TID 리스트 테이블을 이용한 연관 규칙 탐사
(Association Rule Discovery using TID List Table)

채 덕 진 † 황 부 현 ††
(Duckjin Chai) (Buhyun Hwang)

요 약 본 논문에서는 데이타베이스를 단 한번 스캔하여 빈발 항목집합들을 생성할 수 있는 효율적인 알고리즘을 제안한다. 빈발 항목집합은 어떤 트랜잭션이 접근하는 항목 집합의 부분집합이다. 각 항목에 대하여 그 항목을 접근하는 트랜잭션들에 관한 정보를 가지고 있다면, 동일한 트랜잭션 식별자를 갖는 항목들만을 추출함으로써 빈발 항목집합들을 생성할 수 있다. 본 논문에서 제안하는 방법은 한 번의 데이타베이스 스캔으로 각 항목마다 접근하는 트랜잭션 식별자를 저장할 수 있는 자료 구조를 생성하며, 동시에 해쉬 기법을 이용하여 2-빈발 항목집합들을 생성한다. 3-빈발 항목집합부터는 이 자료 구조와 각 항목에 대한 트랜잭션 식별자를 비교함으로써 간단히 빈발 항목집합들을 찾아낼 수 있다. 제안하는 알고리즘은 한 번의 데이타베이스 스캔만으로 빈발 항목집합들을 효율적으로 생성할 수 있다.

키워드 : 데이타마이닝, 연관 규칙, 빈발 항목집합

Abstract In this paper, we propose an efficient algorithm which generates frequent itemsets by only one database scanning. A frequent itemset is subset of an itemset which is accessed by a transaction. For each item, if informations about transactions accessing the item are exist, it is possible to generate frequent itemsets only by the extraction of items having an identical transaction ID. Proposed method in this paper generates the data structure which stores transaction ID for each item by only one database scanning and generates 2-frequent itemsets by using the hash technique at the same time. k(k≥3)-frequent itemsets are simply found by comparing previously generated data structure and transaction ID. Proposed algorithm can efficiently generate frequent itemsets by only one database scanning.

Key words : Data mining, Association rules, Frequent itemsets

1. 서 론

연관 규칙 탐사는 데이타베이스를 이루는 트랜잭션들에서 최소지지도(minimum support) 보다 많이 발생하는 빈발항목집합(Large itemset or Frequent itemset)들을 발견하는 문제로 정의할 수 있다. 지금까지 빈발항목집합들을 찾기 위한 다양한 연관 규칙 탐사 알고리즘들이 제안되었다[1-8]. 기존에 제안된 대부분의 알고리즘들은 Apriori[1] 알고리즘과 같은 휴리스틱(heuristic) 방법을 사용하였다[1-4,7,9-12]. 즉, 빈발 항목집합의 부분집합들은 또한 빈발 하다는 특성을 이용하고 있다. 이러한 특성을 이용하여 기존의 연관 규칙 탐사는 항목의 수가 k(k≥1)인 k-후보 항목집합들이 존재한다고 할 때, 후보 항목집합들이 사용자가 정의한 최소지지도를 만족하면 k-빈발 항목집합이 된다. 생성된 k-빈발 항목집합들은 다시 (k+1)-빈발 항목집합들을 생성하기 위한 (k+1)-후보 항목집합들을 생성할 때 사용되고, 더 이상의 후보 항목집합들이 생성되지 않을 때까지 이 과정은 반복 수행된다. 기존에 연구된 알고리즘들은 대부분 이러한 휴리스틱 방법을 사용하여 적은 수의 후보 항목집합들을 생성하고 데이타베이스의 크기를 줄이는 데 연구가 집중되었다[6]. 그러나 Apriori와 같은 휴리스틱 알고리즘들은 빈번한 데이타베이스 스캔을 하여야 함으로 대단히 많은 컴퓨팅 비용을 요구한다.

[6]에서는 후보 항목집합들을 생성하지 않고 두 번의 데이타베이스 스캔으로 빈발 항목집합들을 생성할 수 있는 빈발 패턴 트리(FP-tree : Frequent Pattern tree)라고 하는 새로운 트리 구조를 제안하였다. FP-tree 알고리즘은 첫 번째 데이타베이스 스캔에서 1-빈발 항목

· This work was supported by Institute of Information Assessment (ITRC)
† 비 회 원 : 전남대학교 전산학과
djchai@sunny.chonnam.ac.kr
†† 종신회원 : 전남대학교 전산학과 교수
bhhwang@chonnam.ac.kr
논문접수 : 2004년 7월 20일
심사완료 : 2005년 1월 22일

들을 생성하고 생성된 1-빈발 항목들을 지지도에 따라 정렬시킨다. 정렬 후 다시 각 트랜잭션들을 스캔하는데 이때, 데이타베이스에 있는 각각의 트랜잭션에 대해서 1-빈발 항목에 속하지 않는 항목들을 전지한다. 트랜잭션에서 전지되고 남은 항목들은 1-빈발 항목들을 정렬시킨 것과 같은 방법으로 정렬된다. 정렬된 트랜잭션은 FP-tree를 생성하는데 사용된다. 모든 트랜잭션에 대해서 이러한 과정을 수행하면 알고리즘은 종료된다.

FP-tree 알고리즘은 후보 항목집합들을 생성하지 않고 빈발 항목집합들을 생성할 수 있는 효율적인 알고리즘이다. 그러나 FP-tree를 생성하고 트리의 모든 노드를 검사하여 빈발 항목집합들을 생성하는 기본적인 연산 비용 외에 모든 트랜잭션에 대해서 빈발하지 않는 항목들을 전지하고 나머지 항목들을 정렬하는데 소요되는 별도의 연산 비용이 필요하다. 또한, 트랜잭션들을 이루는 항목들이 서로 다르면 새로운 노드를 생성하기 때문에 공간적 오버헤드도 커질 수 있다.

본 논문에서는 각 1-빈발 항목 마다 이 항목을 접근하는 트랜잭션들의 ID를 저장함으로써 한 번의 데이타베이스 스캔으로 최종 k-빈발 항목집합들을 생성할 수 있는 FTL(Frequent itemsets extraction using TID List) 알고리즘을 제안한다.

