Prefix-Tree를 이용한 높은 유틸리티 패턴 마이닝 기법

(High Utility Pattern Mining using a Prefix-Tree)

정 병 수 † 아메드 파한 †† 이 인 기 ††† 용 환 승 ††††
(Byeong-Soo Jeong) (Chowdhury Farhan Ahmed) (In-Gi Lee) (Hwan-Seong Yong)

요 약 유틸리티 패턴 마이닝은 데이터 항목에 대한 다른 가중치를 고려할 수 있는 장점으로 인하여 비즈니스 데이터를 분석하는 환경에서 효율적으로 이용되고 있다. 그러나 기존의 빈발 패턴(Frequent Pattern) 마이닝에서의 Apriori 규칙을 그대로 적용하기 어려운 문제점으로 인하여 패턴 마이닝의 성능이 현저하게 떨어지고 있다. 본 연구는 Prefix-tree를 이용하여 지속적으로 증가하는 비즈니스 트랜잭션 데이터베이스에 대한 유틸리티 패턴 마이닝을 효과적으로 수행하기 위한 기법을 제안한다. 제안하는 기법은 Prefix-tree의 각 항목 노드에 유틸리티 값을 저장하여 FP-Growth 알고리즘에서와 같이 트리의 상향 탐색을 통하여 높은 유틸리티 패턴을 빠르게 찾아낸다. 여러 형태의 실험을 통하여 이용할 수 있는 세가지 다른 Prefix-tree 구조들 간의 성능적 특징과 패턴 탐색의 방법들을 비교하였으며 실험 결과에 따라 제안하는 기법이 기존의 기법들에 비해 많은 성능 향상을 가져올 수 있는 것을 입증하였다

키워드 : 유틸리티 패턴 마이닝, 데이터 마이닝, 트랜잭션 빈도, 트랜잭션 유틸리티 가중치

Abstract Recently high utility pattern (HUP) mining is one of the most important research issues in data mining since it can consider the different weight values of items. However, existing mining algorithms suffer from the performance degradation because it cannot easily apply Apriori-principle for pattern mining. In this paper, we introduce new high utility pattern mining approach by using a prefix-tree as in FP-Growth algorithm. Our approach stores the weight value of each item into a node and utilizes them for pruning unnecessary patterns. We compare the performance characteristics of three different prefix-tree structures. By thorough experimentation, we also prove that our approach can give performance improvement to a degree.

Key words : High utility pattern mining, data mining, transaction frequency, transaction weighted utilization

1. 서 론

높은 유틸리티 패턴(high utility pattern)에 대한 마이닝은 데이터 항목(item)에 대한 다른 가중치를 고려할 수 있는 장점으로 인하여 비즈니스 데이터를 분석하는 환경에서 효율적으로 이용되고 있다. 그러나 기존의 빈발 패턴 마이닝에서의 Apriori 규칙을 그대로 적용하기 어려운 문제점으로 인하여 패턴 마이닝의 성능이 현저하게 떨어지고 있다. 빈발 패턴 마이닝의 초기 해결 방법인 Apriori 기법[1,2]의 후보집합 생성 및 검사 전략은 여러 번의 데이터베이스 스캔과 많은 후보집합들을 생성해야 하는 문제점들을 가지고 있었다. FP-Growth[3] 기법은 FP-트리를 이용하여 후보집합의 생성 및 검사 없이 빈발 패턴을 탐색할 수 있어 이러한 문제들을 해결하였다. 비록 빈발 패턴 마이닝[1-3]이 데이터 마이닝

† 종신회원 : 경희대학교 전자정보대학 컴퓨터공학전공 교수
jeong@khu.ac.kr
†† 비 회 원 : 경희대학교 전자정보대학 컴퓨터공학전공
farhan@khu.ac.kr
††† 학생회원 : 이화여자대학교 컴퓨터공학과
lig@ewhain.net
†††† 종신회원 : 이화여자대학교 컴퓨터공학과 교수
hsyong@ewha.ac.kr
논문접수 : 2009년 3월 20일
심사완료 : 2009년 7월 28일

정보과학회논문지: 데이타베이스 제36권 제5호(2009.10)

