대용량 주기억장치 시스템에서 효율적인 연관 규칙 탐사 알고리즘

이 재 문†

요 약

본 논문은 대용량 주기억장치를 가진 시스템에 적합한 연관 규칙 탐사 알고리즘에 관한 연구이다. 이를 위하여 먼저 기존의 잘 알려진 알고리즘인 *DHP*, *Partition* 방법을 대용량 주기억장치를 가진 시스템에서 효율적으로 동작하도록 확장하였고, 다음 *Partition* 방법에 대해서 해쉬 테이블과 비트맵 기법을 적용하여 *Partition* 방법을 개선하는 방법을 제안하였다. 제안된 알고리즘은 실험적 환경에서 *DHP*와 성능이 비교되었으며, 제안하는 알고리즘이 확장된 *DHP*보다 최대 65%까지 성능 개선 효과가 있음을 보인다.

An Efficient Algorithm For Mining Association Rules In Main Memory Systems

Jae-Moon Lee†

ABSTRACT

This paper propose an efficient algorithm for mining association rules in the large main memory systems. To do this, the paper attempts firstly to extend the conventional algorithms such as *DHP* and *Partition* in order to be compatible to the large main memory systems and proposes secondly an algorithm to improve *Partition* algorithm by applying the techniques of the hash table and the bit map. The proposed algorithm is compared to the extended *DHP* within the experimental environments and the results show up to 65% performance improvement in comparison to the expanded *DHP*.

키워드 : 데이터 마이닝(Data Mining), 연관 규칙(Association Rule), 빈발 항목집합(Large Itemset), 해쉬 트리(Hash Tree), 해쉬 테이블(Hash Table), 비트맵(Bit Map)

1. 서 론

급격히 증가하는 전산화된 데이터로부터 내포된 정보를 추출하는 기술이 데이터 마이닝 분야에서 활발히 연구되고 있다[1, 6, 9, 10]. 특히 데이터 마이닝 분야의 한 분야인 연관 규칙 탐사에 많은 연구가 진행되고 있는 것은 연관 규칙이 다른 분야보다도 직접적인 활용도가 높은 이유도 있지만 순차패턴 등 고급 정보 가공의 기반 데이터로 활용되기 때문이다[1, 5, 6]. 연관 규칙은 하나의 트랜잭션이 다수의 항목을 포함하고 있을 때, 이러한 트랜잭션 데이터에서 한 항목들의 그룹과 다른 항목들의 그룹 사이에 연관성의 정도를 찾아내는 것이다. 예를 들어, 네 개의 항목으로 구성된 항목집합 *ABCD*를 포함하는 트랜잭션이 전체 트랜잭션들 중에서 10%이고 *AB*를 포함하는 트랜잭션의 95%가 *CD*도 포함한다고 하자. 이 사실을 연관 규칙 $AB \Rightarrow CD$로 나타내고, 이 규칙의 지지도는 10%이고 신뢰도는 95%이라고 한다. 연관 규칙 탐사는 규칙들의 유효성을 높이기 위하여 대용량 데이터베이스에서 수행하는 것이 일반적이다.

연관 규칙 탐사 문제는 주어진 데이터베이스에서 빈발 항목집합을 찾는 것과 찾아진 빈발 항목집합으로부터 연관 규칙을 생성하는 것으로 나누어지는데, 대부분의 연구는 빈발 항목집합을 효율적으로 찾는데 집중되고 있다. 빈발 항목집합을 찾는 기존의 방법은 대부분 탐사되어야 하는 데이터베이스가 너무 커서 데이터베이스가 파일과 같은 보조기억장치에 저장되어 있고, 필요시 이들을 순차적으로 액세스하는 방법으로 연구되었다[2, 3]. *Apriori, DHP*에서는 항목집합의 크기를 1씩 증가하면서 빈발 항목집합을 탐사하는데 이때마다 전체 데이터베이스를 스캔하고 있다. 특히

※ 본 논문은 2002년도 한성대학교 교내연구비 지원과제임.
† 정 회 원 : 한성대학교 컴퓨터공학부 교수
논문접수 : 2002년 4월 18일, 심사완료 : 2002년 6월 4일

*DHP*에서는 이를 효과적으로 하기 위하여 데이터베이스 전지 기법을 사용하여 더 이상 연관 규칙 탐사에 필요하지 않은 항목은 트랜잭션에서 미리 제거함으로써 데이터베이스 크기를 줄이는 방법을 제안하였다. 이의 효과는 데이터베이스 스캔 비용과 해쉬 트리 탐색 비용을 크게 줄여 주었다[3]. *Partition* 방법에서는 주기억장치의 크기로 데이터베이스를 분할하고, 각 분할된 데이터베이스를 주기억장치에 상주시켜 부분적으로 만족되는 빈발 항목집합을 찾은 후, 각각 찾아진 빈발 항목집합을 전체의 후보 항목집합으로 만든 후 전체 데이터베이스를 만족하는 빈발 항목 집합을 찾는 방법을 제안하였다. 이 방법은 데이터베이스를 두 번만 스캔함으로써 빈발 항목집합을 찾을 수 있는 장점이 있으나, 주기억장치가 작아 분할된 데이터베이스의 수가 많아지는 경우 급격한 성능 저하 현상을 일으키는 단점이 있다[4].