빈발 항목집합들은 다음과 같은 특성을 가진다[2].

1. 어떤 k-항목집합이 최소 지지도를 만족하면 k-항목집합의 부분집합 또한 최소 지지도를 만족한다.
2. 빈발 항목집합은 어떤 트랜잭션에 포함된다.

첫 번째 성질은 어떤 빈발 항목집합의 부분집합들 또한 최소지지도를 만족한다는 것이고, 두 번째 성질은 임의의 한 빈발 항목집합은 어느 트랜잭션이 접근하는 항목집합의 부분집합이라는 것이다. 다시 말하면 빈발 항목집합은 어느 트랜잭션에 반드시 존재한다는 것이다.

본 논문에서 제안하는 방법은 데이타베이스를 한번 스캔하여 1-빈발 항목들을 구하고, 각 1-빈발 항목마다 이 항목을 접근하는 트랜잭션들의 ID를 구한다. 이와 동시에 1-빈발 항목들의 식별자를 해쉬 함수의 매개변수로 사용하여 2-빈발 항목집합들의 지지도를 계산한다. 이것은 상대적으로 많은 컴퓨팅 비용을 수반하는 2-빈발 항목집합들을 한 번의 스캔으로 미리 계산하여 비용을 줄이기 위함이다. k-빈발 항목집합(k≥3)은 각 항목에 대한 트랜잭션들의 리스트를 가지고 있는 테이블에 있는 정보를 사용하여 생성한다. 그러므로 제안하는 방법은 한 번의 데이타베이스 스캔만을 한다.

연관 규칙 탐사 알고리즘들의 대부분의 컴퓨팅 비용은 대용량의 데이타베이스를 스캔하는데 있다. 그러므로 데이타베이스 스캔횟수를 줄이면 줄일수록 알고리즘의 성능은 크게 향상된다. 따라서 본 논문에서 제안하는 FTL 알고리즘은 한 번의 데이타베이스 스캔으로 빈발 항목 집합을 계산할 수 있기 때문에 컴퓨팅 비용을 상당히 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로서 연관규칙 탐사에 대한 개괄적인 소개와 대표적인 연관규칙 탐사 알고리즘인 Apriori 알고리즘과 FP-tree 알고리즘[6]에 대해서 살펴보고, 3장에서는 제안하는 알고리즘을 기술한다. 그리고 4장에서는 제안하는 알고리즘과 FP-tree 알고리즘의 성능을 비교 분석하고, 5장에서는 본 논문에 대한 결론 및 향후 연구에 대하여 논의한다.

2. 관련 연구

2.1 연관규칙 탐사[13]

연관 규칙이란 특정 사건의 발생이 다른 사건의 발생을 암시하는 경향을 표현하는 규칙으로 다음과 같이 정의할 수 있다. $I=\{a_1, a_2, ..., a_n\}$를 데이타 항목(item)들의 집합이라고 하고, 트랜잭션들로 구성된 데이타베이스를 $<T_1, T_2, ..., T_m>$라고 하자. T_i를 트랜잭션 T_i가 접근하는 데이타 항목들의 집합을 표현한다고 하면 $T_i \subseteq I$이다. 이때, 트랜잭션은 한 명의 고객이 구입한 품목들의 집합이라고 정의할 수 있다.

$X(X \subseteq T_i)$와 $Y(Y \subseteq T_i)$의 항목 집합에 대한 연관 규칙은 $X \rightarrow Y$로 표현되며, X를 규칙의 조건부(Antecedent), Y를 결과부(Consequent)라고 한다. 이때 $X \cap Y = \emptyset$이다[13]. $X \rightarrow Y$ 형태의 연관 규칙이 갖는 의미는 X 항목 집합이 나타날 때는 Y 항목 집합도 동반하여 나타나는 경향이 있다는 뜻이다.

연관 규칙에는 지지도(Support)와 신뢰도(Confidence)라는 두 가지 중요한 척도가 있다. 예를 들어, "빵을 구매하는 고객의 40%는 우유도 구매하며, 전체 트랜잭션의 2%는 빵과 우유를 포함하고 있다." 여기서 40%는 이 규칙의 신뢰도라고 하고, 2%는 이 규칙의 지지도라고 한다. 지지도와 신뢰도는 정의 1과 2에서 각각 정의된다. 정의 1과 2에서 표기된 $\sigma(X \cup Y)$는 전체 트랜잭션들 중에서 X집합과 Y집합을 동시에 포함하는 트랜잭션들의 수를 나타내며, $\sigma(X)$는 전체 트랜잭션들 중에서 X집합을 포함하는 트랜잭션들의 수를 나타낸다. 그리고 $\sigma(\text{All transactions})$는 전체 트랜잭션의 수를 나타낸다.

정의 1. 지지도

연관 규칙을 반영하는 트랜잭션들이 전체 데이타베이스에서 얼마만큼의 비율을 차지하고 있는 지를 나타내는 척도인데, 다음과 같이 표현된다.

$$Support = \frac{\sigma(X \cup Y)}{\sigma(All\ transactions)}$$

정의 2. 신뢰도

조건부의 모든 사건항목을 포함하고 있는 트랜잭션의

개수 중 결과부까지 함께 포함하고 있는 트랜잭션의 비율로 다음과 같이 정의된다.

$$Confidence = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

연관규칙 탐사는 상품 혹은 서비스 간의 관계를 살펴보고 이로부터 유용한 규칙을 찾아내고자 할 때 이용된다. 상품이나 서비스의 거래 기록데이타로부터 상품간의 연관성 정도를 측정하여 연관성이 많은 상품들을 그룹화하고 동시에 구매될 가능성이 큰 상품들을 찾아냄으로써 시장 바구니 분석에서 다루는 문제들에 적용될 수 있다.

2.2 빈발 항목집합의 특성

빈발 항목집합을 효과적으로 찾는 과정에서 얻어진 특성들로 대부분의 연관 규칙 탐사 알고리즘에서 다음과 같은 특성들을 사용한다[14]. 항목집합 A는 항목집합 B의 부분집합(A⊂B)이라고 하고 특성들을 표현한다.

