대화형 환경에서 효율적인 연관 규칙 알고리즘

이 재 문†

요 약

대화형 환경에서 연관 규칙 탐사 문제는 동일한 데이터베이스에서 다른 최소 지지도로 반복적으로 연관 규칙을 탐사하는 것이다. 이 문제는 반복적으로 연관 규칙을 탐사한다는 사실만 기존의 연관 규칙 탐사와 다를 뿐 기존의 연관 규칙 탐사에서 발생하는 모든 문제를 포함한다. 본 논문은 전 단계에 계산된 후보 항목집합에 대한 정보를 이용함으로써 성능 향상을 가져오는 효율적인 알고리즘을 제안한다. 제안된 알고리즘은 대화형 환경에서 기존의 알고리즘과 수행 시간 측면에서 비교되었다. 성능 비교의 결과로부터 제안하는 알고리즘이 기존의 방법보다 약 10~30% 정도의 상대적 성능 향상 효과가 있음을 알 수 있었다.

Efficient Algorithms for Mining Association Rules Under the Interactive Environments

Jae-Moon Lee†

ABSTRACT

A problem for mining association rules under the interactive environments is to mine repeatedly association rules with the different minimum support. This problem includes all subproblems except on the facts that mine repeatedly association rules with the same database. This paper proposed the efficient algorithms to improve the performance by using the information of the candidate large itemsets which calculate the previous association rules. The proposed algorithms were compared with the conventional algorithm with respect to the execution time. The comparisons show that the proposed algorithms achieve 10~30% more gain than the conventional algorithm.

키워드 : 데이터 마이닝(Data Mining), 연관 규칙(Association Rules), 빈발 항목집합(Large Itemset), 해쉬 트리(Hash Tree)

1. 서 론

산업 각 분야에서의 업무 전산화 및 바코드와 같은 데이터 입력 시스템의 발달 등으로 유용한 정보를 포함하고 있는 데이터가 폭발적으로 증가하고 있다. 이러한 증가는 자연히 데이터로부터 유용한 정보 및 지식을 추출하는 새로운 기술을 요구하게 되었고, 이 요구에 가장 잘 부응하는 분야가 연관 규칙 탐사 등을 포함하는 데이터 마이닝 분야이다[1, 7, 9]. 이것은 연관 규칙이 다른 분야보다도 직접적인 활용도가 높은 이유도 있지만 순차패턴 등 고급 정보 가공의 기반 데이터로 활용되기 때문이다[1, 5, 6]. 이러한 연관 규칙 탐사는 탐사되는 규칙의 신뢰도 때문에 대용량 데이터베이스에서 탐사하는 것이 필수적이며, 또한 그 방법의 복잡성 때문에 응답 속도를 향상시키는 관점에서 많은 연구가 진행되어 왔다[2-4, 8, 11-14]. 이러한 연구를 기반으로 실용화된 연관 규칙 탐사 시스템이 개발됨에 따라 대화형 환경 등 다양한 측면에서 연관 규칙 탐사가 새로이 연구되고 있다[9]. 본 논문에서의 연구는 대화형 연관 규칙 탐사 분야인 동일한 데이터베이스에 대해서 다른 최소 지지도로 반복적인 연관 규칙 탐사가 요구되는 경우 이를 효율적으로 지원하는 연관 규칙 탐사 방법에 관한 연구로서[12, 14]에서 제안된 알고리즘을 개선하는 알고리즘을 제안한다.

연관 규칙은 하나의 트랜잭션이 다수의 항목을 포함하고 있을 때, 이러한 트랜잭션 데이터에서 한 항목들의 그룹과 다른 항목들의 그룹 사이에 강한 연관성이 있음을 나타내는 것이다. 예를 들어, 네 개의 항목으로 구성된 항목집합 $ABCD$를 포함하는 트랜잭션이 전체 트랜잭션들 중에서 10%이고 AB를 포함하는 트랜잭션의 95%가 CD도 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. 기존의 연구는 연관 규칙 탐사 문제를 두 개의 부문제로 분할하여 해결한다[2-4, 8, 11-14]. 첫 번째 부 문제는 주어진 데이터베이스에서 최소 지지도

※ 본 논문은 2001년도 한성대학교 교내연구비 지원과제임.
† 정 회 원 : 한성대학교 정보전산학부 교수
논문접수 : 2001년 3월 24일, 심사완료 : 2001년 6월 8일

이상 발생하는 항목집합, 즉 빈발 항목집합을 찾는 것이고, 두 번째 부 문제는 첫 부 문제에서 찾아진 빈발 항목집합에 대하여 최소 신뢰도를 만족하는 규칙을 생성하는 것이다. 주어진 빈발 항목집합으로 연관 규칙을 생성하는 것은 비교적 단순한 문제이다. 따라서 기존 연구의 대부분이 연관 규칙 탐사 문제를 두 개의 부문제 중 첫 번째인 빈발 항목집합을 효율적으로 찾는 것에 중점을 두고있다.

대화형 환경에서 연관 규칙 탐사 문제는 동일한 데이터베이스에 대하여 다른 최소 지지도로 반복적으로 연관 규칙을 탐사한다는 사실만 기존의 연관 규칙 탐사와 다를 뿐 기존의 연관 규칙 탐사에서 발생하는 모든 문제를 포함한다[12, 14]. 즉 효율적인 연관 규칙 탐사를 위하여 연관 규칙 탐사 문제가 두 개의 부 문제로 분리되어야 하며, 빈발 항목집합을 효율적으로 찾을 수 있어야 한다. 본 논문은 기존의 연관 규칙 탐사 기법을 활용하여 대화형 환경에서 효율적으로 연관 규칙을 탐사하는 알고리즘을 제안한다. 제안하는 알고리즘의 핵심 개념은 동일한 데이터베이스에 대하여 반복적으로 연관 규칙 탐사가 일어난다는 특징을 활용하여 새로운 연관 규칙 탐사할 때 가능한 이전의 연관 규칙 탐사에서 계산된 정보를 최대한 활용함으로써 계산 량을 줄여 새로운 연관 규칙을 효율적으로 탐사하자는 것이다.