본 논문은 최근 주기억장치의 가격 하락으로 대부분의 시스템이 대용량의 주기억장치를 채용하고 있다는 가정하에 이러한 시스템에 적합한 연관 규칙 탐사 알고리즘에 관한 연구이다. 이를 위하여 기존의 잘 알려진 *DHP*[3] 및 *Partition*[4] 방법을 대용량 주기억장치가 채용된 시스템에 적합하도록 확장하였으며, 또한 *Partition* 방법에 대하여 보다 효율적인 알고리즘을 제안한다. *DHP* 방법의 확장 내용으로는 모든 데이터베이스를 주기억장치에 적재할 수 있도록 하였으며, 또한 해쉬 테이블의 크기를 가능한 크게 하여 해싱할 때 발생하는 충돌 현상을 최소화함으로써 후보 항목집합의 수를 가능한 최소화하도록 확장하였다. *Partition* 방법에서는 모든 데이터베이스를 주기억장치에 적재함으로써 데이터베이스의 분할을 없도록 하였으며, 또한 DHP에서 적용된 해쉬 테이블 기법을 부분적으로 도입하여 후보 항목집합의 수를 줄일 수 있도록 개선하였다. 특히 *Partition* 방법의 성능을 개선하기 위하여 트랜잭션 목록을 합병할 때 비트맵을 이용하도록 개선함으로써 성능을 향상시키는 알고리즘을 제안한다. 성능 비교를 위하여 제안된 방법들을 구현하였으며, 실험적 데이터를 이용하여 그들의 성능을 비교하였다.

본 논문은 2절에서 연관 규칙 및 기존의 연관 규칙 탐사 방법을 주기억장치가 충분한 환경에 적합하도록 확장하는 방법을 설명하며, 3절에서는 *Partition* 방법을 개선하는 알고리즘을 제안한다. 4절에서는 실험적 성능 비교를 하였으며, 5절에서 결론을 논하였다.

2. 문제의 정의 및 기존 방법의 확장

2.1 연관 규칙 탐사의 문제

연관 규칙 탐사에 대한 문제의 정의는 다음과 같이 표현한다. $I=\{i_1, i_2, \cdots, i_m\}$를 m개의 중복이 없는 항목의 집합이라 하자. 트랜잭션 T는 고유한 식별자(TID)를 가지고 있으며 I의 부분집합인 항목들의 집합이다. 데이터베이스 D는 이러한 트랜잭션 T의 집합이다. $X \subset I$, $Y \subset I$이고 $X \cap Y = \emptyset$인 경우에 $X \Rightarrow Y$의 형식으로 연관 규칙을 나타낸다. 이때 연관 규칙 $X \Rightarrow Y$는 다음과 같은 두 가지 조건을 만족해야 한다.

- $X \cup Y$가 빈발 항목집합이어야 한다.
- $\sup(X \cup Y) / \sup(X)$로 정의되는 신뢰도가 사용자가 지정하는 최소 신뢰도(minConf)보다 커야 한다.

상기에서 $\sup(X)$는 D의 트랜잭션 중 X를 포함하는 트랜잭션들의 개수를 의미한다. |D|를 D에 포함된 트랜잭션들의 개수라 할 때 $X \subseteq I$인 X가 $\sup(X) \geq |D| \times s$를 만족하면 항목집합 X를 빈발 항목집합(large itemset)이라 말한다. 여기서 s는 사용자가 지정한 최소 지지도(minSup)이다.

〈표 1〉 예제 데이터베이스

트랜잭션	T_1	T_2	T_3	T_4	T_5	T_6
구성항목	*AC*	*BCE*	*ABCE*	*BE*	*ACE*	*BCDE*

예를 들어 <표 1>의 예제 데이터베이스에서 사용자에 의하여 주어지는 최소 지지도 및 최소 신뢰도가 각각 50%와 90%라고 하자. 이때 빈발 항목집합은 $\{A\}$, $\{B\}$, $\{C\}$, $\{E\}$, $\{AC\}$, $\{BC\}$, $\{BE\}$, $\{CE\}$, $\{BCE\}$이고 연관 규칙은 $A \Rightarrow C$, $B \Rightarrow E$, $BC \Rightarrow E$이다.

기존의 연관 규칙 탐사 문제는 2개의 부문제로 분할하여 해결한다. 첫 번째 부문제는 주어진 데이터베이스에서 최소 지지도 이상 발생하는 항목집합, 즉 빈발 항목집합을 찾는 것이고, 두 번째 부문제는 찾아진 빈발 항목집합을 사용하여 최소 신뢰도를 만족하는 규칙을 생성하는 것이다. 주어진 빈발 항목집합으로 연관 규칙을 생성하는 것은 비교적 단순한 문제이다[2, 3]. 따라서 기존 연구의 대부분이 연관 규칙 탐사의 두 부문제중 첫 번째인 빈발 항목집합을 효율적으로 찾는 것에 중점을 두고 있다.

본 논문에서는 표현을 단순화하기 위하여 <표 2>의 기호를 사용한다. $|I|$가 n인 경우 모든 항목의 값은 0부터 $(n-1)$중의 하나의 값으로 가정하였고 데이터베이스내의 하나의 트랜잭션은 실제 항목의 이름으로 구성되는 있는 것이 아니라 이러한 값들의 집합으로 구성한다고 가정한다. k-항목집합의 의미는 항목집합에서 항목의 수가 k개라는 의미이다. 예를 들면, 항목집합 *ABD*는 *3*-항목집합이 된다.