- 특성 1(부분집합의 지지도) : B를 지지하는 데이타베이스의 모든 트랜잭션들이 필연적으로 A 또한 지지하므로 sup(A)(A의 지지도)≥sup(B)이다.
- 특성 2(빈발하지 않은 집합들을 포함하는 집합들은 빈발하지 않다) : 만일 항목집합 A가 데이타베이스에서 최소 지지도 S_{min}에 미치지 못한다면, 즉 sup(A) < S_{min}, 특성 1에 의하여 sup(B)≤sup(A) <S_{min}이기 때문에 A를 포함하는 집합 B는 빈발하지 않다.
- 특성 3(빈발 항목집합들의 부분집합들은 빈발하다) : 항목집합 B가 데이타베이스에서 빈발 하다면, 즉 sup(B)≥Smin, 특성 1에 의하여 sup(A)≥sup(B)≥Smin이므로 B의 모든 부분집합 A는 데이타베이스에서 또한 빈발하다. 특히, 만약 빈발항목집합 B의 모든 부분집합들도 빈발하다. 그러나 그 역은 성립하지 않는다.

2.3 FP-tree 생성 알고리즘

연관 규칙 탐사의 전체 성능은 첫 번째 단계인 빈발 항목집합을 찾는 과정에서 결정된다. 빈발 항목집합을 찾고 나면 두 번째 단계인 연관 규칙탐사는 쉽게 이루어진다. 이 절에서는 연관 규칙 탐사 알고리즘들 중에서 두 번의 데이타베이스 스캔으로 후보 항목집합들을 생성하지 않고 빈발 항목집합들을 추출할 수 있는 FP-tree 구조를 이용하는 알고리즘에 대해 분석한다.

FP-tree 알고리즘[6]은 빈발 패턴들에 대한 정보를 가지고 있는 FP-tree를 생성하고 최소지지도를 만족하는 빈발 항목집합들을 생성한다. 또한, 데이타베이스의 반복되는 스캔을 피하고 후보 항목집합들의 생성을 회피함으로써 빈발 항목집합들을 구하는 비용을 줄였다.

빈발 패턴 트리는 "null"로 표기된 하나의 루트 노드, item prefix subtrees, 그리고 헤더 테이블로 구성된다. item prefix subtree에서 각 노드는 표 1과 같이 세 부분으로 구성된다.

표 1 Item prefix subtree의 노드 구조

Factor	comment
item-name	빈발 항목의 이름
count	빈발 항목의 지지도
node-link	같은 이름을 가진 빈발 항목에 대한 포인터

헤더 테이블(header table)은 항목이름(item-name)과 그 항목의 헤드로 가는 포인터(head of node-link)로 구성된다. item-name은 1-빈발 항목들이고, head of node-link는 1-빈발 항목이 트리에서 처음 나타나는 노드를 가리킨다.

FP-tree 알고리즘은 먼저, 데이타베이스를 스캔하여 1-빈발 항목들과 지지도를 구한다. 그리고 1-빈발 항목들을 지지도에 따라 정렬하여 목록을 작성한다. 다음은 루트노드인 "null" 노드를 생성하고 데이타베이스의 각 트랜잭션을 스캔한다. FP-tree 알고리즘은 각 트랜잭션에 대해서 1-빈발 항목들의 목록에 속하지 않는 항목들을 전지하고 남은 항목들(빈발 항목)은 다시 1-빈발 항목들의 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. 트리 생성은 루트노드인 "null" 노드로부터 시작하여 조상에서 자손의 순으로 각 노드들을 비교하면서 생성된다. 즉, 트랜잭션의 항목들이 기존에 존재하는 트리의 노드에 있는 항목들과 같으면 카운트를 하나 증가시키고 다르면 새로운 노드를 생성하며 카운트를 1로 초기화시킨다. 이러한 과정은 모든 트랜잭션에 대해 반복적으로 수행된다.

표 2와 그림 1은 알고리즘의 수행과정을 보여주는 하나의 예이다. 표 2는 초기 데이타베이스와 1-빈발 항목들에 속하지 않은 항목들을 전지한 후, 나머지 항목들을 지지도에 따라 정렬한 결과를 나타낸다. 그리고 그림 1은 트랜잭션들의 데이타베이스를 가지고 실제 알고리즘을 수행한 결과를 나타낸다. 그림 1에서 보이는 점선은

표 2 수행보기로서 트랜잭션들의 데이타베이스

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

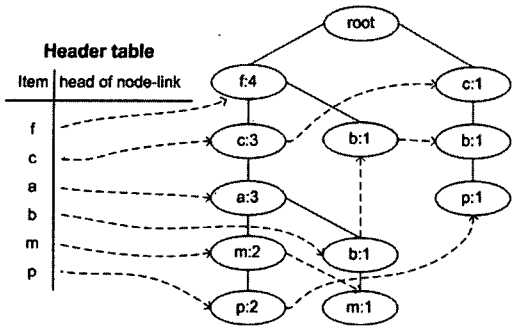

그림 1 수행보기에 대한 The FP-tree

한 항목이 여러 노드에 분산되어 있는데 이러한 노드들의 링크를 나타내고, 실선은 한 트랜잭션을 이루는 항목들 간의 링크이다. 루트에서 하나의 리프노드까지의 실선 경로 상에 있는 항목들의 집합이 어느 트랜잭션에 속하는 항목들이다.

생성된 FP-tree는 연관 규칙들을 생성하는데 이용된다. 생성된 FP-tree의 헤더 테이블에서 각 항목의 head of node-link에서부터 bottom-up 방식으로 링크를 따라 연관 규칙들을 생성하게 된다. 생성된 FP-tree에서 각각의 경로는 하나의 트랜잭션을 이루는 항목들의 관계를 나타내고 있기 때문에 각 노드가 가지고 있는 지지도를 검색하여 쉽게 도출할 수 있다. 즉, 표 3에서 보이는 것처럼 각 항목의 링크 필드는 항목 필드의 항목으로부터 시작하여 루트까지 링크된 모든 경로들을 나타낸다. 그리고 연관 규칙 필드는 그 링크들로부터 최소 지지도를 만족하는 규칙들을 추출한 것이다.