본 논문은 2장에서 연관 규칙 및 대화형 환경에서 연관 규칙 탐사 문제를 설명하며, 3장에서는 기존의 알고리즘을 개선하는 연관 규칙 알고리즘을 제안한다. 4장에서는 실험적 성능 비교를 하였으며, 5장에서 결론을 논하였다.

2. 문제의 정의 및 기존의 해결 방법

2.1 대화형 환경에서 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(min_conf)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(min_sup)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.

[2]에서 제안한 *Apriori* 방법은 연관 규칙 탐사 문제를 최소 지지도를 만족하는 빈발 항목집합을 찾는 것과 탐사된 빈발 항목집합으로부터 최소 신뢰도를 만족하는 연관 규칙을 생성하는 2개의 부문제로 나뉘어 해결하였다. *Apriori* 방법에서는 효율적으로 빈발 항목집합을 찾기 위하여 후보 항목집합을 생성하는 방법을 제안하였는데 이를 *Apriori_gen*이라 하였다. 또한 후보 항목집합의 발생 수를 효율적으로 계산하기 위하여 해쉬 트리를 도입하였다. *Apriori* 방법을 개선한 다양한 방법들이 제안되었다[3, 4, 8, 13]. 특히 [3]에서 제안된 *DHP*는 해쉬 기법을 적용하여 *Apriori_gen*에서 생성된 후보 항목집합을 더욱 최소화함으로써 연관 규칙 탐사 알고리즘의 성능 향상에 탁월한 기여를 하였다. [13]에서 제안된 *CHT* 방법은 *Apriori*, *DHP* 방법에서 가장 큰 병목 지점인 데이터베이스 스캔 및 해쉬 트리 탐색 비용을 줄임으로써 기존의 방법보다 더욱 효율적으로 연관 규칙을 탐사하는 알고리즘이다.

대화형 환경에서 연관 규칙 탐사는 동일한 데이터베이스에 대하여 다른 최소 지지도로 반복적인 연관 규칙을 탐사하는 문제이다[9]. 대화형 환경에서 연관 규칙 탐사 과정은 다음과 같이 크게 5단계로 구분할 수 있다.

1. 최소 지지도 및 신뢰도를 입력받는다.
2. 최소 지지도 및 신뢰도로 연관 규칙을 탐사한다.
3. 탐사된 연관 규칙에서 특정 연관 규칙을 검색한다.
4. 특정 연관 규칙을 찾으면 종료한다.
5. 그렇지 않으면 최소 지지도 및 신뢰도를 재입력받고, 2, 3, 4 과정을 반복한다.

상기 5단계에서 최대 병목 현상은 당연히 선택된 최소 지지도 및 최소 신뢰도를 만족하는 연관 규칙의 탐사 과정이며, 그 중에서도 최소 지지도를 만족하는 빈발 항목집합의 탐사이다. 특히 단계 5에서 재 입력되는 최소 지지도 및 신뢰도는 기존의 값보다 낮은 값으로 선택되어야 한다. 왜냐하면 낮은 최소 지지도로 탐사된 빈발 항목집합은 항상 높은 최소 지지도로 탐사된 것들을 포함하기 때문에 낮은 최소 지지도에서 찾을 수 없는 빈발 항목집합은 당연히 높은 지지도에서도 찾을 수 없기 때문이다. 이러한 이유로 기존의 탐사 방법을 사용하여 연관 규칙을 탐사하는 경우 2~5단계

를 반복하면 할수록 급격히 성능 저하 현상이 일어나게 된다. 따라서 대화형 환경에서 연관 규칙 탐사는 이러한 문제를 효과적으로 해결하는 방법이 절실히 요구된다. [14]에서는 i번째 연관 규칙이 탐사되고 다시 새로운 최소 지지도에 의하여 $(i+1)$번째 연관 규칙 탐사가 수행될 때 i번째 연관 규칙에서 탐사되었던 빈발 항목집합에 대해서는 $(i+1)$번째 연관 규칙 탐사에서 제외함으로써 성능 향상을 가져오는 방법을 제안하였다.

본 논문은 대화형 환경에서 연관 규칙 탐사의 이러한 특징을 활용하여 효율적으로 연관 규칙을 탐사하는 알고리즘을 제안하는 것이다. <표 2>은 논문의 서술을 편리하게 하기 위한 기호이다.

〈표 2〉 기호 및 의미

기 호	의 미
D	연관 규칙에 적용되는 데이터베이스
k-항목집합	k개의 항목으로 구성된 항목집합
$C_{n(k)}$	s_n%에 의하여 생성된 k-후보 항목집합
$L_{n(k)}$	s_n%에 의하여 탐사된 k-빈발 항목집합
$CD_{n(k)}$	$C_{n(k)} - L_{n(k)}$
$H_2(c)$	2-항목집합 c에 대한 해쉬 테이블의 값
$HC_k(c)$	k-후보 항목집합 c에 대한 해쉬 테이블의 값

또한 전 단계의 연관 규칙 탐사를 o단계로 표시하며, 현 단계의 연관 규칙 탐사를 n단계로 표시한다. 따라서 전 단계의 연관 규칙 탐사의 최소 지지도, k-후보 항목집합 및 k-빈발 항목집합을 각각 s_o, $C_{o(k)}$, $L_{o(k)}$로 표시하며, 현 단계의 연관 규칙 탐사의 최소 지지도, k-후보 항목집합 및 k-빈발 항목집합을 각각 s_n, $C_{n(k)}$, $L_{n(k)}$로 표시한다.