〈표 2〉 기호 및 의미

기 호	내 용
D	트랜잭션을 포함하는 데이터베이스
T	데이터베이스내의 하나의 트랜잭션, $T \in D$
C_k	항목의 수가 k인 후보 항목집합(*candidate itemset*)
L_k	항목의 수가 k인 빈발 항목집합(*large itemset*)
$\|P\|$	집합 P의 원소의 개수
H_k	항목의 수가 k인 항목집합의 해쉬 테이블
TL_k	항목의 수가 k인 항목집합의 TID 목록

2.2 DHP 방법의 확장

빈발 항목집합을 찾는 여러 알고리즘들이 제안되었다[2-4, 7]. 이러한 방법들의 공통점은 후보 항목집합을 생성하고, 데이터베이스를 스캔하여 후보 항목집합에 대하여 발생 수를 계산한다. 그리고 이러한 후보 항목집합으로부터 최소 지지도를 만족하는 후보 항목집합을 빈발 항목집합으로 생성한다. 다시 생성된 빈발 항목집합으로부터 후보 항목집합을 생성하고 계속적으로 앞의 내용을 반복하는데, 이것은 더 이상 빈발 항목집합이 생성되지 않을 때까지 반복된다. *Apriori* 방법에서는 $(k-1)$-빈발 항목집합에서 k-후보 항목집합을 다음과 같이 생성한다. $l_1, l_2 \in L_{k-1}$이고 l_1, l_2이 각각 $(l_1[1] = x_1, l_1[2] = x_2, \cdots, l_1[k-2] = x_{k-2}, l_1[k-1] = x)$ $(l_2[1] = x_1, l_2[2] = x_2, \cdots, l_2[k-2] = x_{k-2}, l_2[k-1] = y)$ 항목으로 구성되어 있을 때 $c = l_1[1]\ l_1[2] \cdots l_1[k-2]\ l_1[k-1]\ l_2[k-1]$라는 새로운 항목집합을 생성한 후 c의 모든 $(k-1)$-부분항목집합이 L_{k-1}에 존재하는 것에 한하여 k-후보 항목집합으로 생성한다. 이러한 방법은 후보 항목집합을 생성하는 계산량을 줄여 주며, 후보 항목의 수를 크게 줄이는 효과를 얻을 수 있었다. *Apriori* 방법에서 k-후보 항목집합의 빈도수를 효율적으로 계산하기 위하여 해쉬 트리를 사용하였다. [3]에서 제시한 *DHP* 방법의 경우 *Apriori* 방법에 대한 전형적인 개선 알고리즘이다. *DHP* 방법에서는 후보 항목집합의 수를 줄이기 위하여 해쉬 테이블을 이용하였고 또한 더 이상 빈발 항목집합의 생성에 기여하지 못하는 트랜잭션의 항목들을 사전에 전지함으로써 k-후보 항목집합의 빈도수의 계산량을 현저히 줄였다. 이는 *Apriori* 방법에 비하여 탁월한 성능 향상 효과를 가져왔다[3].

DHP 방법이 주기억장치가 충분한 시스템에서 효율적으로 동작하기 위해서는 크게 두 가지 확장 요소가 있다. 하나는 각 단계별로 적용되는 해쉬 테이블의 크기를 충분히 크게 하는 것이며, 다른 하나는 데이터베이스를 주기억장치에 상주시키는 것이다. 후자의 경우는 매우 단순한 일이나, 전자의 경우 해쉬 테이블의 초기화에 들어가는 비용을 고려하여 그 크기를 정해야 한다. [3]에 의하면 H_2의 경우 *DHP* 성능에 결정적인 영향을 주므로 가능한 크게 하여야 하나, $H_k(k>2)$에 대해서는 성능에 크게 영향을 주지 못하기 때문에 그 크기를 크게 하여도 초기화 등의 비용이 많이 들어 크게 성능 향상에 기여하지 못하는 것을 실험을 통하여 알수 있었다. 따라서 본 논문에서는 $k>2$에 대해서는 H_k의 크기는 $|C_{k-1}|$의 두배로 제한하였다. 다음은 DHP 방법을 주기억장치가 충분한 시스템에 적합하도록 확장한 것이다.

```
스텝01 : 전체 데이터베이스를 주기억장치에 적재한다.
스텝02 : H2의 크기를 |I|×(|I|-1)/2로 한다.
스텝03 : D1(=D)을 스캔하여 L1과 H2를 생성한다.
스텝04 : for ( k=2 ; |Dk-1|!=0 ; k++) do begin
스텝05 : Lk-1과 Hk를 이용하여 Ck을 생성한다
스텝06 : Hk+1의 크기를 2×|Ck|로 한다.
스텝07 : Dk-1를 스캔하여 Ck의 발생수를 계산한다.
스텝08 : 스캔된 Dk-1로부터 Hk+1과 주기억장치에 Dk를 생성한다.
스텝09 : c ∈ Ck인 c의 발생수가 minSup보다 크면 Lk=Lk ∪{c}
         를 한다.
스텝10 : end
스텝11 : L={L1, L2, ···, Lk}을 출력한다.
```