표 3 각 항목에서 생성될 수 있는 규칙(item:support)

항목	각 항목의 링크(bottom-up)	연관 규칙
p	{(f:2, c:2, a:2, m:2), (c:1, b:1)}	{(c:3, p:3)}
m	{(f:4, c:3, a:3, m:2), (f:4, c:3, a:3, b:1, m:1)}	{(f:3, c:3, a:3, m:3)}
b	{(f:4, c:3, a:3, b:1), (f:4, b:1), (c:1, b:1)}	∅
a	{(f:3, c:3)}	{(f:3, c:3, a:3)}
c	{(f:3)}	{(f:3, c:3)}
f	∅	∅

FP-tree를 이용한 연관규칙 탐사 알고리즘(fp-growth)은 두 번의 데이타베이스 스캔을 하지만 후보 항목집합들을 생성하지 않고 빈발 항목집합을 생성할 수 있는 효율적인 알고리즘이다. 그러나 FP-tree 생성 과정에서 크게 두 가지의 문제점이 있다.

첫째, 하나의 트랜잭션을 스캔할 때 세 가지의 연산을 수행한다는 것이다. 이것은 단순히 임의의 항목들을 삽입하고 삭제하는 비교적 가벼운 연산이 아닌 비교 연산을 수행하기 때문에 상당한 시간적 비용을 수반하게 된다. 빈발 항목집합을 생성하기 위한 작업을 다음과 같은 세 가지 연산으로 나눠볼 수 있다.

- 트랜잭션 스캔
- 빈발 하지 않는 항목 전지
- 전지되고 남은 빈발 항목들의 정렬

위에서 언급한 세 가지 연산은 FP-tree를 생성하기 전에 준비해야 할 별도의 연산과정이다. 둘째, 트리를 생성해나감에 따라 공간적 비용이 커질 수 있다는 것이다. 즉, 하나의 트랜잭션을 스캔하여 트리를 생성할 때, 루트의 자식노드부터 비교해 가면서 같은 노드 항목이면 지지도 카운트를 하나 증가시키면 되지만 노드 항목이 다르면 새로운 노드를 생성하게 된다. 이는 같은 노드 항목이 존재해도 조상이 틀리면 새로운 노드 항목으로 생성되어져야 하기 때문에 항목들의 분포에 따라 무수히 많은 노드들이 생성될 수 있기 때문이다. 또한, 각 노드들은 각각의 지지도를 저장할 공간을 가지고 있으므로 이런 경우에는 공간적 오버헤드를 가지게 된다.

본 논문에서는 이러한 문제점들을 해결하고자 FTL (Frequent itemsets extraction using TID List) 알고리즘을 제안한다. 제안하는 알고리즘은 한 번의 데이타베이스 스캔으로 빈발 항목집합들을 생성할 수 있기 때문에 빠른 시간에 원하는 빈발 항목집합들을 찾아낼 수 있다.

3. FTL 알고리즘

연관규칙 탐사에서 빈발 항목집합을 발견하는데 가장 효율적인 방법은 트랜잭션에 숨겨져 있는 빈발 항목집합들을 한 번의 스캔으로 찾아내는 것이라 할 수 있다. 그러나 대용량의 데이타베이스에서 빈발 항목집합들을 눈으로 들여다보듯이 찾아낼 수 없기 때문에 기존의 알고리즘들은 1-빈발 항목집합들로부터 순차적으로 빈발 항목집합들을 찾아야만 했다. 그러나 각 k-빈발 항목집합들을 찾을 때마다 대용량의 데이타베이스를 스캔하는 것은 대단히 많은 계산 비용 부담을 수반 하게 된다. 그러므로 기존의 알고리즘들은 데이타베이스 스캔 횟수를 줄이고자 하는 노력과 k-후보 항목집합들의 수를 줄이는 방법으로 연구가 되어왔다.

이 절에서는 한 번의 데이타베이스 스캔으로 빈발 항목집합들을 찾아낼 수 있는 방법을 제안한다. 빈발 항목집합을 찾는 문제는 찾고자하는 빈발 항목집합이 어떤 트랜잭션들에 존재하며 그 트랜잭션들의 수가 사용자가 정한 최소지지도를 만족하는지를 검사하는 문제이다. 그러므로 데이타베이스를 이루는 각 항목이 어떤 트랜잭션에 존재하는지에 대한 정보를 가지고 있는 자료구조

를 생성한다면 그 이후에는 데이타베이스를 스캔하지 않고 빈발 항목집합들을 찾아낼 수 있다.

본 논문에서는 Apriori 알고리즘에서 사용하던 후보 항목집합 생성 방법을 사용하여 후보 항목집합을 생성한다. 그리고 생성된 후보 항목집합을 본 논문에서 제안하는 TID List 테이블에 저장된 각 항목들의 트랜잭션 식별자들과 비교하여 최소 지지도를 만족하는지를 판별하게 된다. 본 논문에서는 3-빈발 항목집합부터 TID List 테이블을 사용하는데 1-빈발 항목들은 한 번의 데이타베이스를 스캔하여 TID List 테이블을 생성할 때 추출되고, 2-빈발 항목집합들은 데이타베이스를 스캔할 때 해쉬 기법을 사용하여 추출하게 된다. 그러므로 데이타베이스를 스캔할 때 생성되는 자료 구조는 TID List 테이블과 해쉬 테이블이 생성된다. 이때 1-빈발 항목들은 TID List 테이블에 존재하는 각 항목들의 크기가 최소지지도를 만족하면 1-빈발 항목으로 추출된다. 2-빈발 항목집합들은 해쉬 테이블의 각 버켓에 저장된 2-항목들의 카운트가 최소 지지도를 만족하는지 검사되고 만족하면 2-빈발 항목집합으로 추출된다.

3.1 TID List 테이블과 1-빈발 항목집합 생성

빈발 항목집합은 데이타베이스를 이루는 어느 트랜잭션에 존재한다. 즉, 하나의 k-빈발 항목집합이 어느 트랜잭션에 존재하므로 이 빈발 항목집합의 k 개의 항목들은 각각 동일한 트랜잭션 식별자를 가진다. 예를 들어 하나의 항목집합 {A, B, C}가 지지도 3으로 빈발하다고 하면 항목 A, B, C를 모두 포함하고 있는 트랜잭션이 세 개 있다는 것이다. 본 논문에서 제안하는 FTL 알고리즘은 이러한 특성을 이용하여 데이타베이스를 스캔할 때 각 항목에 대하여 이 항목을 포함하고 있는 트랜잭션의 식별자(Transaction ID)를 TID List 테이블에 저장한다. 표 4는 표 2에서 사용한 예제에 대하여 각 항목에 대한 트랜잭션 식별자들을 TID List 테이블에 저장한 예이다.

