행하기 위한 기본 개념을 다음과 같이 정의한다.

정의 1. XML 스트림 데이터를 표현하는 트리는 XFP_tree = $<E_1, E_2,..E_n>$의 에지들로 구성되며, 스트림 데이터 집합인 XFP_tree의 에지는 부모노드와 자식노드의 순서관계 E_i = $<P_i, C_i>$를 유지한다.

스트림 데이터를 표현하는 트리 XFP_tree는 루트노드가 같으면서 부모노드와 자식노드 관계가 같으면 에지는 공통이므로 하나의 에지로 표현한다. 즉, (그림 2)에서 ST_1, ST_3의 <A, B>, <A, C>는 공통의 에지로써 하나의 에지로 표현된다. 그러나 ST_1, ST_2의 에지<B, D>는 루트 노드가 다르므로 해당 루트의 자식으로 각각 표현된다.

정의 2. XML 트리에서 루트 노드 n_1부터 특정 노드 n_i까지의 연속된 에지들의 구조정보를 XML의 SPath라고 하고, SPath=<n_1, n_2,..., n_i>는 노드들의 레벨에 따른 순서를 고려한다.

부모와 자식관계의 노드를 형성하는 두 개의 노드간의 관계를 나타내는 에지를 확장한 개념으로써 에지들 간의 순서적 개념을 고려함으로써 XML의 구조에 대한 레벨정보를 고려하여 빈발 구조를 발견하는 중요한 기준이 된다.

정의 3. XML 트리를 구성하는 에지 $EI=<P_i, C_i>$가 XML 스트림 데이터에서 발생하는 빈도 $XSup(E_i)$는 스트림 트리 XFP_tree에 포함되어 있는 트리의 발생 빈도의 합이다. 이에 대한 식은 다음과 같이 나타낸다.

$$XSup(Ei) = |Ei| / |XFP\_tree|$$

여기서 |XFP_tree|는 XML 스트림 데이터 트리 XFP_tree에 포함되어 있는 트리의 수를 의미하며, $|E_i|$는 트리에 나타나는 에지의 발생빈도를 의미한다. 에지는 각 XML 데이터를 나타내는 트리에서 유일한 노드의 관계로 구성되며, 주어진 사용자 지지도(min_sup = $0<\Theta<1$)를 만족하면 $XSup(E_i) \geq$ min_sup 이므로 에지 E_i는 빈발하다는 것을 의미한다. (그림 2)에서 에지 <A, B>의 빈발도는 2/3(0.66)이다.

스트림 데이터를 다루기 위한 슬라이딩 윈도우는 시간을 기반으로 윈도우의 크기를 결정하는 시간 기반 슬라이딩 윈도우와 튜플을 기반으로 윈도우 크기를 결정하는 튜플 기반 슬라이딩 윈도우가 있다[14]. 시간 단위로 윈도우 크기를 정의하는 경우에는 트랜잭션의 발생 시간을 고려하여 현재 범위에 포함되는 유효 정보 여부가 결정되며, 튜플 개수 단위로 윈도우 크기가 정의되는 경우에는 트랜잭션 발생 순서를 고려하여 현재 윈도우 범위에 포함되는 유효 정보 여부가 결정된다. 구조정보를 가지는 스트리밍 XML 데이터는 일반적인 스트림 데이터에서 튜플과 같이 단순한 속성(attribute)과 값(value)의 쌍으로 이루어지지 않으며, 단순하게 시간간격을 기준으로 나눠질 수 없다. 그러므로 XML 스트림 데이터는 문서를 트랜잭션으로 고려하는 구조정보의 기준에 의한 트랜잭션 수로 나눈다. 이 논문에서도 윈도우 크기를 바탕으로 현재 윈도우 범위에 속하는 모든 XML 구조에 대한 탐색을 수행하기 위하여 트랜잭션의 개수로서 윈도우 크기 정의하고 고정된 윈도우 크기를 사용자가 지정할 수 있다. 그리고 이 방법은 시간단위로 정의되는 경우에도 응용하여 적용할 수 있다.