데이터베이스 D를 주기억장치에 적재하는 것은 다양한 방법에 의하여 구현될 수 있다. 본 논문에서는 순차적인 접근이 가장 효과적으로 일어날 수 있도록 하나의 트랜잭션은 다음의 트랜잭션의 위치를 저장하도록 하였다. 또한 감소된 데이터베이스 D_k 생성을 효과적으로 하기 위하여 D_k는 D_{k-1}의 위치에 겹쳐 쓰도록 함으로써 항상 데이터베이스가 주기억장치에 상주하도록 하였다. H_2의 크기를 $\frac{|I|\times(|I|-1)}{2}$로 제한한 것은 $|I|$개의 항목으로 가능한 모든 종류의 2-항목집합의 수가 $\frac{|I|\times(|I|-1)}{2}$이기 때문이다.

2.3 *Partition* 방법의 확장

Apriori, *DHP* 방법은 후보 항목집합의 발생 수를 계산하기 위하여 해쉬 트리를 사용하였다. [4]에서 제시한 *Partition* 방법은 이들과 대조적으로 후보 항목집합의 발생 수를 계산하기 위하여 TID 목록을 이용한다. 이 방법은 임의의 후보 항목집합의 서브 빈발 항목집합의 TID 목록들을 교집합하여 교집합 목록의 TID 수에 따라 빈발 항목집합 여부를 판별하는 것이다. 이것을 하기 위해서는 빈발 항목집합의 TID 목록을 주기억장치에 상주 시켜야 한다. [4]에서는 제한된 주기억장치를 가진 시스템임을 가정하여 주어진 데이터베이스를 주기억장치에 상주시킬 수 있도록 데이터베이스를 분할하여 빈발 항목집합을 탐사하였다. 먼저 분할된

데이터베이스에 대하여 각각의 빈발 항목집합을 찾고, 찾아진 모든 빈발 항목집합에 대하여 이들의 합집합을 찾은 후, 이들에 대하여 한번의 데이터베이스 스캔으로 전체적인 빈발 항목집합을 계산하였다.

Partition 방법은 알고리즘 자체가 데이터베이스의 크기에 비하여 주기억장치의 크기가 작다는 환경하에 제안된 방법이다. 따라서 *Partition* 방법을 주기억장치가 충분한 시스템에 적합하도록 개선하는 것은 단순히 데이터베이스를 분할하지 않고 한번에 빈발 항목집합을 찾으면 된다. 다음은 Partition 방법을 확장한 것이다.

스텝01 : 전체 데이터베이스를 스캔하여 L_1과 그들의 TID 목록을 생성한다.
스텝02 : for (k = 2 ; $|L_{k-1}|$! = 0 ; k++) do begin
스텝03 : forall $l_1 \in L_{k-1}$ do begin
스텝04 : L_x = { c | $c \in L_{k-1}$, $c[1] = l_1[1] \wedge \cdots \wedge c[k-2] = l_1[k-2] \wedge c[k-1] \neq l_1[k-1]$}
스텝05 : forall l_2 in L_x do begin
스텝06 : $c = l_1[1]\, l_1[2] \cdots l_1[k-1]\, l_2[k-1]$로 구성된 c를 생성한다.
스텝07 : c의 k-1 서브 항목집합들이 L_{k-1}에 모두 없으면 스텝 05를 수행한다.
스텝08 : l_1의 TID 목록과 l_2의 TID 목록을 교집합하여 c의 TID 목록을 생성한다.
스텝09 : c의 TID 목록의 개수가 minSup보다 크면 $L_k = L_k \cup \{c\}$를 한다.
스텝10 : end
스텝11 : end
스텝12 : end
스텝13 : $L = \{L_1, L_2, \cdots, L_k\}$을 출력한다.

스텝 1에서 전체 데이터베이스를 스캔하여 L_1과 그들의 TID 목록을 만든다는 것은 트랜잭션별로 그룹화된 항목집합을 항목별로 트랜잭션을 그룹화하는 것이다. 따라서 구현에 따라 메모리 소요량은 차이가 있으나, 전체 데이터베이스의 내용이 주기억장치에 모두 적재된 것과 같게 된다. 예를 들어 <표 1>의 데이터베이스에 대하여 최소지지도 50%를 만족하는 빈발 항목집합을 찾으면 $L_1 = \{A, B, C, E\}$이 되며, 그들의 TID 목록은 <표 3>과 같다.

〈표 3〉 L_1 및 TID 목록

$l_i \in L_1$	l_i의 TID 목록
A	1, 3, 5
B	2, 3, 4, 6
C	1, 2, 3, 5, 6
E	2, 3, 4, 5, 6

스텝 04에서 $c[i]$, $l_1[i]$는 각각 c, l_1의 항목집합 중 i번째 항목을 의미한다. 스텝 04에서 L_x를 찾는 것은 L_{k-1}에서 항목집합이 항목값으로 정렬되어 유지되기 때문에 특별한 부가적인 비용을 요구하지 않는다. 상기 방법에서 가장 많은 계산량을 갖는 것은 스텝 08에서 두 개의 TID 목록을 교집합하여 하나의 새로운 TID 목록을 생성하는 것이다. 이것을 효과적으로 하기 위하여 모든 TID 목록은 정렬된 상태로 유지되도록 하는 것이 중요하다. 이 정렬은 L_1을 생성할 때 그들의 TID 목록을 정렬한 상태로 생성할 수 있으므로 정렬을 위한 별도의 비용도 필요하지 않다. <표 3>에서 A, B의 TID 목록을 얻기 위해서는 A, B의 TID 목록인 {1, 3, 5}와 {2, 3, 4, 6}을 교집합하여 {3}이라는 AB의 TID 목록을 얻게된다.