애플리케이션에서 중요한 역할을 하고 있지만 그것은 두 가지 제한점을 갖는다. 첫 번째는 모든 항목을 동일한 중요도(weight)로 간주한다는 것이고, 두 번째는 하나의 트랜잭션에서 각각의 항목이 이진수(0/1) 형태로 나타난다는 것이다. 그러나 우리의 실생활에서 볼 수 있는 것과 같이 시장에서의 제품들은 서로 다른 중요도나 가격을 가질 수 있고, 한 명의 고객이 한 종류의 제품을 여러 개 살 수도 있다. 따라서 비즈니스 데이터베이스에서 전통적인 빈발 패턴을 찾는 것이 비즈니스상에서 총 수익의 중요한 부분을 차지하는 가장 중요한 고객이나 제품들을 찾고자 하는 요구사항을 충족할 수는 없었다.

최근에 유틸리티 마이닝[3,4] 모델은 데이터베이스로부터 보다 실제적인 유용한 지식을 발견하기 위해 정의되었다. 우리는 유틸리티 개념으로부터 항목집합의 중요도를 측정할 수 있다. 예를 들어 서로 다른 가격의 제품들을 다른 중요도를 갖는 유틸리티 데이터집합으로 간주 할 수 있으며, 그것은 실제로 우리 실생활에서의 장바구니(market basket) 데이터를 반영한다. 유틸리티 마이닝에 의해 비즈니스 분야에서 수익, 비용, 마케팅등과 관련된 중요한 결정들이 내려질 수 있으며, 수익에 큰 기여를 하는 제품들과 고객들에 관련된 정보를 발견할 수 있다. 그 외에 생물학적 유전자 데이터베이스와 웹 클릭 스트림과 같은 분야를 고려해 보았을 때도 각 유전자 또는 웹 페이지의 중요성이 서로 다르며, 발생빈도 또한 이진수 형태로 제한적이지 않다는 것을 알 수 있다. 네트워크 트래픽, 웹 서버 로그, 센서 네트워크 데이터, 통신 통화이력 등과 같은 다른 애플리케이션분야들도 유사한 해결방법을 필요로 할 수 있다.

이러한 유틸리티 마이닝 분야에서의 기존 연구들[3-6, 7]은 대부분 고정된 데이터베이스(static database)을 고려한 것들이 대부분 이었다. 즉 데이터베이스에서 새로운 트랜잭션들이 삽입되거나 삭제될 수 있는 상황을 고려하지 않은 마이닝 알고리즘들이었다. 본 연구에서는 점증적인(incremental) 특성을 가질 수 있는 Prefix-트리 구조를 이용하여 데이터베이스가 수정되거나 유틸리티 한계값이 변경되었을 때, 이전의 데이터 구조들과 마이닝 결과들을 사용할 수 있고 불필요한 계산을 피할 수 있는 효과적인 HUP(high utility pattern) 마이닝 기법을 제안한다. 기존의 연구들[8-10]에서는 단지 빈발 패턴만을 고려하여 점증적 데이터베이스를 다루어 왔다. 그리고 점증적인 Prefix-트리 구조가 현재 이용 가능한 Giga Bytes 수준의 메모리 사이즈를 사용하여 효과적으로 처리가 가능하다는 것을 보여 주고 있다. 따라서 본 연구에서도 주 메모리에 저장되는 점증적인 Prefix-트리 구조를 이용하여 높은 유틸리티를 갖는 패턴들을 효과적으로 탐색하는 기법이 효용성이 있음을 보인다.