2.2 기존의 해결 방법

[14]에서는 대화형 환경에서 효율적인 연관 규칙 탐사 방법으로 *DEC*와 *DEC'*를 제안하였다. 이 방법의 주요 개념은 전 단계에서 탐사된 빈발 항목집합에 대해서는 현 단계의 빈발 항목집합을 탐사할 때 탐사 대상에서 제외함으로써 빈발 항목집합의 탐사를 효율적으로 하자는 것이다. 즉, 후보 항목집합을 최소화함으로써 해쉬 트리의 탐색 시간을 줄이자는 것이다. 이것은 전 단계의 빈발 항목집합은 반드시 현 단계의 빈발 항목집합에 포함되기 때문에 당연히 성립한다. [14]에서는 성능 향상을 위하여 [13]에서 제시한 복합 해쉬 트리 기법도 적용하였다. 다음은 [14]에서 제시한 *DEC'*(본 논문에서는 표시의 편리함을 위하여 이 알고리즘을 *ODEC*라 표기한다) 알고리즘에 대한 간략한 서술이다.

알고리즘 *ODEC*(입력 : $\{D, s_n, H_2, L_o\}$, 출력 : $\{L_n\}$)

스텝1. $C_{n(1)} \leftarrow I - L_{o(1)}$에 대하여 D를 스캔하여 $L'_{n(1)}$를 구하여 $L_{n(1)} \leftarrow L'_{n(1)} \cup L_{o(1)}$로 하고 $k=2$로 한다.

스텝2. $L_{n(k-1)}$를 이용하여 $C_{n(k)}$를 구한다. 만약 $C_{n(k)}=\phi$이면 중단한다. 또한 $k=2$이면 $c \in C_{n(2)}$인 c에 대하여 $H_2(c) < |D| \times s_n$인 c는 $C_{n(2)}$에서 제거한다.

스텝3. $C_{n(k)}$를 이용하여 $C_{n(k+1)}$, $C_{n(k+2)}$, …, $C_{n(k+v)}$를 구한다.

스텝4. $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$를 계산한다. 여기서 $C'_{n(k+i)} = C_{n(k+i)} - L_{o(k+i)}$이고 $0 \le i \le v$이다.

스텝5. D를 스캔하여 $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$에 모든 후보 항목집합의 발생 수를 계산한다.

스텝6. $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$에서 발생 수가 $|D| \times s_n$ 이상인 후보 항목집합을 $L'_{n(k)}$, $L'_{n(k+1)}$, …, $L'_{n(k+v)}$으로 한다.

스텝7. $0 \le i \le v$인 모든 i에 대하여 $L_{n(k+i)} \leftarrow L'_{n(k+i)} \cup L_{o(k+i)}$로 한다.

스텝8. $k = k+v+1$로 하여 스텝 2를 반복한다.

알고리즘의 입력으로 D는 데이터베이스이며, s_n은 n단계에서 주어지는 최소 지지도이며, H_2는 데이터베이스 D에서 생성되는 모든 2-항목에 대한 해쉬 테이블이며, L_o는 o단계에서 탐사된 빈발 항목집합이다. 알고리즘의 출력으로는 L_n는 새로운 빈발 항목집합이다. 스텝 2에서 $C_{n(k)}$의 생성은 [2]에 제안한 *Apriori_gen*을 사용한다. 스텝 3은 크기가 다른 다수의 후보 항목집합을 생성하는 과정이다. 여기에 적용된 방법도 *Apriori_gen*이며 단지 스텝 2와의 차이점은 입력되는 항목집합이 빈발 항목집합($L_{n(k-1)}$)이 아니라 후보 항목집합($C_{n(k+i)}$)이라는 것이다. 즉, $C_{n(k)}$를 이용하여 $C_{n(k+1)}$을 생성하고, $C_{n(k+1)}$를 이용하여 $C_{n(k+2)}$를 생성하며, 마지막으로 $C_{n(k+v-1)}$를 이용하여 $C_{n(k+v)}$를 생성한다. [13]에서는 v의 값이 가변적임을 지적하였고 좋은 성능을 얻기 위한 v값을 실험적으로 제안하였다. 스텝 5에서는 D를 스캔하기 전에 $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$에 대한 하나의 복합 해쉬 트리를 구성한다.

<표 3>은 <표 1>의 예제 데이터베이스에 대하여 각각

〈표 3〉 DHP에서의 s_o = 50%로 수행결과

k	$C_{o(k)}$	$L_{o(k)}$
1	*A, B, C, D, E*	*A, B, C, E*
2	*AB, AC, AE, BC, BE, CE*	*AC, BC, BE, CE*
3	*BCE*	*BCE*

〈표 4〉 ODEC에서 s_n = 33%로 수행결과

k	$C_{n(k)}$	$C'_{n(k)}, \cdots, C'_{n(k+v)}$	$L_{n(k)}$
1	*A, B, C, D, E*	*D*	*A, B, C, E*
2	*AB, AC, AE, BC, BE, CE*	*AB, AE, ACE*	*AC, AE, BC, BE, CE*
3	*ACE, BCE*		*ACE, BCE*

최소 지지도 50%로 수행한 결과이고, <표 4>는 s_n = 33%에 대하여 <표 3>의 결과를 이용하여 *ODEC*를 수행한 결과이다. <표 4>에서 볼 수 있듯이 *ODEC*는 후보 항목집합도 적을 뿐만 아니라 복합 해쉬 트리를 사용함으로써 한번의 데이터베이스 스캔을 줄일 수 있다.