3. 대용량 주기억장치에서 효율적인 연관 규칙 탐사 알고리즘

DHP 방법은 반복적으로 데이터베이스를 스캔하고, 저장한다. 따라서 주기억장치가 작아서 전체 데이터베이스를 적재할 수 없는 경우 보조기억장치를 반복적으로 액세스하여야 한다. 그러나 주기억장치가 충분하여 전체 데이터베이스를 주기억장치에 적재하면, 보조기억장치의 액세스는 전혀 없게 되어, 더 이상 주기억장치의 용량에 따른 성능의 의존도는 크지 않게 된다. 반면, *Partition* 방법에서는 주기억장치를 이용한 성능 개선의 여지가 있다. 본 논문에서는 주기억장치가 충분한 시스템에서 *Partition* 방법의 성능 개선을 위한 두 가지 방법을 제안한다. 제안된 방법의 하나는 *DHP*에서와 같이 해쉬 테이블을 사용하여 후보 빈발항목의 수를 줄임으로써 TID 목록의 교집합 횟수를 줄이는 것이며, 다른 하나는 전체 트랜잭션 수의 크기로 된 비트맵을 사용하여 TID 목록의 교집합을 효율적으로 계산하는 것이다.

3.1 해쉬 테이블을 사용한 *Partition* 방법의 개선

DHP 방법에서는 해쉬 테이블을 사용하여 후보 항목집합의 수를 *Apriori* 방법에 비하여 현저히 적게 함으로써 큰 성능 개선 효과를 가져왔다. 본 논문에서도 *DHP*에서 사용한 기법인 해쉬 테이블 기법을 적용하여 가능한 후보 빈발항목의 수를 최소화하는 기법을 *Partition* 방법에 적용하여 성능을 개선하는 것을 제안한다. *DHP*에서는 $k \geq 2$인 모든 경우에 대하여 해쉬 테이블을 생성하고, 각 단계의 후보 빈발항목들에 대하여 해쉬 테이블을 검색하여 후보 빈발항목의 가능성을 검사하였다. 그러나 실제 상황에서는 $k \geq 3$

인 경우에는 H_k를 생성하는 비용이 너무 많아서 성능 개선에 크게 기여하지 못하는 것을 실험을 통하여 알 수 있었다. *Partition* 방법에서는 $k \geq 3$에 대하여 H_k를 적용하는 것은 H_k를 생성하는 것에 너무 많은 비용이 들어가기 때문에 용이하지는 않다. 왜냐하면, 각 k에 대하여 H_k를 생성하기 위해서는 전체 데이터베이스를 스캔하여 모든 가능한 k-항목집합을 계산하여 H_k를 설정하여야 하는데 이 비용이 너무 크기 때문이다. *DHP*에서 이것이 가능한 이유는 각 k에 대하여 데이터베이스를 전지하여 데이터베이스 크기가 줄어들기 때문에 그 비용이 비교적 적게 들기 때문이다. 따라서 *Partition* 방법에서 해쉬 테이블의 적용은 $k=2$인 경우에 한하여 적용한다. 다음은 2장에서 확장한 *Partition* 방법에 해쉬 테이블을 사용하여 후보 항목집합을 최소화하기 위하여 개선한 알고리즘이다. 이 알고리즘을 $Partition_1$라고 하기로 한다.

스텝01 : 전체 데이터베이스를 스캔하여 L_1과 그들의 TID 목록과 H_2을 생성한다.
스텝02 : for($k=2$; $|L_{k-1}|!=0$; k^{++}) do begin
스텝03 : forall $l_1 \in L_{k-1}$ do begin
스텝04 : $L_x=\{c \mid c \in L_{k-1}, c[1]=l_1[1]\wedge\cdots\wedge c[k-2]=l_1[k-2]\wedge c[k-1] \neq l_1[k-1]\}$
스텝05 : forall l_2 in L_x do begin
스텝06 : $c=l_1[1]\,l_1[2]\cdots l_1[k-1]\,l_2[k-1]$로 구성된 c를 생성한다.
스텝6.5 : if($k==2$ && $H_2(c)$<minSup) 스텝 05를 수행한다.
스텝07 : c의 k-1 서브 항목집합들이 L_{k-1}에 모두 없으면 스텝 05를 수행한다.
스텝08 : l_1의 TID 목록과 l_2의 TID 목록을 교집합하여 c의 TID 목록을 생성한다.
스텝09 : c의 TID 목록의 개수가 minSup보다 크면 $L_k=L_k \cup \{c\}$를 한다.
스텝10 : end
스텝11 : end
스텝12 : end
스텝13 : $L=\{L_1, L_2, \cdots, L_k\}$을 출력한다.