1-빈발 항목집합은 TID List 테이블을 조사하여 각 항목이 가지는 TID List 길이를 지지도와 비교함으로서 쉽게 계산할 수 있다. 표 4의 TID List 테이블을 조사함으로서 지지도 50% 이상을 만족하는 1-빈발항목집합을 구하면 표 5와 같다.

표 4 TID List 테이블의 보기

Item	TID List	Item	TID List
a	100, 200, 500	j	300
b	200, 300, 400	k	400
c	100, 200, 400, 500	l	200, 500
d	100	m	100, 200, 500
e	500	n	500
f	100, 200, 300, 500	o	200, 300
g	100	p	100, 400, 500
h	300	s	400
i	100		

표 5 1-빈발항목 집합들

Item	TID List	support
a	100, 200, 500	3
b	200, 300, 400	3
c	100, 200, 400, 500	4
f	100, 200, 300, 500	4
m	100, 200, 500	3
p	100, 400, 500	3

3.2 해쉬 기법을 이용한 2-빈발 항목집합 생성

연관 규칙은 하나의 트랜잭션에 있는 항목들 사이의 관계를 표현한다. 해쉬 기법을 사용하여 모든 트랜잭션들 각각이 가지고 있는 2-항목집합에 대한 버켓에 이 항목을 포함하고 있는 트랜잭션들의 수를 저장한다면 우리가 찾고자하는 2-빈발 항목집합은 쉽고 빠르게 찾을 수 있다. 그러나 데이타베이스를 이루는 트랜잭션들이 접근하는 항목집합에 대한 모든 부분집합들에 대한 버켓을 유지해야하는 공간상의 문제가 존재한다. 현재 하드웨어가 빠르게 발전하고 있다고 하지만 저장 공간이 지수적으로 증가하는 문제를 해결하기는 어렵다.

본 논문에서는 2-빈발 항목집합들을 추출하는데 해쉬 기법을 사용한다. 제안하는 알고리즘에서는 TID List 테이블을 생성 할 때 각 트랜잭션이 접근하는 항목집합에 대한 2-항목집합들 각각에 대하여 해쉬 함수를 적용하여 버켓을 구하고 그 버켓에 카운트를 하나 증가 시킨다. 모든 트랜잭션들을 처리하고 나면 버켓에는 2-항목집합들의 발생 빈도수가 저장된다. 표 6은 트랜잭션들을 스캔하여 해쉬 함수를 적용하여 구한 버켓과 그 버켓에 2-항목집합들의 발생 빈도수를 저장한 예를 보여주고 있다. 본 논문에서 찾고자하는 연관 규칙은 순차패턴이 아니므로 교환 법칙이 성립한다. 즉, {a, b}라는 항목집합과 {b, a}라는 항목집합은 동일한 것으로 간주한다. 그리고 편의상 알파벳 순서가 빠른 것을 먼저 표기한다. 또한, 버켓의 주소는 2-항목집합을 표현할 수 있는 2차원 배열을 사용한다. 예를 들어, {a,b}라는 항목집합이 발생하면 2차원 배열[a][b]에 빈도수를 누적한다. 해쉬 테이블이 생성되면 해쉬 테이블의 각 버켓의 발생 빈도를 최소 지지도 임계값과 비교하여 임계값 이상이면 2-빈발 항목집합으로 추출된다. 그러므로 표 6의 해쉬 테이블로부터 표 7과 같은 2-빈발 항목집합들을 계산할 수 있다. 여기서 최소지지도는 50%로 가정하였다.

모든 빈발 항목집합들을 해쉬 기법을 사용하여 추출한다면 기존의 알고리즘들보다 훨씬 적은 시간에 빈발

표 6 2-항목집합의 빈도수를 저장하는 해쉬 테이블의 예 해쉬함수 f(x,y) = array[x][y]

index-->	{a,f}	{a,c}	{a,d}	{a,g}	{a,i}	{a,m}	{a,p}	{a,b}	{a,l}	{a,o}	{a,e}	{a,n}
count-->	3	3	1	1	1	3	2	1	2	1	1	1
	{b,c}	{b,f}	{b,l}	{b,m}	{b,o}	{b,k}	{b,s}	{b,p}				
	2	1	1	1	1	1	1	1				
	{c,f}	{c,d}	{c,g}	{c,i}	{c,m}	{c,p}	{c,l}	{c,o}	{c,k}	{c,s}	{c,e}	{c,n}
	3	1	1	1	3	3	2	1	1	1	1	1
	{d,f}	{d,g}	{d,i}	{d,m}	{d,p}							
	1	1	1	1	1							
	{f,g}	{f,i}	{f,m}	{f,p}	{f,l}	{f,o}	{f,h}	{f,j}	{f,o}	{f,e}	{f,n}	
	1	1	3	2	2	1	1	1	1	1	1	
	{g,i}	{g,m}	{g,p}									
	1	1	1									
	{i,m}	{i,p}										
	1	1										
	{l,m}	{l,o}	{l,n}	{l,p}								
	2	1	1	1								
	{m,p}	{m,o}	{m,n}									
	2	1	1									
	{h,j}	{h,o}	{j,o}	{k,s}	{p,s}	{e,p}	{e,m}	{e,n}	{e,l}	{p,n}		
	1	1	1	1	1	1	1	1	1	1		