3. 대화형 환경에서 효율적인 연관 규칙 탐사 알고리즘

[14]에서 제안한 대화형 환경에서 연관 규칙 탐사 알고리즘 *ODEC*는 전 단계에서 찾아진 빈발 항목집합에 대해서만 현 단계의 후보 항목집합에서 제외함으로써 후보 항목집합을 최소화하여 성능 개선을 가져왔다. 본 논문에서는 전 단계에서 계산된 빈발 항목집합뿐만 아니라, 후보 항목집합에 대해서도 현 단계에서 후보 항목집합에서 이들을 제외함으로써 *ODEC*를 개선하는 알고리즘을 제안한다. 예를 들어 <표 4>의 $C_{n(2)}' = \{AB, AE\}$에서 AB 항목집합을 고려하자. 이 항목집합은 $C_{o(2)}$에서 이미 존재하였으므로 실제로 o 단계에서 발생 수가 계산되었다. 만약 n단계에서 $C_{n(2)}'$를 생성할 때 o단계에서 계산된 AB 항목집합의 발생 수를 알 수 있다면 AB 항목집합을 $C_{n(2)}'$에서 미리 제거할 수 있을 것이다. 본 논문에서는 이렇게 $C_{n(k)}'$에는 존재하나 $L_{n(k)}$에 선택되지 못할 항목집합은 $C_{n(k)}'$에서 미리 제거함으로써 후보 항목집합의 수를 적게 하고, 그 결과 해쉬 트리 탐색 비용을 줄이는 알고리즘을 제안한다.

3.1 제안하는 연관 규칙 탐사 알고리즘

제안하는 연관 규칙 탐사 알고리즘은 o단계에서 발생 수가 계산되었던 모든 후보 항목집합 중 빈발 항목집합으로 선택되지 못한 항목집합을 n단계까지 유지한 후, 이것을 이용하여 C_n을 최소화하는 것이다. n단계에서 k-후보 항목 생성된 후보 항목집합이 o단계에서 생성되어 유지된 k-후보 항목집합($C_{o(k)}$)에 존재하는 경우, 이를 $C_{n(k)}$에서 제거한다. 제거된 후보 항목집합의 발생 수가 만약 $|D| \times s_n$보다 크거나 같으면 그 발생 수를 다시 계산할 필요 없이 이 후보 항목집합을 $L_{n(k)}$에 포함한다. 따라서 o단계에서 계산된 후보 항목집합은 n단계에서 빈발 항목집합에 포함되는 경우와 후보 항목집합에서 제외되는 두 가지 경우 중에 하나이므로 절대로 C_n에 포함될 수 없다. 그러므로 C_n을 이용한 해쉬 트리의 탐색 비용이 줄어들게 된다. 다음은 제안하는 알고리즘이다.

알고리즘 $NDEC_1$**(입력 :** $\{D, s_n, H_2, CD_o, L_o\}$**,**
출력 : $\{CD_n, L_n\}$**)**

스텝1. $C'_{n(1)} \leftarrow I - L_{o(1)}$에 대하여 D를 스캔하여 $L'_{n(1)}$를 구하여 $L_{n(1)} \leftarrow L'_{n(1)} \cup L_{o(1)}$로 하고 $k = 2$로 한다.

스텝2. $L_{n(k-1)}$를 이용하여 $C_{n(k)}$를 구한다. 만약 $C_{n(k)} = \phi$이면 중단한다. 또한 $k = 2$이면 $c \in C_{2(n)}$인 c에 대하여 $H_2(c) < |D| \times s_n$인 c는 $C_{n(2)}$에서 제거한다.

스텝3. $C_{n(k)}$를 이용하여 $C_{n(k+1)}$, $C_{n(k+2)}$, …, $C_{n(k+v)}$를 구한다.

스텝4. $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$를 계산한다. $c \in CD_{o(k+i)}$인 모든 c에 대해서 c가 $\sup(c) >= |D| \times s_n$인 경우에는 $L_{o(k+i)} = L_{o(k+i)} \cup c$를 하고, 그렇지 않은 경우 $CD_{n(k+i)} = CD_{n(k+i)} \cup c$를 한다. 최종적으로 $C_{n(k+i)}$으로부터 $C'_{n(k+i)} = C_{n(k+i)} - L_{o(k+i)} - CD_{n(k+1)}$를 계산한다. 여기서 $0 \le i \le v$이다.

스텝5. D를 스캔하여 $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$에 모든 후보 항목집합의 발생 수를 계산한다.

스텝6. $C'_{n(k)}$, $C'_{n(k+1)}$, …, $C'_{n(k+v)}$로부터 $L'_{n(k)}$, $L'_{n(k+1)}$, …, $L'_{n(k+v)}$를 계산한다. $c \in C'_{n(k+i)}$인 모든 c에 대해서 c가 $\sup(c) >= |D| \times s_n$인 경우에는 $L'_{n(k+i)} = L'_{n(k+i)} \cup c$를 하고, 그렇지 않은 경우 $CD_{n(k+i)} = CD_{n(k+i)} \cup c$를 한다. 여기서 $0 \le i \le v$이다.

스텝7. $0 \le i \le v$인 모든 i에 대하여 $L_{n(k+i)} \leftarrow L'_{n(k+i)} \cup L_{o(k+i)}$로 한다.

스텝8. $k = k + v + 1$로 하여 스텝 2를 반복한다.

알고리즘의 입력으로 D는 데이터베이스이며, s_n은 n단계에서 주어지는 최소 지지도이며, H_2는 데이터베이스 D에서 생성되는 모든 2-항목에 대한 해쉬 테이블이며, CD_o는 o단계에서 후보 항목집합 중에서 빈발 항목집합으로 되지 못한 항목집합이다. L_o는 o단계에서 탐사된 빈발 항목집합이다. 알고리즘의 출력으로는 L_n는 새로운 빈발 항목집합이며, CD_n는 n단계에서 후보 항목집합 중 빈발 항목집합으로 되지 못한 항목집합이다. 알고리즘은 간단하고 명확하다. 스텝 4가 CD_o를 이용하여 후보 항목집합을 최소화하는 과정이다. CD_n은 초기에 공집합임을 가정한다. 스텝 6에서는 빈발 항목집합을 선정할뿐만 아니라 다음 단계의 연관 규칙 탐사를 위하여 새로운 후보 항목집합, CD_n을 구성하는 과정을 포함하고 있다.