4. 스트리밍 XML 데이터 마이닝

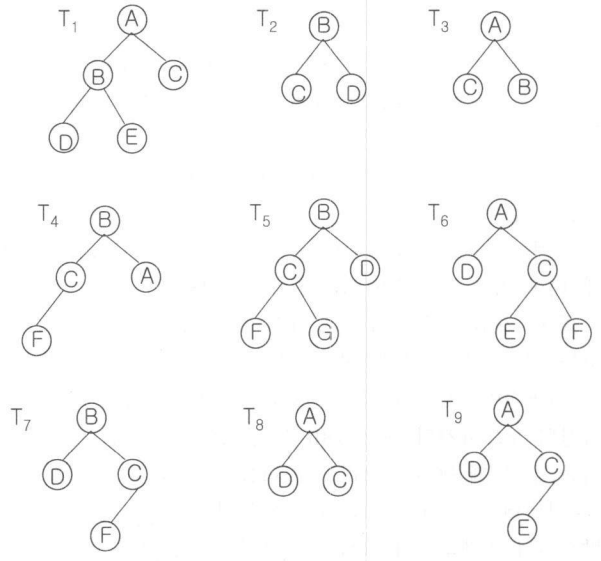
연속적으로 입력되는 스트림 데이터로부터 유용한 정보를 추출하기 위해서는 데이터를 효율적으로 저장하고 관리하는 방법이 필요하다. 기존의 Apriori 알고리즘이 트랜잭션에 속한 아이템들의 트리 조인 과정이 오래 걸리고, 많은 수의 후보 집합 생성이라는 단점을 보완한 FP-growth 방식은 후보생성과정을 수행하지 않고, 트리의 prefix를 공유하고 있는 아이템을 찾아서 트리의 경로를 찾아간다. 그리고 같은 아이템은 포인터를 사용하여 가리키게 하는 방식으로 수행되므로 기존의 방식보다 데이터 마이닝의 성능을 향상시킨다. prefix tree 구조를 확장한 형태인 FP-tree를 이용하므로 트랜잭션 데이터베이스를 압축된 형태로 표현한다. 이 논문에서는 슬라이딩 윈도우 기반의 스트리밍 XML 데이터에서 빈발 구조를 추출하기 위하여 FP-tree를 변형한 XFP_tree를 이용한다. XFP_tree는 [13]의 DSTree를 기반으로 XML의 구조를 고려한 트리구조로써 빈발한 XML의 구조를 빠르게 추출하는 특징이 있다.

기존 연구에서는 트리를 구성하기 위해 트랜잭션들에 대한 항목들을 일련의 순서를 구성하고 이를 기준으로 트리 구조를 구성한다. 그러나 XML 데이터는 구조적인 순서가 중요한 요소이기 때문에 구조 정보를 보존하면서 트리를 구성해야 한다.

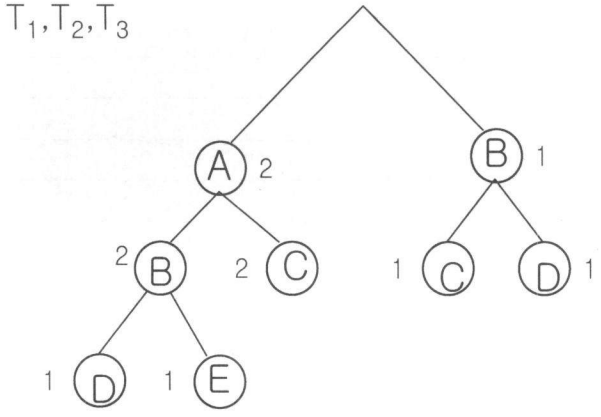
입력되는 XML 데이터에 의해 구성되는 트리 XFP_tree는 XML의 구조를 구성하는 엘리먼트들을 트리의 노드로 구성하여 항목들을 나타내고 각 항목에 대한 빈발도를 유지한다. 그리고 같은 경로를 갖는 노드들은 경로를 공유하는 구조로 나타낸다. (그림 3)은 XFP_tree의 구조 예를 보이기 위해 간단한 구조의 XML 스트림 데이터로 구성된 트리의 예이다.