표 7 2-빈발항목집합

{a,f}	{a,c}	{a,m}
3	3	3
{c,f}	{c,m}	{c,p}
3	3	3
{f,m}		
3		

항목집합들을 추출할 수 있을 것이다. 왜냐하면 데이타베이스의 모든 트랜잭션 각각에 대하여 모든 k(k>1)-부분집합들을 버켓에 저장하고 단지 이들의 지지도만 최소지지도와 비교하여 빈발 항목집합들을 찾아내면 되기 때문에 빠른 시간 내에 모든 빈발 항목집합들을 찾을 수 있다. 그러나 이러한 방법으로 후보 항목집합들을 생성하는 것은 $\binom{n}{k}$연산을 수행하게 된다. n은 데이타베이스를 이루는 항목들의 개수이고 k는 후보 항목집합을 이루는 원소의 개수이다. 그러므로 k가 커지면 생성되는 버켓의 수는 지수적으로 증가하게 된다. 이는 너무 많은 공간을 요구하여 사용하기 어렵다. 그러나 부분적으로 이 방법을 적용하면 효율적으로 빈발항목 집합들을 구할 수 있다. 특히 대부분 2-빈발 항목집합이나 3-빈발 항목집합이 많기 때문에 이러한 방법을 적용하면 시스템 성능 개선에 많은 효과를 볼 수 있다.

3.3 TID List 테이블을 이용한 k(k)2)-빈발 항목집합 생성

후보 항목집합들은 Apriori 알고리즘에서 사용했던 Apriori-gen 알고리즘을 사용하여 생성한다. 기존의 알고리즘에서는 생성된 후보 항목집합들이 최소 지지도를 만족하는지 확인하기 위해 데이타베이스를 스캔하였다. 제안하는 알고리즘은 첫 번째 데이타베이스를 스캔할 때 생성한 TID List 테이블을 이용하여 빈발 항목집합을 추출할 수 있다. TID List 테이블은 데이타베이스를 이루는 항목들이 어떤 트랜잭션에서 발생하는 지에 대한 모든 정보들을 가지고 있다. 그러므로 후보 항목집합의 각 원소들을 TID List 테이블에서 찾아 그 항목들이 가지고 있는 트랜잭션 식별자들을 비교하면 후보 항목집합의 지지도가 계산된다.

표 8은 표 4에서 생성된 TID List 테이블과 표 7의 2-빈발 항목집합들을 가지고 k(k>2)-빈발 항목집합들을 생성하는 과정을 보여주는 예이다. 표 7에서 보는 것처럼 2-빈발 항목집합들은 {a,f}, {a,c}, {a,m}, {c,f}, {c,m}, {c,p}, {f,m}이다. 3-후보 항목집합들의 생성을 위해 Aprioro-gen 알고리즘을 적용하면 일부는 전지되고 {a, c, f}, {a, c, m}, {a, f, m}, {c, f, m}만 후보 항목집합으로 남는다. 빈발 항목집합들을 추출하기 위해 후보 항목집합의 지지도를 계산하는 것은 TID List 테이블에 있는 각 항목들의 TID를 비교함으로써 해결할 수 있다. 왜냐하면 빈발 항목집합은 어떤 트랜잭션에 존재하므로 TID List 테이블에서 후보 항목집합을 이루는 각 원소가 공통의 TID를 가지면 그 원소들은 하나의 트랜잭션에 존재하는 것이기 때문이다. 표 8에서 나타난 것처럼 {a, c, f}, {a, c, m}, {a, f, m}, {c, f, m}은 트랜잭션 100, 200 그리고 500에 모두 포함됨으로 이들은 최소 지지도 임계값 50%를 만족한다. 그러므로 이들은

표 8 TID List 테이블을 사용한 k(k>2)-빈발항목집합 생성

Candidate itemset	Item	TID List	Common TID
{a, c, f}	a	100, 200, 500	100, 200, 500
	c	100, 200, 400, 500	
	f	100, 200, 300, 500	
{a, c, m}	a	100, 200, 500	100, 200, 500
	c	100, 200, 400, 500	
	m	100, 200, 500	
{a, f, m}	a	100, 200, 500	100, 200, 500
	f	100, 200, 300, 500	
	m	100, 200, 500	
{c, f, m}	c	100, 200, 400, 500	100, 200, 500
	f	100, 200, 300, 500	
	m	100, 200, 500	
{a, c, f, m}	a	100, 200, 500	100, 200, 500
	c	100, 200, 400, 500	
	f	100, 200, 300, 500	
	m	100, 200, 500	

3-빈발 항목집합으로 추출된다. 다시 이들은 4-후보 항목집합을 생성하기 위해 사용되는데 이 예제에서는 {a, c, f, m}이 생성된다. 다시 TID List 테이블에 저장된 각 항목들의 TID를 비교하면 표 8에서 최소 지지도를 만족하므로 4-빈발 항목집합으로 추출되고 알고리즘은 종결된다.

```
Algorithm(FTL : TID List construction and Frequent Itemset extraction)
Input : A transaction database DB and a minimum support threshold ξ.
Output : TID List table. k-frequent itemsets
method : algorithm is performed in the following steps.
// TID List table construction and 2-frequent itemsets generation
Procedure TIDList&2-frequent(DB)
{
   for each transaction do
      for each item do
         generate TID List table
         generate bucket using hash function
   for all buckets do
      if count value of bucket > ξ
      then insert to 2-frequent itemsets
}
// k(k>2)-frequent itemsets generation using TID List table
Procedure k-frequent(TID List, k-frequent itemsets)
{
   generate (K+1)-candidate itemsets
   compare TID(Transaction ID) of each item
   if # of the common TID > ξ
   then insert to (k+1)-frequent itemsets
   if candidate itemsets can generate
   then call k-frequent(TID List, (k+1)-frequent itemsets)
}
```