본 논문에서는 위에 언급한 기존 연구들의 문제점들을 해결하기 위한 방법으로 점증적 데이터베이스 환경하에서 유틸리티 패턴 마이닝을 위한 세가지 새로운 Prefix-트리 구조를 제안한다. 제안하는 기법은 후보 생성 및 검사 전략의 문제점들을 해결할 수 있는 FP-Growth 방식을 기반으로 하고 있으며 다른 Prefix-트리 구조에 따라 성능적 특징이 매우 다를 수 있음을 보이고 최적의 Prefix-트리 구조를 사용할 경우 평균적으로 기존의 유틸리티 패턴 마이닝 기법에 비하여 성능적으로 우수함을 여러 실험을 통하여 입증한다. 제안하는 기법에서의 Prefix-트리 구조는 트랙잭션 삽입에 사용되는 항목의 순서에 따라 세가지 형태로 구분한다. 첫 번째 경우는 먼저 항목의 사전적 순서(lexicographic order)를 따르는 구조이며, 이 트리 구조는 어떠한 재구성 절차 없이 점증적으로 트랜잭션 데이터를 삽입할 수 있다. 그러나 각 트랜잭션들의 Prefix의 공유 정도가 최적이 될 수 없으므로 Prefix-트리 크기가 상대적으로 커질 수 있는 단점이 있다. 두 번째 경우는 삽입되는 항목의 순서를 항목의 빈도 값에 따라 저장하는 Prefix-트리 구조로 이 경우는 각 트랜잭션들의 Prefix의 공유 정도가 최적이 되도록 하여 Prefix-트리 구조를 조밀하게 구성할 수 있다. 그러나 이 트리는 높은 빈도 값을 갖는 항목들이 높은 유틸리티 값을 갖지 못하는 경우 성능이 떨어질 수 있는 문제점이 있다. 만일 낮은 유틸리티 항목들이 트리의 위쪽에 위치할 경우 마이닝 시간이 길어질 수 있다. 세 번째 트리 구조는 삽입되는 항목의 순서를 항목의 TWU(transaction weighted utilization)값에 따라 처리함으로써 트리의 가지에서 높은 유틸리티 항목들의 후보가 낮은 유틸리티 항목들 앞에 항상 먼저 나타날 수 있도록 설계되었다. 본 논문의 실험 결과에 따르면 세 번째 트리 구조가 가장 적은 수행시간을 필요로 하는 것으로 나타났다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하고 3장에서의 문제정의를 통해 4장에서 우리가 제안하는 트리 구조를 설명 한다. 5장에서는 실험을 통해 제안한 방법의 우수성을 보이고 6장에서 결론을 맺는다.

2. 관련 연구

가중치를 부여한 항목 집합(itemset 또는 pattern) 마이닝에 관한 연구들[11,12]은 그 동안 많은 사람들에 의해 연구되어 왔다. 그 중 항목집합 공유 접근법[7]은 각 트랜잭션에서 항목의 다중 빈도를 고려한 연구이며, 유틸리티 항목집합들의 마이닝 기법에 관한 [13]의 연구에

서는 다른 환자들에 대해 동일한 의학적 치료를 가정하여, 다른 트랜잭션들이 그들의 효능에 따라 치료 효과가 다양해 질 수 있다는 것을 보여주고 있다. MEU(mining with expected utility)라고 불려지는 이론적 모델과 정의에 관련된 연구를 [4]에서 볼 수 있다. 이 연구에서는 항목집합이 후보집합으로 선정될 수 있는지 여부를 휴리스틱 접근법을 사용하여 판단한다. 이 방법은 일반적으로 후보 집합을 모든 항목들의 조합으로 구성하는 시작 단계에서 성능적 문제를 야기한다. 따라서, 서로 다른 항목들의 숫자가 많아지고 유틸리티 한계 값이 작아질 경우에는 현실적으로 사용할 수 없게 된다. 그 후 동일한 저자에 의해 높은 유틸리티 값을 갖는 패턴들을 계산하기 위한 두 개의 새로운 알고리즘 UMining과 UMining_H[4]을 제안되었다. UMining에서는 유틸리티 최대값을 기반으로 하는 가지치기 전략을 사용하고, UMining_H는 휴리스틱 접근법을 기반으로 가지치기 전략을 사용하고 있다. 그러나 이들 방법은 일부 높은 유틸리티 값을 가진 항목 집합들을 잘못 가지치기하는 경우가 발생할 수 있으며, 또한 Apriori-규칙을 만족하지 못하여 후보 생성 및 검사 과정에서 많은 불필요한 패턴들을 생성하는 문제점을 갖고 있다.