스텝 01에서는 *DHP*와 같이 전체 데이터베이스를 스캔하면서 모든 트랜잭션의 모든 서브 2-항목집합에 대하여 그들의 발생수를 H_2에 저장하는 것이 추가되었다. 스텝 6.5는 스텝 06에서 생성된 하나의 2-후보 항목집합에 대하여 $H_2(c)$를 찾아서 이의 값이 최소지지도 보다 작은 경우 후보 항목집합에서 제외하는 것이다. 여기서도 주기억장치가 충분한 상태이므로 H_2의 크기를 $\frac{|I|\times(|I|-1)}{2}$로 하여 해쉬 테이블에서 충돌 현상이 발생하지 않도록 하였다.

3.2 비트맵을 이용한 TID 목록 생성 방법의 개선

앞절에서 언급하였듯이 *Partition* 방법에서 가장 많은 비용을 요구하는 부분이 후보 빈발항목의 TID 목록을 생성하기 위하여 두 개의 빈발항목의 TID목록을 교집합하는 스텝 08이다. 스텝 03에서 선택된 하나의 l_1에 대하여 $l_i \in L_x$인 모든 l_i와 TID 목록 교집합을 구하여야 한다. 따라서 *Partition* 방법에서 대부분의 비용은 TID 목록을 교집합하는데 있다. 이것은 *Partition* 방법의 성능 향상을 위해서는 TID 목록의 교집합에 대한 효율적인 기법이 요구된다는 것을 의미한다. [4]에서는 두 개의 TID 목록에 대한 교집합을 생성하기 위하여 모든 TID 목록을 TID 값에 따라 정렬된 상태를 유지하였으며, 정렬된 두 개의 TID 목록은 합병(Merge) 방법을 사용하여 효과적으로 교집합을 구하도록 하였다. 본 논문에서는 하나의 TID 목록(l_1)에 대하여 다수의 TID 목록($l_i \in L_x$)이 순차적으로 여러번 교집합되어야 한다는 것에 착안하여 l_1의 TID 목록을 효율적으로 액세스하는 방법으로 l_1에 대한 비트맵을 생성하여 이를 이용하는 방법을 제안한다. 제안하는 방법은 l_1의 TID 목록에 대응하는 하나의 비트맵을 생성한다. 이때 비트맵의 초기 값은 0이고, l_1의 TID 목록에 있는 TID에 대응하는 비트맵의 비트는 1로 설정한다. 예를 들어 <표 3>의 L_1과 TID 목록을 고려하자. l_1=A라고 하면 $L_x=\{B, C, E\}$가 된다. 따라서 l_1에 의하여 생성되는 비트맵(*bmap*)의 크기는 6이며, 그 내용은 다음 (그림 1)과 같다.

bmap[1]	*bmap*[2]	*bmap*[3]	*bmap*[4]	*bmap*[5]	*bmap*[6]
1	0	1	0	1	0

〈그림 1〉 l_1에 대한 비트 맵

이 비트맵 *bmap*을 이용하여 *AB*의 TID 목록을 구하는 방법은 B의 TID 목록인 {2, 3, 4, 6}에서 각 원소들에 대하여 순차적으로 *bmap*[2], *bmap*[3], *bmap*[4], *bmap*[6]을 참조하여 그 값이 1인 3이 *AB*의 TID 목록에 포함되는 것이다. 따라서 이 경우 *AB*의 TID 목록은 {3}이 되고 {3}의 원소의 개수는 1이기 때문에 minSup를 만족하지 못하므로 *AB*는 L_2에 포함되지 못하게 된다. *AC*, *AE*의 TID 목록을 얻기 위해서도 *C*, *E*의 TID 목록에 대하여 같은 방법으로 계산하여야 한다. 다음은 비트맵을 이용하여 효율적으로 TID 목록의 교집합을 구하는 방법을 $Partition_1$에 적용한 알고리즘이다. 이 알고리즘을 $Partition_2$라고 하기로 한다.

스텝01 : 전체 데이터베이스를 스캔하여 L_1과 그들의 TID 목록과 H_2을 생성한다.
스텝02 : for($k=2$; $|L_{k-1}| != 0$; k^{++}) do begin
스텝03 : forall $l_1 \in L_{k-1}$ do begin
스텝3.5 : l_1의 TID에 대하여 비트맵 *bmap*을 생성한다.
스텝04 : $L_x = \{ c \mid c \in L_{k-1}, c[1] = l_1[1] \wedge \cdots \wedge c[k-2] = l_1[k-2] \wedge c[k-1] \neq l_1[k-1] \}$
스텝05 : forall l_2 in L_x do begin
스텝06 : $c = l_1[1]\, l_1[2] \cdots l_1[k-1]\, l_2[k-1]$로 구성된 c를 생성한다.
스텝6.5 : if($k == 2$ && $H_2(c) <$ minSup) 스텝 05를 수행한다.
스텝07 : c의 $k-1$ 서브 항목집합들이 L_{k-1}에 모두 없으면 스텝 05를 수행한다.
스텝08 : l_1의 *bmap*와 l_2의 TID 목록을 이용하여 c의 TID 목록을 생성한다.
스텝09 : c의 TID 목록의 개수가 minSup보다 크면 $L_k = L_k \cup \{ c \}$를 한다.
스텝10 : end
스텝11 : end
스텝12 : end
스텝13 : $L = \{ L_1, L_2, \cdots, L_k \}$을 출력한다.