그림 2 FTL 알고리즘

데이타베이스를 이루는 트랜잭션들을 스캔할 때, 첫 번째부터 순차적으로 스캔하기 때문에 TID List 테이블에 저장된 트랜잭션 식별자들은 순차적으로 정렬되어 있다. 이러한 구조에서 트랜잭션 식별자들을 비교하는 연산은 간단히 포인터를 둠으로써 매번 처음부터 비교해야하는 연산 비용을 줄일 수 있다. 예를 들면, 표 8의 예에서 3-항목집합 {a, c, f}는 각각 {100, 200, 500}, {100, 200, 400, 500} 그리고 {100, 200, 300, 500}의 TID List를 가지고 있다. 처음 {a} 항목이 가지고 있는 리스트 중에서 첫 번째인 100이라는 트랜잭션 식별자가 다른 항목들이 가지고 있는 식별자 리스트 중에 있는지 검사하게 된다. 이때 포인터들은 각각 첫 번째를 가리키게 된다. 모두 첫 번째에 같은 식별자를 가지고 있으므로 지지도는 1로 세팅된다. 다음 200의 식별자가 다른 리스트에도 있는지를 검색하는데 이때, 식별자들은 정렬되어 있으므로 지금 현재 포인터가 가리키고 있는 곳을 비교할 필요는 없다. 그러므로 포인터를 하나 증가시키게 된다. 즉, a, b, c 항목들이 가지고 있는 TID List 테이블에서 포인터들은 두 번째 식별자인 200을 가리키게 된다. 세 항목 모두 두 번째에 200이라는 같은 식별자들이 존재하므로 지지도는 하나 증가되어 2가 된다. 마지막으로 {a} 항목이 가지고 있는 500이라는 식별자를 다른 항목들이 가지고 있는지를 검색하게 되는데 이때 포인터들을 하나씩 증가시켜서 같은 식별자들이 있는지 검색하게 된다. 이처럼 포인터를 둠으로써 처음부터 비교해야하는 연산 비용을 크게 줄일 수 있다.

본 논문에서 제안하는 알고리즘은 한 번의 데이타베이스 스캔으로 데이타베이스를 이루는 모든 항목들이 어떤 트랜잭션에서 발생했는지를 알 수 있는 TID List 테이블을 생성한다. 그 이후의 k-빈발 항목집합들은 단순히 TID List 테이블만 가지고 추출될 수 있기 때문에 알고리즘 성능이 우수하다. FTL 알고리즘은 그림 2에 기술되어 있다.

4. 성능 평가

이 절에서는 시뮬레이션 실험을 통하여 기존의 알고리즘들 중에서 최근에 제안된 FP-tree 생성 알고리즘과 성능을 비교한다. 실험은 리눅스 운영체제, 1 GB 메인 메모리, 2GHz Pentium PC를 가지고 수행되었다. 실험 데이타는 오라클 9i 데이타베이스관리 시스템을 사용하여 오라클에 생성된 데이타베이스로부터 직접 입력을 받았다. 그리고 프로그램은 JAVA 버전 1.4.2를 사용하여 작성되었다.

실험에서 기록한 실행 시간은 오라클 데이타베이스로부터 데이타를 입력 받을 때부터 최종 알고리즘 수행이 끝날 때까지를 기록하였다. 그리고 실험에서 사용한 데이타는 기존의 알고리즘들에서 성능 평가를 위해 사용

하였던 데이타 생성기를 사용하여 생성하였다. 데이타생성기는 http://www.almaden.ibm.com/software/quest/Resources/index.shtml에서 무료로 다운로드 받을 수 있다.

실험은 두 개의 데이타 집합을 가지고 수행하였다. 하나는 전체 항목들의 개수가 10000개인 T20.I10.D10K이다. 이후에는 이 데이타 집합을 D1로 표시할 것이다. 다른 하나는 전체 항목들의 개수가 100000개인 T20.I20.D100K이다. 이 데이타 집합은 D_2로 표시할 것이다. 이때 T20의 의미는 트랜잭션을 이루는 항목들의 수가 평균 20개이고, I20은 잠재적으로 가능한 최대 빈발 항목집합의 크기가 20이라는 의미이다.

D_1과 D_2, 두 개의 데이타 집합에 대해서 실험한 결과는 그림 3과 그림 4에 나타나 있다. 각 그래프는 지지도 임계값에 따른 알고리즘의 실행 시간을 표시한 것이다.

그림 3은 데이타 집합 D_1을 사용한 것이고, 그림 4는

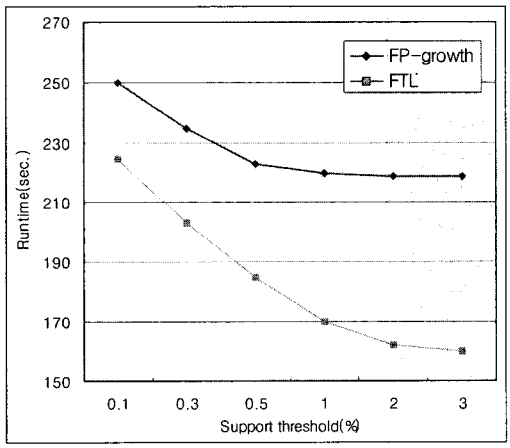

그림 3 Scalability with threshold(Data set : D1)

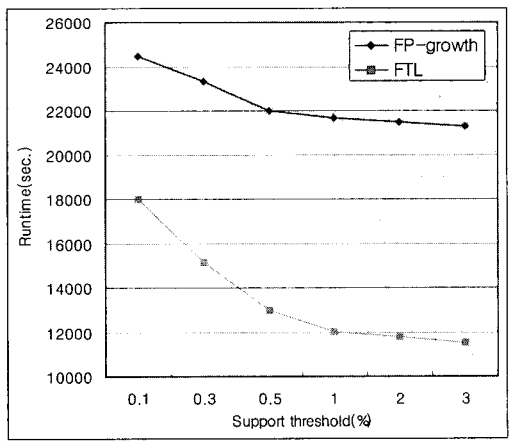

그림 4 Scalability with threshold(Data set : D_2)

데이타 집합 D_2를 사용한 것으로 지지도 임계값을 3%에서부터 0.1%까지 줄이면서 실험을 하였다. 실험 결과에서 보듯이 FP-tree 생성 알고리즘보다 본 논문에서 제안하는 FTL 알고리즘의 성능이 훨씬 더 우수하다는 것을 알 수 있다. 이러한 현상은 데이타베이스를 스캔하는 횟수를 한 번으로 줄인 것과 빈발 항목집합을 생성하는 연산의 비용이 FP-tree 생성 알고리즘 보다 더 적기 때문이라고 말할 수 있다. 또한, 지지도 임계값이 높을 때보다 낮을 때 두 알고리즘의 실행시간의 차가 줄어드는데, 이는 지지도가 낮을 때 빈발 항목집합의 수가 많이 발생하므로 다음 단계의 빈발 항목집합들을 생성하기 위한 연산의 수가 증가한 이유라고 볼 수 있다.