Two-Phase[14,15] 알고리즘은 Apriori-규칙을 사용해서 높은 유틸리티 값을 갖는 항목집합들을 효율적으로 찾아내기 위한 방법으로 [4]의 유틸리티 모델을 기반으로 제안되었다. Two-Phase[5,6] 알고리즘에서는 "TWU (transaction weight utilization)" 개념을 정의하고 이를 이용하여 Apriori-규칙의 속성을 유틸리티 마이닝에서도 활용할 수 있음을 증명하였다. 첫 번째 데이터베이스 스캔에서 한 개의 항목으로 구성된 모든 항목집합의 TWU 값을 계산하고 2개 항목의 TWU 항목집합을 생성하는데 기반으로 삼는다. 두 번째 데이터베이스 스캔에서는 두 개 항목의 모든 TWU 항목집합을 찾아서 세 개 항목의 TWU 항목집합을 위한 후보들을 생성하는데 사용한다. 마지막 스캔에서는 높은 TWU 항목집합들로부터 가장 높은 유틸리티 항목집합들을 찾아낸다. 이 알고리즘은 Apriori-규칙의 속성을 활용하는 방안을 제시하였으나 다른 연구에서와 같이 후보 생성 및 검사 전략이 지니는 불필요한 항목집합을 생성하게 되는 문제점을 갖는다.

그 밖의 연구들로는 전통적인 빈발 패턴(frequent pattern) 마이닝 분야에서 점증적인 마이닝 기법과 관련된 연구들[7,16-19,20]이 발표되고 있다. AFPIM[16]은 FP-트리 구조를 경로 조정 기법(path adjusting method)을 사용해서 재구성하였고, Cantree[18]는 정규 순서로 트랜잭션들을 처리하고 데이터베이스가 증가하거나 감소하는 동안 트리 안에서 데이터베이스의 전체 정보를 저장할 수 있도록 하여, 한번의 트리 생성으로 반복적인 마이닝 작업을 쉽게 수행할 수 있도록 하였다. CP-tree [20]는 항목의 빈도수 값에 따라 점증적 트리를 재구성함으로써 기존 Cantree의 마이닝 속도를 향상시키고 있다. 그러나 현재까지의 유틸리티 패턴 마이닝 기법들은 새롭게 많은 트랜잭션들이 추가되고, 삭제되는 점증적인 데이터베이스를 위한 마이닝을 수행하기에는 부적절하고 또한 후보 패턴의 생성 과정에서 많은 불필요한 연산을 수행하여 마이닝의 속도를 떨어뜨리고 있다. 본 논문에서는 Prefix-트리 구조를 기반으로 점증적인 유틸리티 패턴 마이닝이 가능한 새로운 방법을 제안한다. 제안하는 기법은 Prefix-트리 구조의 각 노드에 TWU및 빈도수 값을 저장하여 FP-Growth방법처럼 후보 집합의 생성없이 Apriori-규칙을 적용할 수 있도록 하였으며 다양한 실험을 통하여 세가지 다른 형태의 Prefix-트리 구조에 따라 성능적 특징이 어떻게 달라질 수 있는 지를 분석하였다.

3. 문제 정의

오늘날 데이터의 추가, 삭제, 수정 작업이 빈발하게 발생하는 점증적인 데이터베이스의 필요성을 이해하기 위해 우리는 실생활의 예제를 다음과 같이 설명하고 논문에서 활용하고자 한다. 고객A는 펜 3자루, 연필 4자루, 지우개 1개를 샀고, 고객B는 컴퓨터 마우스 한 개를 샀다. 얼마 후에 고객C는 빵2개와 우유1개를 사고, A는 다시 연필 2자루를 환불하였고, B는 마우스를 환불하였다. 이와 같이 실제 시장에서는 새로운 트랜잭션들이 추가 될 수도 있고, 오래된 트랜잭션들이 빈번하게 수정되거나 삭제될 수도 있다. 결과적으로 우리는 추가, 삭제, 수정 트랜잭션들을 실제 데이터 집합에서 고려해 주어야 한다. 표 1은 트랜잭션들의 새로운 그룹이 어떻게 점증적인 데이터베이스에서 추가될 수 있는지 보여준다. 우리는 본 논문에서 제안하는 Prefix-트리를 설명하기 위해 위의 예제를 활용하고자 한다. 표 1(a)는 각각의 트랜잭션을 나타내며 A B C …등은 항목(item)으로 빵 우유 등과 같은 상품명을 의미하게 되고 트랜잭션 내에서의 각 숫자는 해당 항목의 구매 횟수를 표시한다. 즉 T1은 A와 B를 각각 2개씩 구매한 것을 의미한다. 표 1(b)은 각 항목들의 중요도(혹은 가격)를 나타내는 테이블이다. 표 1(a)에서 Trans. Utility는 표 1(b)의 유틸리티 값과 각 트랜잭션에서의 구매 횟수를 곱하여 산정한 각 트랜잭션의 유틸리티 총합이다. 즉 T1의 경우 A, B의 구매 횟수와 각각의 유틸리티 값을 이용하여 (2 * 2 + 2 * 15 = 34) 계산한 값이다.