(그림 4)와 (그림 5)는 순서대로 입력된 (그림 3)의 T1, T2, T3, T4로 구성된 XFP_tree의 예를 보여준다. XFP_tree에서 트리 T1과 T2는 루트 A와 B를 가진 서브트리이다. 그리고 XFP_tree를 구성할 때 각 서브트리들의 루트가 다르면 분리된 구조의 서브트리형태로 구성된다.

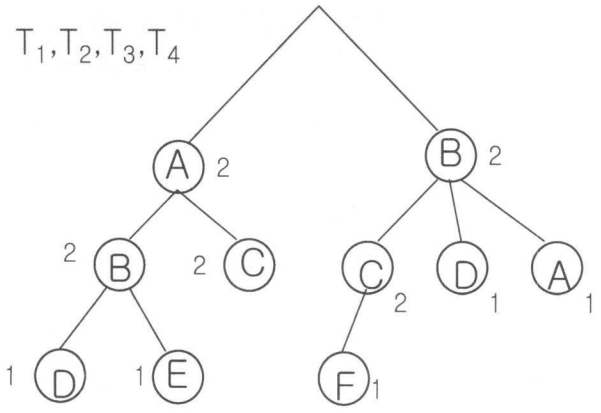
트리의 구성과 더불어 고려해야 할 것은 각 노드의 빈발도를 유지하는 것이다. XFP_tree는 트리에 대한 노드를 구



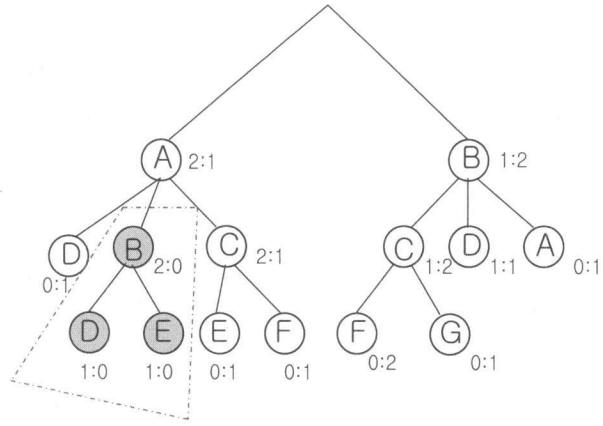
(그림 3) 스트리밍 XML 데이터의 트리 예



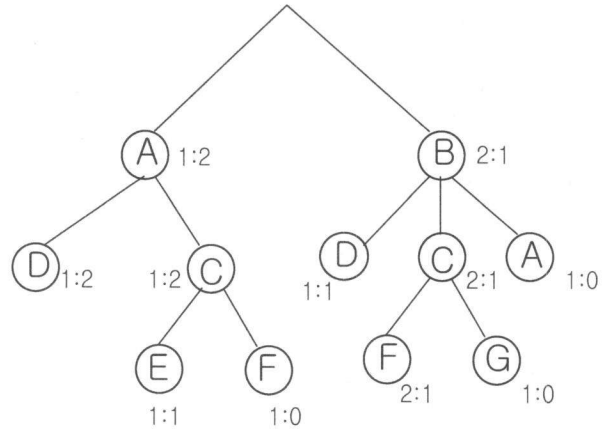
(그림 4) XFP_tree (T1, T2, T3 입력)



(그림 5) XFP_tree (T1, T2, T3, T4 입력)