앞의 예제 데이터베이스를 알고리즘 $NDEC_1$에 적용하여 보자. 첫 단계에서 s_o = 50%로 빈발 항목을 탐색한다. 그 결과 $CD_{o(2)} = \{AB, AC\}$가 된다. 이때 $\sup(AB) = 1$, $\sup(AE) = 2$의 정보도 CD_o에 포함되어 유지된다. 다음 단계에서 s_n

=33%로 빈발 항목집합을 탐색한다고 하자. 이 경우 $C_{n(2)}$ = {AB, AE}, $C_{n(3)}$ = {ACE}이 생성되나, 스텝4를 수행하면 $C'_{n(2)}$, $C'_{n(3)}$가 <표 5>와 같이 된다. 또한 스텝4와 스텝6의 결과로써 CD_n = {AB}가 된다.

〈표 5〉 $NDEC_1$에서 s_n = 33%로 수행결과

k	$C_{n(k)}$	$C'_{n(k)}, \cdots, C'_{n(k+v)}$	$L_{n(k)}$
1	A, B, C, D, E	D	A, B, C, E
2	AB, AC, AE, BC, BE, CE	ACE	AC, AE, BC, BE, CE
3	ACE, BCE		ACE, BCE

$NDEC_1$은 명확하고 간단하다. 또한 성능 비교에서 보이겠지만 C'_n의 크기를 크게 감소시키기 때문에 성능 향상 효과도 매우 크다. 그러나 이 알고리즘의 문제점은 CD_o의 크기가 경우에 따라서는 너무 크다는 것이다. <표 6>은 100,000개의 트랜잭션을 포함하고, 트랜잭션의 평균 항목 수가 20인 데이터베이스(<표 8>에서 T20I2D100)를 대상으로 연관 규칙을 탐사할 때 각 단계에서 CD_o를 유지하기 위해서 필요로 하는 메모리의 량을 보이고 있다. 최소 지지도가 작아지면서 필요로 하는 메모리 량은 급격히 증가하고 있음을 알 수 있다. 이러한 메모리 량은 구현 기법에 따라 다소의 차이는 있을 수 있겠으나 경우에 따라서는 시스템의 성능 저하를 일으킬 수도 있는 량이다.

〈표 6〉 $NDEC_1$에서 o단계에서 생성된 CD_o을 저장하는데 필요한 메모리 량

s_o	1.0%	0.9%	0.8%	0.7%	0.6%	0.5%	0.4%	0.3%
메모리량	120KB	200KB	367KB	678KB	1,430KB	3,261KB	8,292KB	22,054KB

3.2 메모리 사용을 최소화하는 연관 규칙 탐사 알고리즘

본 절에서 새롭게 제안하는 알고리즘은 $NDEC_1$ 알고리즘보다 성능 관점에서는 못하나 $NDEC_1$ 알고리즘의 문제점인 과다한 메모리의 사용을 해결하는 알고리즘이다. 제안하는 방법은 o단계에서 생성된 후보 항목집합을 저장하지 않고 단지 이들에 대한 해쉬 테이블을 유지한다. 즉, 후보 항목집합의 전체 내용 대신 후보 항목집합의 발생 수를 해쉬 테이블에 저장함으로써 메모리의 사용을 작게하는 것이다. 이러한 해쉬 테이블을 이용하는 방법은 n단계에서 k-후보 항목집합을 생성할 때 먼저 생성된 후보 항목집합에 대하여 해쉬 테이블의 값을 조사하고, 이 값이 $|D|\times s_n$을 만족하지 못한다면 $C_{n(k)}$에서 제거하는 것이다. 이 방법을 구현하기 위해서는 n단계에서 후보 항목집합을 생성할 때 생성된 후보 항목집합이 o단계에서 이미 생성된 후보 항목집합인지 아니면 n단계에서 처음으로 생성된 후보 항목집합인지를 판단할 수 있어야 한다. 그래서 만약 생성된 후보 항목집합이 o단계에서 이미 생성된 후보 항목집합이라면 해쉬 테이블을 조사하여 이 값이 $|D|\times s_n$보다 작다면 후보 항목집합에서 제거한다. 이 문제는 $NDEC_1$처럼 후보 항목집합을 직접 유지하는 경우에는 발생하지 않고, 이들에 대한 해쉬 테이블을 유지하기 경우에만 발생하는 문제이다. n단계에서 생성된 후보 항목집합, C_n중에서 o단계에서 이미 생성되었던 후보 항목집합을 C_{on}이라 하고, 그렇지 않은 후보 항목집합을 C_{nn}이라 할 때 다음 두 개의 정리는 $c\in C_{nn}$인 c의 특성을 설명하여 준다.

정리1 : $c\in C_{nn(2)}$인 c가 {XY}로 구성될 때 $\{X\}\in L_{n(1)} - L_{o(1)}$ 또는 $\{Y\}\in L_{n(1)} - L_{o(1)}$를 만족하든가 아니면 $H_2(c) < s_o|D|$를 만족한다. 여기서 $H_2(c)$는 c에 대한 H_2의 해쉬 테이블 값이다.

증명 : $c\in C_{n(2)}$인 c가 {XY}로 구성될 때 X, Y의 조건에 따라 $c_3(X\notin L_{o(1)}, Y\notin L_{o(1)})$, $c_2(X\in L_{o(1)}, Y\notin L_{o(1)})$ 및 $c_3(X\notin L_{o(1)}, Y\in L_{o(1)})$, 3가지 경우로 나누어진다. c_1, c_2의 경우는 명백히 c_1, $c_2\notin C_{o(2)}$이다. 반면 c_3의 경우는 $H_2(c_3)\geq s_o|D|$인 경우에만 $c_3\in C_{o(2)}$가 성립한다. 따라서 c_1, c_2, $c_3\in C_{nn(2)}$이면 c_1, c_2의 경우 이든가, 아니면 c_3의 경우에는 $H_2(c_3) < |D|\times s_o$ 조건을 만족하여야 한다.