다음은 이전 연구들[3-6]에서 정의된 것과 유사한 방법으로 유틸리티 패턴 마이닝의 문제를 정의한다. 각각

표 1 트랜잭션 데이터베이스와 유틸리티 데이터 예제

	TID \ ITEM	A	B	C	D	E	Trans. Utility
Original DB	T_1	2	2	0	0	0	34
	T_2	3	0	12	4	2	88
	T_3	0	0	15	0	3	66
	T_4	4	0	0	0	0	8
db_1 +	T_5	0	10	0	8	9	277
	T_6	0	7	3	0	4	142
db_2 +	T_7	1	0	2	0	1	15
	T_8	2	0	0	1	3	33

(a) Transaction Database

ITEM	PROFIT($) (per unit)
A	2
B	15
C	3
D	8
E	7

(b) Utility Table

의 용어들은 아래와 같이 정의되며 예제 데이터 값들은 표 1의 값을 이용하여 계산하였다. 먼저 집합 I = {i_1,i_2, ···i_m}는 항목들의 집합, D는 트랜잭션 데이터베이스 {T_1,···T_N}라고 하자. 여기서 집합D에 속하는 T_i는 I의 부분 집합이다.

정의 1. 내부 트랜잭션 유틸리티 값 $l(i_p,T_q)$는 트랜잭션 T_q에서 i_p의 수량을 표현한다. 예를 들면 표 1(a)의 $l(C,T_2)$=12이다.

정의 2. 외부 유틸리티 $p(i_p)$는 항목 i_p의 단위 이윤 값이다. 예를 들어 표 1(a)의 p(C)=3이다.

정의 3. 유틸리티 $u(i_p,T_q)$는 트랜잭션 T_q에서 항목 i_p에 대한 유틸리티의 양적 계산이며 다음과 같이 정의된다.

$$U(i_p,T_q) = l(i_p,T_q) * p(i_p)$$

예를 들어 표 1에서 $u(C,T_2)$=12*3=36이다.

정의 4. 트랜잭션 T_q에서의 항목집합 X의 유틸리티 $u(X,T_q)$는 다음과 같이 정의된다.

$$u(X,T_q) = \sum_{ip \in Tq} u(ip,Tq)$$

X={i_1,i_2,.....i_k}는 k-항목집합일 때, X ⊆ T_q이고 1≤k ≤m이다. 예를 들어 표 1에서 $u(AC,T_2)$=3*2+12*3=42 이다.

정의 5. 항목집합 X의 유틸리티는 다음과 같이 정의된다.

$$u(X) = \sum_{Tq \in D} \sum_{ip \in X} u(ip,Tq)$$

예를 들어 표 1에서 u(AC)=u(AC,T2)+u(AC,T7)= 42+8=50이다.

정의 6. 트랜잭션 T_q의 트랜잭션 유틸리티는 다음과 같이 정의되며, $tu(T_q)$는 그 트랜잭션의 총 이윤을 의미한다.

$$tu(T_q) = \sum_{ip \in Tq} u(ip,Tq)$$

예를 들어 표 1에서 $tu(T_1)$=u(A,T_1)+u(B,T_1)=4+30= 34이다.