(그림 6) batch_1과 batch_2에 대한 XFP_tree



(그림 7) batch_2와 batch_3에 대한 XFP_tree

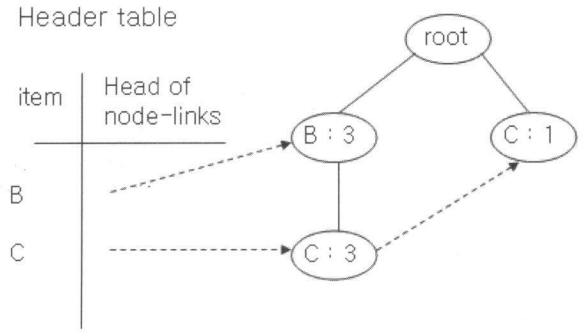
성할 때 루트부터 임의의 노드까지 같은 경로를 가진 노드들에 대한 빈발도를 유지한다. 이를 위해 입력되는 XML 트리를 하나의 트랜잭션으로 가정하고 트리를 구성하는 노드들을 트랜잭션의 항목으로 고려하여 각 항목에 대한 빈발도를 유지하며 윈도우를 이동시킬 때 삭제할 트랜잭션과 항목들의 빈발도를 관리하기 위해 각 노드 항목에 대해 루트부터 노드까지의 경로에 해당하는 패스경로를 하나의 구조항목으로 저장 및 관리한다. 예를 들어, 노드 D에 대한 경로 A-B-D의 경우에 ABD, 0, 1(구조항목, batch_i-1 빈발도, batch_i 빈발도) 의 세 가지 값을 하나의 엔트리로 구성한다. 윈도우의 이동으로 오래된 batch를 삭제하고 삭제되는 batch에 포함되어 있는 항목의 변화는 XFP_tree 트리의 모든 노드를 순회하면서 노드항목에 대한 엔트리를 검색하여 빈발도를 갱신한다.

이 논문에서는 트랜잭션들로 이루어진 스트림이 고정된 크기로 일괄(batch)처리되어 입력된다고 가정한다. 즉, 하나의 batch에는 일정한 수의 트랜잭션으로 구성되어 있다. FP_tree의 빈발도 유지에 대한 예를 보이기 위해 각 batch에 포함되어 있는 트랜잭션을 batch_1(t1, t2, t3), batch_2(t4, t5, t6), batch_3(t7, t8, t9)라 하고, 윈도우 사이즈를 2 batch라 가정한다. (그림 3)의 트리를 적용하면 스트림에서 처음에 입력되는 2개의 batch(window size=2) 즉, batch_1과 batch_2에 대한 XFP_tree는 (그림 6)과 같다.

입력된 batch_1과 batch_2에 포함되어 있는 트랜잭션의 항목들에 대한 빈발도는 노드의 루트부터 해당 노드까지의 경로에 해당하는 구조항목을 주된 키로 하여 해쉬 테이블을 이용하여 유지한다. 그리고 각 batch에 포함되어 있는 각 트랜잭션에 따른 항목의 상세한 정보는 유지하지 않고 각 batch에 포함된 구조항목의 전체적인 빈도수를 유지한다. (그림 6)에서 A 노드의 빈발도를 나타내는 2:1는 batch_1에 대한 빈발도 2와 batch_2의 빈발도 1를 나타낸 것이다.

그리고 batch_3이 입력되면 윈도우 사이즈가 2이므로 입력된 순서에 의해 가장 입력된 순서가 빠른 batch_1는 제거된다. 그러므로 batch_1에 해당하는 트랜잭션 T1, T2, T3에 포함되어 있는 항목들의 빈발도는 제거하고 batch_2에 대한 항목들의 빈발도가 이동되고 새롭게 입력된 batch_3에 포함되어 있는 항목들의 빈발도가 저장된다. 이것에 대한 결과를 (그림 7)에서 보여준다. 여기서 루트노드 A의 자손노드인 B노드를 루트로 하는 서브트리(그림 6)의 회색 노드로 표현된 서브트리)는 batch_2에서 빈도수가 0이었고 batch_3에서도 빈도수가 0이었기 때문에 (그림 6)의 XFP_tree로부터 완전히 제거된다.