정리2 : $c\in C_{nn(k)}$인 c의 k개의 $(k-1)$-부분 항목집합을 c_1, c_2, …, c_k라 하자. 이때 $c_i\in C_{nn(k-1)}$인 c_i가 반드시 하나 이상 존재한다. 여기서 $1\leq i\leq k$이다.

증명 : c_1, c_2, …, $c_k\in C_{on(k-1)}$가 성립한다고 하자. 이것은 *Apriori_gen*에 의하여 o단계에서 이미 후보 항목집합으로 생성되었다는 것을 의미하며, 따라서 $c\in C_{on(k)}$임을 의미한다. 그러므로 c_1, c_2, …, c_k 중 최소한 하나 이상은 $C_{on(k-1)}$에 속하지 않아야 한다.

정리1은 임의의 $c\in C_{n(2)}$인 c가 $c\in C_{nn(2)}$인지 아니면 $c\in C_{on(2)}$인지를 판단하는데 사용한다. $L_{o(1)}$에 대하여 비트맵을 사용한다면 임의의 $c\in C_{n(2)}$인 c = {XY}가 $c\in C_{on(2)}$임 인식하는 비용은 X, Y에 대하여 각각 한번의 비트맵 참조와 $H_2(c)$ 참조 비용의 합이다. 따라서 이것을 계산하는 비용은 $O(1)$이다. 정리2는 *Apriori_gen*을 이용하여 $C_{n(k)}$를 생성할 때 k개의 $(k-1)$-후보 항목집합을 참조함으로써 생성되는 후보 항목집합이 $C_{on(k)}$인지 아니면 $C_{nn(k)}$인지를 알수 있다. 이것의 비용은 $O(k)$이다. 다음은 상기의 정리를 기초로 메모리 사용을 최소화하는 연관 규칙 탐사 알고리즘 $NDEC_2$에 대한 설명이다.

알고리즘 $NDEC_2$**(입력 :** {D, s_n, H_2, HC, L_o},

출력 : $\{HC, L_n\}$)

스텝1. $C'_{n(1)} \leftarrow I - L_{o(1)}$에 대하여 D를 스캔하여 $L'_{n(1)}$를 구하여 $L_{n(1)} \leftarrow L'_{n(1)} \cup L_{o(1)}$로 하고 $k=2$로 한다.

스텝2. $L_{n(k-1)}$를 이용하여 $C_{n(k)}$를 구한다. 만약 $C_{n(k)} = \phi$이면 중단한다. 또한 $k=2$이면 $c \in C_{2(n)}$인 c에 대하여 $H_2(c) < |D| \times s_n$인 c는 $C_{n(2)}$에서 제거한다.

스텝3. $C_{n(k)}$를 이용하여 $C_{n(k+1)}, C_{n(k+2)}, \cdots, C_{n(k+v)}$를 구한다.

스텝4. $C'_{n(k)}, C'_{n(k+1)}, \cdots, C'_{n(k+v)}$를 계산한다. $c \in C_{on(k+i)}$인 모든 c에 대해서 c가 $HC_{(k+i)}(c) < |D| \times s_n$인 경우에는 $C_{n(k+i)}$에서 c를 제거하고, $C_{n(k+i)}$으로부터 $C'_{n(k+i)} = C_{n(k+i)} - L_{o(k+i)}$를 계산한다. 여기서 $0 \le i \le v$이다.

스텝5. D를 스캔하여 $C'_{n(k)}, C'_{n(k+1)}, \cdots, C'_{n(k+v)}$에 모든 후보 항목집합의 발생 수를 계산한다.

스텝6. $C'_{n(k)}, C'_{n(k+1)}, \cdots, C'_{n(k+v)}$에서 발생 수가 $|D| \times s_n$ 이상인 후보 항목집합을 $L'_{n(k)}, L'_{n(k+1)}, \cdots, L'_{n(k+v)}$으로 한다. $c \in C_{on(k+i)}$이고 $c \in L'_{n(k+i)}$인 모든 c에 대해서는 $HC_{(k+i)}(c)$에서 발생 수를 감해주고, $c \in C_{nn(k+i)}$이고 $c \notin L'_{n(k+i)}$인 모든 c에 대해서는 $HC_{(k+i)}(c)$에 그 발생 수를 더함으로써 HC를 갱신한다.

스텝7. $0 \le i \le v$인 모든 i에 대하여 $L_{n(k+i)} \leftarrow L'_{n(k+i)} \cup L_{o(k+i)}$로 한다.

스텝8. $k = k+v+1$로 하여 스텝 2를 반복한다.