정의 7. 최소 유틸리티 한계 값 δ는 데이터베이스의 총 트랜잭션 유틸리티 값들의 백분율로 정의 된다. 표 1에서 모든 트랜잭션 유틸리티 값들의 합은 663이다. 만일 δ값이 30% 인 경우 최소 유틸리티 값은 다음과 같이 정의될 수 있다.

$$minutil = \delta \times \sum_{Tq \in D} tu(Tq)$$

예를 들어 표 1에서 *minutil* = 0.3*663=198.9이다.

정의 8. 만일 u(X) ≥ *minutil* 이라면 항목집합 X는 높은 유틸리티 항목 집합(high utility pattern)이다.

높은 유틸리티 항목 집합을 찾기 위해서는 항목집합 X 에서 u(X) ≥ *minutil* 범위의 모든 값을 발견해야 한다. 유틸리티 패턴 마이닝 분야에서 주요하게 논의되고 있는 문제는 항목 집합 유틸리티가 빈발패턴 마이닝에서 항목의 출현빈도 값을 계산할 때 이용하는 Apriori-규칙 속성을 갖지 않는다는 것이다. 예를 들어 표1에서 *minutil*=198.9일 경우 정의 5와 정의 8에 따라 u(D)는 104로 낮은 유틸리티 항목이지만 u(DE)는 202로 높은 유틸리티 항목이 된다.

정의 9. 항목집합X의 twu(transaction weighted utilization)는 X를 포함하는 모든 트랜잭션들의 유틸리티 값의 합으로 정의한다.

$$twu\ (X) = \sum_{X \subseteq Tq \in D} tu(Tq)$$

예를 들어 표 1에서 twu(AC)=tu(T_2)+tu(T_7)=88+15 =103이다. 항목집합(즉 패턴) X에 twu 값은 정의 8의 u(X)에서는 적용될 수 없었던 Apriori-규칙 속성을 twu(X) 값에서는 적용할 수 있다. 즉 twu(ACE)는 항상 twu(AC) = 103보다 항상 같거나 작게 된다.

높은 유틸리티 패턴(high utility pattern) 마이닝 문제는 트랜잭션에 나타나는 모든 항목 집합(즉 패턴)들 중에서 정해진 임계값, *minutil* 보다 높은 u(X) 값을 갖는 모든 X를 찾아내는 것으로 정의할 수 있다.

4. Prefix-트리 구조를 이용한 유틸리티 패턴 탐색

본 장에서는 Prefix-트리를 이용하여 높은 유틸리티 패턴을 효과적으로 탐색하는 방법을 설명하기로 한다. 이를 위하여 먼저 유틸리티 패턴 탐색을 위하여 사용될 수 있는 세가지 다른 형태의 Prefix-트리 구조 및 생성

방법, 그리고 패턴 탐색에 이용되는 과정을 비교 설명하고, 이어서 Prefix-트리를 이용하여 유틸리티 패턴을 탐색하는 알고리즘과 알고리즘의 성능적 특성에 대하여 살펴보기로 한다.

4.1 Prefix-트리 구성

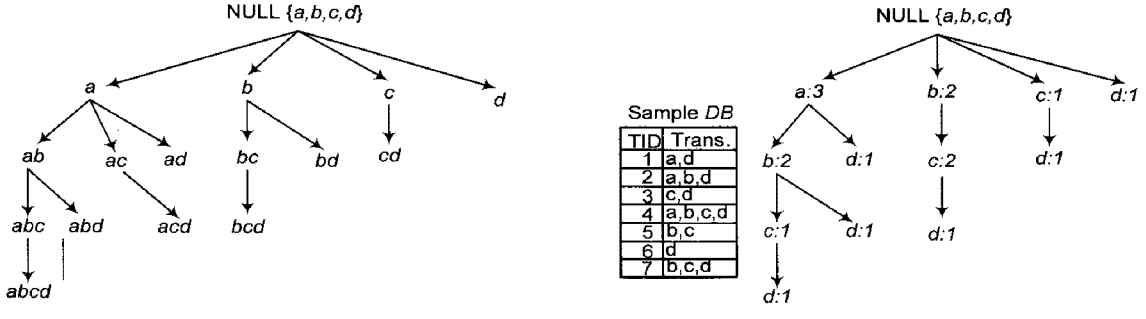
Prefix-트리는 일종의 순서-트리(ordered-tree) 형태로서 트랜잭션 데이터베이스의 내용을 압축된 형태로 나타낼 수 있도록 하는 자료 구조이다. 항목의 집합으로 이루어진 개개의 트랜잭션들은 일정한 항목들 순서에 따라 차례로 트리의 각 노드에 삽입하며 이때 count 값을 하나씩 계속 증가시킨다. 이렇게 만들어진 Prefix-트리의 각 경로(path)는 각각의 트랜잭션을 나타내게 된다. 일반적으로 각각의 트랜잭션에는 여러 개의 항목들이 공통적으로 나타날 수 있으므로 Prefix-트리의 각 경로들은 여러 트랜잭션들을 중복으로 표현할 수 있게 된다. 중복된 항목들이 많으면 많을수록 보다 조밀한 Prefix-트리의 구성이 가능해지는 것이다.