(그림 7)에서와 같이 지속적으로 입력되는 트랜잭션들을 포함하는 batch에 대해 XFP_tree를 구성하는 과정은 다음과 같다.



(그림 8) 노드 F의 조건부 트리

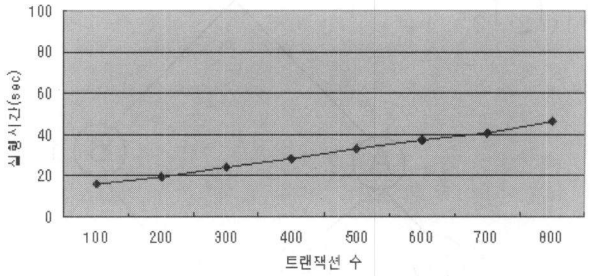
■ 입력된 batch의 수가 윈도우 사이즈보다 작거나 같은 경우
if batch_1
batch_1에 의한 XFP_tree 초기 생성
else
XFP_tree에 XML 트리 구조를 추가하여 갱신
새로운 구조 항목은 트리에 추가 생성

■ 입력된 batch의 수가 윈도우 사이즈보다 큰 경우
oldest batch를 XFP_tree로부터 삭제
XFP_tree에 XML 트리 구조를 추가하여 갱신
새로운 구조 항목은 트리에 추가 생성

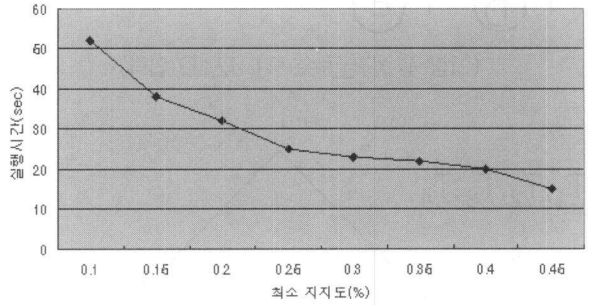
스트림 데이터가 입력된 임의의 시간 t에 대한 XFP_tree 상태를 (그림 7)이라 할 때, 최소 빈발 지지도 min_sup =3라 하면, 이를 만족하는 항목 및 경로는 {A:4, B:3, C:6, D:5, F:4, AC:3, AD:3, BC:3, BF:3, CF:4, BCF:3}이다. 예를 들어, (그림 7)에서 경로 ACE는 batch_2에서 빈발도 1, batch_3에서 빈발도 1를 나타내므로 batch_2와 batch_3를 포함하는 시간 t에서의 빈발도 합은 2이다.

XFP_tree에서의 빈발도는 FP-tree와 같은 방법으로 주어진 지지도를 만족하는 트리의 노드들에 대한 부분 데이터(projected database)를 생성하여 빈발도를 얻을 수 있다. 예를 들어, 노드 F에 대한 조건부 트리는 루트 노드 A로부터의 패스 A-C-F(1 : 0)에서 (A 1 : 0, C 1 : 0)와 루트 노드 B로부터의 패스 B-C-F(2 : 1)에서(B 2 : 1, C 2 : 1)를 기초로 (그림 8)과 같이 구축된다(노드 F에 대한 패스 ACF의 빈발도는 1이므로 min_sup만족하지 않아 트리구성에서 제외되지만 노드 C는 에지 CF:4(루트노드 A의 자손 에지 CF(1 : 0)과 루트노드 B의 자손 에지 CF(2 : 1)를 고려하여 조건부 트리를 구성한다).