알고리즘의 입력과 출력은 HC만 제외하고는 $NDEC_1$과 동일하다. 입력에서 HC는 o단계에서 후보 항목집합 중에서 빈발 항목집합으로 되지 못한 항목집합의 발생 수를 저장하고 있는 해쉬 테이블이고 출력에서 HC는 n단계에서 후보 항목집합 중 빈발 항목집합으로 되지 못한 항목집합의 발생 수를 저장하고 있는 해쉬 테이블이다. 이것은 다음 단계에서 입력으로 사용된다. 스텝 4가 후보 항목집합을 최소화하는 과정이다. 먼저 o단계에서 생성된 후보 항목집합에 대해서 HC를 조사하여 발생 수가 $|D| \times s_n$보다 작은 경우 후보 항목집합에서 제거하고, 또한 이미 o단계에서 빈발 항목집합으로 계산된 후보 항목집합도 제거한다. 이 과정에서 $NDEC_1$ 알고리즘과의 차이점은 $HC_{(k+i)}(c) \ge |D| \times s_n$을 만족한다고 하여도 $L_{o(k+i)} = L_{o(k+i)} \cup c$를 할수 없다는 것이다. 이것의 이유는 $\sup(c) \le HC_{(k+i)}(c)$이기 때문이다. 즉, 해싱시 충돌에 의하여 $HC_{(k+i)}(c) \ge |D| \times s_n$을 만족할지라도 $\sup(c) \ge |D| \times s_n$을 만족 못하는 경우가 있을 수 있기 때문이다. 스텝 6에서는 HC의 재사용을 위하여 HC의 값을 재계산하는 과정이다. $c \in C_{on(k+i)}$이고 $c \in L'_{n(k+i)}$인 c의 의미는 c가 o단계에서는 후보 항목집합으로는 선택되었으나, 빈발 항목집합으로 선택되지 못한 경우이다. 그러나 n단계에서는 빈발 항목집합으로 선택되었기 때문에 o단계에서 HC에 더해진 그 발생 수를 감하여야 한다. 반면 $c \in C_{nn(k+i)}$이고 $c \notin L'_{n(k+i)}$인 c의 의미는 c가 n단계에서 생성된 새로운 후보 항목집합이고 빈발 항목집합으로 선택되지 못했기 때문에 다음 단계에서 HC를 재사용하기 위하여 이 항목집합의 발생 수를 HC에 저장하여야 한다.

예제 데이터베이스를 $NDEC_2$에 적용하여 보자. 첫 단계에서 $s_o = 50\%$로 빈발 항목을 탐색한다. 이 단계에서 첫 HC_2를 생성하여야 한다. HC_2에 해싱되어야 하는 후보 항목집합이 $C_{o(2)} - L_{o(2)} = \{AB, AE\}$이다. $HC_2(x)$가 항목집합 x에 해당하는 HC_2의 해쉬 테이블 값이라 하자. 이 경우 해싱시 충돌이 없다는 가정을 하면 $HC_2(AB) = 1$, $HC_2(AE) = 2$로 설정하면 된다. 다음 단계에서 $s_n = 33\%$로 빈발 항목집합을 탐색한다고 하자. 이 경우 <표 4>에서와 같은 $C'_{n(k)}, C'_{n(k+1)}, \cdots, C'_{n(k+v)}$가 생성되나, $C'_{n(2)}$에서 $HC_2(AB) < |D| \times s_n$이므로 AB는 후보 항목집합에서 제외된다. 결과적으로 후보 항목집합은 <표 7>과 같이 된다. <표 7>은 $NDEC_2$ 알고리즘에서 후보 항목집합과 빈발 항목집합을 보이고 있다.

〈표 7〉 $NDEC_2$에서 $s_n = 33\%$로 수행결과

k	$C_{n(k)}$	$C'_{n(k)}, \cdots, C'_{n(k+v)}$	$L_{n(k)}$
1	A, B, C, D, E	D	A, B, C, E
2	AB, AC, AE, BC, BE, CE	AE, ACE	AC, AE, BC, BE, CE
3	ACE, BCE		ACE, BCE

또한 단계 6에서 생성되는 HC_2의 값은 $HC_2(AE) = 2$로 재설정하면 된다. 왜냐하면 AE는 이제 빈발 항목집합에 포함되었기 때문이다.

4. 성능 비교

본 논문에서 제안하는 알고리즘의 성능 비교를 위하여 DHP, $ODEC$, $NDEC_1$ 및 $NDEC_2$ 알고리즘 모두를 C++로 구현하였다. $NDEC_2$의 구현에서 해쉬 테이블의 크기는 400 KB로 고정하였다. 구현 및 성능 비교 환경으로는 256M 주기억장치를 가진 펜티움 III 800MHZ 컴퓨터이고, 운영체제는 윈도우즈98이다. 성능은 기존의 알고리즘인 DHP에 대하여 $ODEC$와 제안하는 알고리즘 $NDEC_1$, $NDEC_2$를 상대적으로 비교하였다.

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 베스킷 데이터의 특징을 잘 포함하고 있는 것으로 알려져 있다[2]. <표 8>은 실험 데이터베이스를 생성할 때 사용하는 매개 변수를 나타내며, <표 9>은 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 8〉 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

〈표 9〉 실험 데이터베이스 및 매개 변수 값

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

<표 10>은 <표 9>의 6개의 데이터베이스에 대하여 *DHP* 방법과 $ODEC$, $NDEC_1$, $NDEC_2$ 사이의 성능 비교의 결과를 나타내고 있다. <표 10>에서 *DHP* 알고리즘의 값은 실제 실험적 환경에서 수행 시간을 측정한 값으로 단위는 $\frac{1}{1000}$초(msec)이다. $ODEC$, $NDEC_1$, $NDEC_2$ 수행 값은 다음과 같은 공식에 의하여 *DHP*에 대한 상대적 성능 개선 효과를 백분율로 나타낸다.

$$G_{\{o|n1|n2\}} = \frac{DHP\text{의 실행시간} - \{ODEC|NDEC_1|NDEC_2\}\text{의 실행시간}}{DHP\text{의 실행시간}} \times 100(\%)$$