상기 알고리즘은 *Partition*$_1$에서 스텝 3.5가 추가되었으며, 스텝 08이 수정되었다.

4. 성능 비교

본 논문에서는 대용량 주기억장치에서 *DHP*와 제안하는 알고리즘의 성능을 비교하기 위하여 *DHP*, *Partition*$_1$ 및 *Partition*$_2$ 알고리즘 C++로 구현하였다. 구현 및 성능 비교 환경으로는 512M 주기억장치를 가진 펜티움 III 800MHZ 컴퓨터이고, 운영체제는 윈도우즈2000이다. 성능 비교는 *DHP*에 대하여 *Partition*$_1$ 및 *Partition*$_2$을 상대적으로 비교하였다.

〈표 4〉 실험 데이터베이스 생성을 위한 매개 변수

매개변수	내 용
$\|D\|$	데이터베이스에서의 트랜잭션의 수
$\|T\|$	트랜잭션에서의 항목의 평균 수
$\|I\|$	잠재적인 최대 빈발 항목집합의 평균 크기
$\|L\|$	잠재적인 최대 빈발 항목집합의 수
N	항목의 수

본 논문에서는 [9]에서 제공하는 프로그램을 수행하여 실험 데이터베이스를 생성하였다. [9]에서 제공하는 프로그램은 *Apriori* 방법의 성능 측정을 위하여 개발된 프로그램으로써, 이 프로그램에 의하여 생성된 데이터는 실제 상황에서 생성되는 배스킷 데이터의 특징을 잘 포함하고 있는 것으로 알려져 있다[2]. <표 4>는 실험 데이터베이스를 생성할 때 사용하는 매개 변수를 나타내며, <표 5>는 각각의 매개 변수 값에 따른 실험 데이터베이스의 이름 및 크기를 나타낸다. 여기서 $|L|$과 N은 [2, 3, 4]에서와 같이 각각 2000과 1000으로 고정하였다.

〈표 5〉 실험 데이터베이스 및 매개 변수 값

데이터베이스 이름	$\|T\|$	$\|I\|$	$\|D\|$	크기
T05I2D100	5	2	100,000	2.4MB
T10I2D100	10	2	100,000	4.4MB
T10I4D100		4	100,000	
T20I2D100	20	2	100,000	8.4MB
T20I4D100		4	100,000	
T20I6D100		6	100,000	

<표 6>는 <표 5>의 6개의 데이터베이스에 대하여 *DHP* 방법과 개선된 *Partition* 방법 사이의 성능 비교의 결과를 나타내고 있다. <표 6>에서 *Partition*$_1$, *Partition*$_2$는 각각 3.1절과 3.2절에서 제안한 알고리즘이다. <표 6>에서 *DHP* 값은 실제 실험적 환경에서 수행 시간을 측정한 값으로 단위는 $\frac{1}{1000}$초(msec)이고, *Partition*$_1$, *Partition*$_2$는 *DHP*에 대한 상대적 성능 향상율을 나타내는 이득으로 다음 공식에 의하여 계산되었다.

$$G_i(\%) = \frac{DHP\text{의 실행시간} - Partition_i\text{의 실행시간}}{DHP\text{의 실행시간}} \times 100(\%)$$

〈표 6〉 성능 측정 결과

데이터베이스 \ 알고리즘 최소지지도	확장된 *DHP*(msec)				*Partition*$_1$ (%)				*Partition*$_2$ (%)			
	1.00	0.75	0.50	0.33	1.00	0.75	0.50	0.33	1.00	0.75	0.50	0.33
T05I2D100	200	251	280	320	-20.5	4.4	10.7	12.2	-35.0	-4.0	3.6	3.1
T10I2D100	812	881	1022	1202	-2.3	2.2	4.0	0.8	-7.4	-4.5	1.1	3.3
T10I4D100	731	781	1161	1672	-10.9	-6.4	9.5	13.2	-17.8	-12.8	9.4	16.1
T20I2D100	3976	5367	11817	24455	1.3	-1.9	11.1	4.3	8.1	14.5	37.0	39.7
T20I4D100	3986	5838	12248	23794	3.5	13.2	28.4	27.3	7.0	21.9	44.0	49.5
T20I6D100	3405	5518	19378	38415	-6.5	8.3	49.6	51.6	-2.6	18.5	58.6	65.4

실험은 <표 6>에서 볼 수 있듯이 6개의 데이터베이스에 대하여 최소 지지도를 1.0, 0.75, 0.5, 0.33로 변화시키면서 세 가지 방법에 대하여 실행 시간 및 상대적 이득을 측정