XFP_tree는 현재 윈도우에 포함되는 트랜잭션들의 트리 구조에 대한 빈발도를 중심으로 주어진 지지도와 관계없이 구축된다. 그리고 주어진 지지도를 만족하는 경로 및 에지를 마이닝 할 때는 FP-growth와 같은 방법으로 XFP_tree 트리의 노드들에 대한 조건부 트리를 구성하며, 조건부 트리를 구성할 때는 최소 지지도를 만족하지 않는 노드는 고려하지 않는다. 그리고 이들 조건부 트리로부터 에지 중심의 빈발한 경로를 발견한다. 즉, (그림 8)로부터 노드 F에 대한 경로 및 에지의 빈발도는 BCF : 3, CF : 4임을 알 수 있다.



(그림 9) 트랜잭션 수의 변화에 따른 수행시간



(그림 10) 최소 지지도의 변화에 따른 수행시간

이와 같이 윈도우 이동에 따른 XFP_tree를 구성하고, 트리를 구성하는 노드를 중심으로 조건부 부분 트리를 생성하여 노드간의 에지 및 노드의 경로에 대한 빈발도를 유지하여 사용자의 마이닝 결과에 대한 요구에 빠른 응답을 줄 수 있다.

5. 실험결과

이 절에서는 스트리밍 XML 데이터의 마이닝 방법에 대한 효율성 및 성능을 측정하기 위하여 http://dblp.uni-trier.de/xml 사이트의 데이터 및 XML 데이터 뱅크[18]의 DTD를 이용하여 IBM's AlpaWords XML generator에 의해 실험 데이터를 생성하였다. 생성된 1000개의 XML 데이터를 스트림 데이터가 입력되는 형태로 순차적으로 입력하여 실험하였다. 생성된 XML 데이터의 평균적인 트리 깊이는 5.3이다. 실험에서 윈도우 사이즈 $\omega = 4$, 그리고 batch에 포함되는 총 트랜잭션의 수를 80개 이상으로 하여 트랜잭션의 수에 따른 수행속도를 실험하였다. (그림 9)는 현재 윈도우에 포함되어 있는 트랜잭션 수의 변화에 따른 실행 시간을, (그림 10)은 최소 지지도 min_sup의 변화에 따른 실행 시간을 보여준다.

(그림 9)에서는 트랜잭션 수의 변화에 따라 처리 시간이 선형적으로 증가하는 것을 보여준다. 그리고 XML 트리의 지속적인 입력으로 인해 노드의 수가 증가하고 이에 대한 처리 시간도 급격한 변화 없이 점진적으로 증가하고 있음을 보인다. 이 실험으로 알 수 있는 것은 윈도우 초기 상태 즉, 트리가 입력되는 초기에는 트리를 구성하고, 노드를 구성하는 항목들에 대한 구성으로 인해 처리시간이 조금 지연되지만 트리의 초기 구성과 항목 관리를 위한 데이터 구성이 초기에 구축되면 지속적으로 입력되는 트리에 대한 항목 관리 및 빈발 구조를 발견하는 데 소요되는 시간은 점진적으로

수행되는 것을 알 수 있다. (그림 10)에서는 지지도가 커질수록 수행시간이 급격히 줄어드는 것을 볼 수 있다.

두 번째 실험에서는 이 논문에서 제안한 XFP_tree로부터 추출되는 빈발구조의 정확성을 측정하였다. XFP_tree는 윈도우의 이동에 따른 가장 최근의 트랜잭션을 포함하는 batch의 XML 트리집합을 나타낸다. XFP_tree 로부터 마어닝하여 얻은 결과의 빈발 구조와 각각의 서브 트리 즉, 입력된 개별 트리로부터 직접 추출한 빈발 구조의 내용을 비교하였다. XFP_tree로부터 추출된 (department, deptname, gradstudent, name, phone, email), (movie, title, year, Directed-by, Genres, cast, Actor) 등과 같은 경로의 구조들은 개별 트리에서 FP-tree를 적용하여 추출된 빈발 구조와 비교하였을 때 100% 같은 추출 결과를 보였다. 이것은 개별적인 트리로부터 추출되는 빈발 구조와 스트림으로 입력되는 트리의 집합을 나타내는 XFP_tree로부터 빈발구조를 발견하는 것이 일치하는 것을 알 수 있다.