5. 결 론

임의의 k-빈발 항목집합은 어떤 트랜잭션이 접근하는 항목들의 집합이다. 빈발 항목집합을 구하기 위하여 데이타베이스를 이루는 모든 트랜잭션들을 검사 하여야 한다. 빈발 항목집합들을 생성하는 문제는 어떤 트랜잭션에 어떤 빈발 항목집합이 존재하는지를 찾는 문제로 정의할 수 있다. 이러한 문제를 해결하기 위해서 기존의 알고리즘들은 bottom-up 방식으로 항목의 개수가 늘어나면서 빈발 항목집합들을 생성하였다. 또한, 빈발 항목집합들을 생성하는 과정에서 데이타베이스를 스캔하는 것은 필수적 이였다. 기존의 대부분의 알고리즘들은 대용량의 데이타베이스를 스캔하는 횟수를 줄이는데 초점을 맞추고 있다. 스캔횟수를 줄인 대표적인 알고리즘이 FP-tree 생성 알고리즘인데, 이 알고리즘은 데이타베이스 스캔을 2회만 하지만 빈발 항목집합들을 생성하는데 필요한 연산이 많기 때문에 많은 컴퓨팅 비용을 요구한다.

본 논문에서는 대용량 데이타베이스를 스캔횟수를 줄이고, 빈발 항목집합을 효율적으로 생성하기 위한 FTL 알고리즘을 제안하였다. FTL 알고리즘은 데이타베이스를 스캔하면서 TID List 테이블 생성하고 해쉬를 사용한 2-항목집합들의 빈도수를 계산한다. TID List 테이블은 데이타베이스를 이루는 각 항목이 어떤 트랜잭션에 존재하는지에 대한 정보를 가지고 있다. TID List 테이블은 빈발 항목들에 대한 정보만을 가진 축소된 데이타베이스라고 할 수 있다. 왜냐하면 빈발 항목집합의 부분집합은 빈발하기 때문에 빈발 하지 않는 항목들은 전지될 수 있다. 그러므로 TID List 테이블은 전체 데이타베이스에서 빈발 항목집합과 관련된 정보만을 가지고 있다. TID List 테이블을 사용하여 빈발 항목들이 서로 같은 트랜잭션을 공유하는 횟수가 몇 번 인지 카운트만 계산하면 쉽게 k-빈발항목 집합을 구할 수 있기 때문에 불필요한 연산이 없이 빠르게 연관 규칙을 탐사할 수 있다. 시뮬레이션 실험을 통하여 제안한 FTL 알

고리즘이 FP-tree 알고리즘 보다 지지도가 커질수록 우수함을 보였다.

본 논문에서 제안한 FTL 알고리즘은 데이타베이스 스캔 횟수를 한 번으로 줄였다는데 큰 의미가 있다. 그러나 빈발 항목집합들을 생성하기 위한 후보 항목집합들의 효과적인 전지 기법이 뒤 따라야 할 것이다. 또는 데이타베이스 스캔 횟수를 한 번으로 하되 후보 항목집합 생성을 하지 않고 빈발 항목집합들을 생성할 수 있다면 훨씬 성능이 좋은 알고리즘이 될 것이다.

참 고 문 헌

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," In VLDB, pp. 487-499, 1994.

[2] J.S. Park, M.S. Chen and P.S. Yu, "An effective hash-based algorithm for mining association rules," In ACM SIGMOD, pp. 175-186, 1995.

[3] B. Lent, A. Swami and J. Widom, "Clustering association rules," In ICDE, pp. 220-231, 1997.

[4] R. Ng, L. V. S. Lakshmanan, J. Han and A. Pang, "Exploratory mining and pruning optimizations of constrained associations rules," In SIGMOD, pp. 13-24, 1998.

[5] B. Liu, W. Hsu and Y. Ma, "Mining association rules with multiple minimum supports," In ACM SIGKDD, pp. 337-341, 1999.

[6] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," In ACM SIGMOD, pp. 1-12, 2000.

[7] G. Grahne, L. Lakshmanan and X. Wang, "Efficient mining of constrained correlated sets," In ICDE, 2000.

[8] R. Agarwal, C. Aggarwal and V. V. V. Prasad, "A tree projection algorithm for generation of frequent itemsets," In J. Parallel and Distributed Computing, 2000.

[9] M. Klemettinen, h. Mannila, P. Ronkainen, h. Toivonen and A.I. Verkamo, "Finding interesting rules from large sets of discovered association rules," In CIKM, pp. 401-408, 1994.

[10] A. savasere, E. Omiecinski and S. Navathe, "An efficient algorithm for mining association rules in large databases," In VLDB, pp. 432-443, 1995.

[11] R. Srikant, Q. Vu and R. Agrawal, "Mining association rules with item constraints," In KDD, pp. 67-73, 1997.

[12] S. Sarawagi, S. Thomas and R. Agrawal, "Integrating association rule mining with relational database systems: Alternatives and implications," In SIGMOD, pp. 343-354, 1998.

[13] R. Agrawal, T. Imielinski and A. Swami, "Mining association rules between sets of items in large database," In ACM SIGMOD, pp.207-216, 1993.

[14] 박종수, 유원경, 홍기형 "연관 규칙 탐사와 그 응용", 한국 정보과학회지 제16권 제 9호, pp. 37-44, 1998.

채 덕 진

1999년 동신대학교 컴퓨터학과 졸업(이학사). 2001년 전남대학교 대학원 전산학과 졸업(이학석사). 2004년 전남대학교 대학원 전산학과 박사수료. 2004년~현재 전남대학교 전자컴퓨터정보통신공학부 강사. 관심분야는 데이타마이닝, Bioinformatics 등

황 부 현

1978년 숭실대학교 전산학과(학사). 1980년 한국과학기술원 전산학과(공학석사) 1994년 한국과학기술원 전산학과(공학박사). 1980년~현재 전남대학교 전자컴퓨터정보통신공학부 교수. 관심분야는 이동 데이타베이스, 데이타마이닝, 멀티미디어 데이타베이스, 객체지향 시스템