<표 10>에서 볼 수 있듯이 *DHP*에 비하여 $ODEC$, $NDEC_1$, $NDEC_2$의 성능이 월등히 우수함을 알 수 있다. 여기서 밑줄은 동일 환경에서 가장 우수한 알고리즘에 대한 값을 표시한다. 제안된 알고리즘 중에서 가장 성능이 우수한 알고리즘은 $NDEC_1$이다. 이것은 $NDEC_1$에서는 빈발 항목집합으로 선택되지 못한 모든 후보 항목집합을 유지하여 다음 단계에서 활용함으로써 해쉬 트리의 탐색 비용을 최소화하기 때문이다. $NDEC_2$의 성능은 대부분의 경우에 대하여 $NDEC_1$의 성능에 미치지 못한다. 이것은 이들의 알고리즘을 고려할 때 충분히 예견할 수 있는 일이다. 즉, $NDEC_1$에서는 $c \in CD_o$이고 $\sup(c) \geq |D| \times s_n$인 c들은 해쉬 트리의 탐색없이 바로 빈발 항목집합에 포함되기 때문이고, 또한 $NDEC_1$는 CD_o를 이용하여 후보 항목집합을 최소화할 수 있으나 $NDEC_2$는 해쉬 테이블에서 충돌 때문에 $NDEC_1$보다 후보 항목집합을 최소화할 수 없는 경우가 발생할 수도 있기 때문이다. 비록 $NDEC_2$의 성능이 $NDEC_1$의 성능보다는 못하나 대부분의 경우 $ODEC$의 성능보다는 우수하게 나타난다. 이러한 사실은 시스템의 자원에 따라서 $NDEC_2$ 알고리즘도 유용하게 사용될 수 있음을 의미한다. 제안된 알고리즘의 성능 향상 효과는 트랜잭션에서 평균 항목의 수가 10이하인 경우(T05I2D100, T10I2D100, T10I4D100)에는 대부분 80% 이상이나 트랜잭션에서 평균 항목의 수가 20인 경우에는 30%~70% 정도의 성능 향상 효과를 보이고 있다. <표 10>에 직접적으로 나타나지는 않았지만 $ODEC$에 대하여 $NDEC_1$, $NDEC_2$의 상대적 성능 개선 효과는 트

〈표 10〉 성능 측정 결과

$S_o \rightarrow S_n$ \ D		T05I2D100	T10I2D100	T10I4D100	T20I2D100	T20I4D100	T20I6D100
1.00% → 0.75%	*DHP*	485 *msec*	1,440 *msec*	1,225 *msec*	11,310*msec*	12,465 *msec*	11,745 *msec*
	G_o	79.38%	79.17%	82.45%	37.80%	40.03%	26.27%
	G_{n1}	79.38%	79.17%	82.45%	45.71%	48.94%	32.06%
	G_{n2}	79.38%	79.17%	82.45%	41.91%	43.68%	27.67%
0.75% → 0.60%	*DHP*	560 *msec*	1,605 *msec*	2,750 *msec*	16,068*msec*	17,590 *msec*	19,845 *msec*
	G_o	80.36%	78.82%	83.09%	29.66%	41.39%	26.38%
	G_{n1}	80.36%	80.37%	83.64%	49.68%	55.46%	46.11%
	G_{n2}	80.36%	79.44%	83.45%	42.75%	48.83%	33.28%
0.60% → 0.50%	*DHP*	565 *msec*	1,795 *msec*	2,866 *msec*	22,380*msec*	23,915 *msec*	36,270 *msec*
	G_o	79.82%	68.52%	80.60%	28.26%	39.72%	34.42%
	G_{n1}	78.94%	81.06%	84.12%	52.28%	58.06%	49.32%
	G_{n2}	78.76%	79.67%	81.51%	45.82%	51.30%	41.94%
0.50% → 0.45%	*DHP*	585 *msec*	2,725 *msec*	3,205 *msec*	27,015*msec*	29,200 *msec*	43,620 *msec*
	G_o	78.63%	86.97%	83.00%	28.63%	42.38%	53.27%
	G_{n1}	79.49%	89.36%	87.83%	62.98%	69.11%	73.88%
	G_{n2}	79.49%	88.26%	83.93%	57.04%	61.76%	65.94%

랜잭션에서 평균 항목의 수가 10이하인 경우 각각 20%, 10%이고, 20인 경우에는 각각 30%, 20% 정도임을 실험적으로 알 수 있었다.

5. 결 론

본 논문은 대화형 연관 규칙 탐사 환경에서 연관 규칙 탐사를 효율적으로 지원하는 알고리즘을 제안하였다. 제안한 알고리즘의 핵심 개념은 동일한 데이터베이스에 대하여 반복적으로 연관 규칙 탐사가 일어난다는 특징을 활용하여 새로운 연관 규칙 탐사할 때 가능한 이전의 연관 규칙 탐사에서 계산된 정보를 최대한 활용함으로써 계산량을 줄여 새로운 연관 규칙을 효율적으로 탐사하자는 것이다. 기존의 방법에서는 새로운 연관 규칙 탐사에 활용되는 데이터로 이전의 연관 규칙 탐사에서 계산된 빈발 항목집합을 사용하였다. 개선된 방법에서는 전 단계에서 계산된 빈발 항목집합 외에 후보 항목집합을 이용하여 현 단계에서 계산 량을 대폭적으로 줄이는 알고리즘을 제안하였다. 후보 항목집합을 유지하는 방법에 따라 두 가지 알고리즘 $NDEC_1$과 $NDEC_2$를 제안하였다. 제안된 알고리즘의 성능 비교를 위하여 *DHP*, *ODEC*, $NDEC_1$, $NDEC_2$ 모두를 구현하였으며, 제안된 방법인 $NDEC_1$과 $NDEC_2$가 기존의 방법인 *ODEC* 보다 10%~30%정도의 성능 개선 효과가 있음을 알 수 있었다.

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.

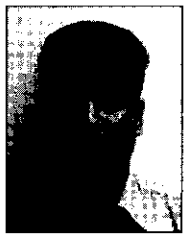
[10] R. Agrawal and et al, "Programs Generating Test Data in Data Mining," http ://www.almaden.ibm.com/cs/quest, 1997.

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

[12] 이재문, "점진적 연관 규칙 탐사에 대한 효율적인 알고리즘", 한성대학교 정보통신논문집, Vol.1, pp.79-90, 1999.

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

[14] 이재문, 박종수, "대화형 환경에서 연관 규칙 탐사 알고리즘", 한국정보처리학회 99 춘계학술발표논문집, pp.51-55, 1999.



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)

1992년~1994년 한국통신 연구개발단 선임연구원

1994년~현재 한성대학교 정보전산학부 부교수

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