한편, 이 논문에서 제안하는 XFP_tree 구성에서 고려해야 할 것은 XML 트리가 유사하지 않은 구조의 XML 데이터가 계속적으로 입력될 경우에는 XFP_tree의 사이즈가 상당히 커질 수 있다는 단점이 있다. 그러므로 이러한 점들을 고려하는 방법과 트리의 사이즈에 따른 메모리 요구량, 윈도우 사이즈에 따른 성능평가 등에 대한 연구가 지속적으로 필요하다.

6. 결 론

XML은 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있기 때문에 현재 유비쿼터스 환경을 위한 기초 데이터로 사용되고 있는 것이 일반적이다. 그리고 이러한 XML 데이터의 입력은 연속적인 형태를 지닌다. 이 논문에서는 스트리밍 XML 데이터로부터 빈발구조를 추출하는 마이닝 방법을 제안한다. 이를 위해 XML 데이터를 트리로 표현하였고 개별적인 XML 트리들을 하나의 트리 집합으로 표현하는 XFP_tree를 제안하였다. 그리고 XFP_tree로부터 빈발구조의 에지 및 경로를 추출하는 마이닝 기법을 제안하였다. 스트리밍 XML 데이터의 지속적인 입력으로부터 최신의 XML 데이터에 대한 마이닝 결과를 추출하기 위해 슬라이딩 윈도우 기법을 기반으로 하였으며 XFP_tree의 각 노드는 현재 윈도우에 포함되어 있는 항목들의 빈발리스트를 유지하므로 써 효율적인 마이닝 결과를 얻을 수 있었다. 그리고 실험을 통해 제안하는 방법에 대한 성능 및 마이닝 결과에 대한 정확도를 알아보았다. 향후 연구에서 기존 연구 방법과의 실험 비교를 통해 XFP_tree의 사이즈 변화에 대한 처리 속도 및 성능 효과를 분석할 것이다.

참 고 문 헌

[1] 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

[2] V. Ganti, J. Gehrke, R. Ramakrishnan, "DEMON: Mining and Monitoring Evolving Data," TKDE 1391), pp.50-63, 2001

[3] R. Nayak, R. Witt, A. Tonev, "Data Mining and XML Documents," International Conference on Internet Computing, 2002.

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

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

[6] S. Babu, J. Widom, "Continuous Queries over Data Stream," SIG MOD Record 30(3), pp.109-120, 2001

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

[8] 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

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

[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

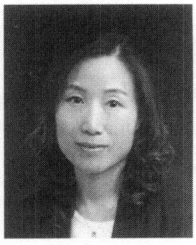
[14] J. Li. D. Maier, "Semantics and Evaluation Techniques for Window Aggregates in Data Streams," In Proc. of ACM SIGMOD International Conference on the Management of Data, 2005

[15] 장중혁, 이원석, "데이터 스트림에서 개방 데이터 마이닝 기반의 빈발 항목 탐색," 정보처리학회논문지D, 제10-D권 제3호, 2003.

[16] 김현규, 김철기, 김명호, "비순서화된 스트림 처리를 위한 슬라이딩 윈도우 기법," 정보과학회, 제33권 제 6호, 2006.

[17] 김영현, 강현철, "XML 스트림 데이터에 대한 적응력 있는 질의 처리 시스템," 정보과학회, 제33권 제 3호, 2006.

[18] 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 대학원 전자계산학과 (이학석사)

2005년 충북대학교 대학원 전자계산학과 (이학박사)

2001년~2005년 정우씨스템(주) 연구소장

2006년~현 재 남서울대학교 전임강사

관심분야 : XML 및 웹 데이터베이스, 스트림 데이터 관리, 데이터 마이닝, 유비쿼터스 컴퓨팅, 시공간 데이터베이스