하였다. 세 가지 알고리즘 모두 최소 지지도가 낮아지거나, 데이터베이스의 평균 트랜잭션 크기가 커지는 경우 실행 시간이 많아지는 공통점을 가지고 있다. 동일 데이터베이스에 대하여 최소 지지도가 낮다는 것의 의미는 탐사되어야 하는 빈발 항목집합이 많아진다는 것을 의미하며, 이것은 후보 항목집합을 증가시키는 원인이 되어 실행 시간이 많아지게 되는 것이다. 데이터베이스의 평균 트랜잭션의 크기가 큰 경우는 데이터베이스의 크기가 커지며 또한 탐사되는 빈발 항목집합이 증가하고, 이것 역시 후보 항목집합을 증가시키므로 실행시간이 많아지게 된다. 알고리즘별 성능은 데이터베이스와 최소 지지도에 따라 다르게 나타나고 있다. 데이터베이스의 평균 트랜잭션이 작고 그리고 최소 지지도가 낮은 경우에는 $Partition_1$, $Partition_2$보다 *DHP*가 우수한 것으로 나타나고, 그렇지 않은 경우에는 *DHP*보다 $Partition_1$, $Partition_2$가 대체로 우수한 것으로 나타나고 있다. 어쨌든 데이터베이스의 평균 트랜잭션이 작고 그리고 최소 지지도가 낮은 경우는 큰 문제가 되지 않는다. 이것은 탐사되는 빈발 항목집합이 적어 실행시간이 많지 않아서 실제적인 실행시간의 차이는 크지 않기 때문이다. 따라서 $Partition_1$, $Partition_2$의 경우처럼 데이터베이스가 크거나 최소 지지도가 낮은 경우에 우수한 성능을 나타내는 것이 더욱 중요하다. 또한 <표 6>은 데이터베이스와 최소 지지도에 관계없이 $Partition_1$에 비하여 $Partition_2$가 대부분의 경우 우수하다는 것을 보이고 있다. 이것은 비트맵을 이용한 TID 목록의 합병이 효과가 매우 크기 때문이다.

5. 결 론

본 논문은 최근 주기억장치의 가격 하락으로 대부분의 시스템이 대용량의 주기억장치를 채용하고 있다는 가정하에 이러한 시스템에 적합한 연관 규칙 탐사 알고리즘에 관한 연구하였다. 기존의 잘 알려진 *DHP* 및 *Partition* 방법을 대용량 주기억장치가 채용된 시스템에 적합하도록 확장하였다. 또한 *Partition* 방법의 성능을 개선하기 위하여 *DHP*에서 적용된 해쉬 테이블 기법을 부분적으로 도입하여 후보 항목집합의 수를 줄일 수 있도록 하였으며, 트랜잭션 목록을 합병할 때 비트맵을 이용함으로써 성능을 향상시키는 알고리즘을 제안하였다. 제안된 알고리즘은 기존의 알고리즘과 성능이 비교되었으며, 대부분의 환경에서 제안한 알고리즘이 우수한 성능을 보였으며, 확장된 *DHP*에 비하여 최대 65%까지 성능 이득을 나타내었다.

참 고 문 헌

[1] R. Agrawal, T. Imielinski and A. Swami, "Database Mining : A Performance Perspective," IEEE Trans. On Knowledge and Data Engineering, Vol.5, No.6, pp.914-925, 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," Proceedings of the 20th International Conference on Very Large Databases, 1994.

[3] J. S. Park, M.-S. Chen and P. S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," Proceedings of ACM SIGMOD, pp.175-186, 1995.

[4] A. Savasere, E. Omiecinski and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," Proceedings of the 21th International Conference on Very Large Databases, pp. 432-444, 1995.

[5] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," Proceedings of the 21th International Conference on Very Large Databases, pp. 407-419, 1995.

[6] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," Proceedings of the 21th International Conference on Very Large Databases, pp.420-431, 1995.

[7] H. Toivonen, "Sampling Large Databases for Association Rules," Proceedings of the 21th International Conference on Very Large Databases, pp.134-144, 1995.

[8] 박종수, "대용량 데이터베이스 상의 효과적인 연관 규칙 탐사를 위한 전지 기법", 한국정보과학회 산하 데이터베이스 연구회지, 제12권 제4호, pp.59-75, 1996.

[9] M-S Chen, J. Han, and Philip S. Yu, "Data Mining : An Overview from a Database Perspective," IEEE Transactions on Knowledge and Data Engineering, 8(6) : pp.866-883, 1996.

[10] R. Agrawal and et al, "Programs Generating Test Data in Data Mining," http://www.almaden.ibm.com/cs/quest, 1997.

[11] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkmo, "Fast Discovery of Association Rules," In Adavanes in Knowledge Discovery

and Data Mining, ed. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, pp.307-328, 1996.

[12] 이재문, 박종수, "복합 해쉬 트리를 이용한 효율적인 연관 규칙 탐사 알고리즘", 한국정보과학회논문지, 제26권 제3호, pp.343-352, 1999.

[13] 이재문, "대화형 환경에서 효율적인 연관 규칙 알고리즘", 정보처리학회논문지 D, 제8-D권 제4호, pp.339-346, 2001.

[14] 이재문, "대용량 주기억장치에서 연관 규칙 알고리즘 비교", 한성대학교논문집 2001.

이 재 문

e-mail : jmlee@hansung.ac.kr

1986년 한양대학교 전자공학과 졸업(학사)

1988년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)

1992년 한국과학기술원 전기 및 전자공학과 졸업(공학박사)

1992~1994년 한국통신 연구개발단 선임연구원

1994~현재 한성대학교 컴퓨터공학부 부교수

관심분야 : 데이터베이스, 데이터 마이닝, 멀티미디어, 정보처리, XML