그림 1(a)는 네개의 항목 L = {a, b, c, d}에 대하여 사전적 순서로 만들어진 완전(즉 모든 패턴을 포함하는) Prefix-트리의 모습을 나타낸다. 그림 1(b)는 왼쪽의 Sample DB의 내용을 역시 항목들의 사전적 순서에 따라 구성한 Prefix-트리를 보여주고 있다. 트리의 모든 패턴 p에 대하여 p 다음에 추가되는 항목들은 p를 포함하는 p 보다 긴 패턴을 나타내게 된다. 이때 추가되는 항목들을 p의 후보 확장(candidate extensions)이라 한다. 예를 들면 그림 1(b)에서 항목 c와 d는 ab의 후보 확장이고 b는 ac의 후보 확장이 될 수 없다. 그 이유는 b는 c 보다 사전적 순서가 앞에 있기 때문이다. 만약 빈발 패턴 p에 대하여 후보 확장을 하게 되면 확장된 패턴도 빈발 패턴이 될 수 있다. 각 노드의 빈도수 값은 Prefix-트리의 루트 노드로부터 해당 노드까지의 경로로 표시된 항목 집합의 총 빈도수를 나타내게 된다. 예를 들면 그림 1(b)에서 {ab}의 빈도수는 2이고 {abc}의 빈도수는 1이 된다. Prefix-트리의 기본적인 성질의 하나로 각 노드의 빈도수 값은 자식 노드들의 빈도수 값의 합보다 항상 같거나 크게 된다.

유틸리티 패턴의 탐색을 위하여 본 논문에서 사용하는 Prefix-트리는 위의 기본 구조에 각 항목 집합의 twu값을 함께 고려하는 구조이다. 즉 트리의 각 노드에 빈도수 값과 더불어 해당 패턴에 대한 twu값도 같이 저장한다. 이때 twu값도 빈도수 값과 같이 자식 노드들의 twu 값의 합보다 항상 같거나 크게 된다. 트리를 구성하는 방법은 우선 삽입되는 항목들의 순서에 따라 달라지게 되는 데, 그 경우로는 항목들을 사전적 순서로 배열되어 Prefix-트리에 삽입하는 것, 다음에는 첫 번째 트리 구조의 단점을 보완하는 방안으로 항목들의 순서를 항목들의 출현 빈도 값에 따라 삽입하는 경우, 그리고 twu 값의 크기에 따라 삽입한 경우를 생각할 수 있다.

사전적 순서에 따른 Prefix-트리의 구성은 첫 번째 데이터베이스 스캔에서 항목들을 사전적 순서로 배열하고 트리 안에 노드로 항목들을 삽입한다. 헤더 테이블은 FP-트리[10]와 유사한 방식으로 관리된다. 그러나 FP-트리는 항목의 빈도 값만을 관리하지만 여기서는 twu (transaction weighted utilization)값과 tf(transaction frequency)값을 헤더테이블과 트리의 노드 두 곳에서 관리한다. 트리의 각 노드에 twu값과 tf값을 저장하는 이유는 FP-Growth방법에 따라 트리의 상향 순회 과정에서 높은 유틸리티 패턴이 될 수 있는 후보 패턴을 효과적으로 찾아내기 위함이다. 자세한 내용은 4.2절에서 설명하기로 한다. 그리고 트리 순회를 유용하게 하기 위해서 인접한 링크들도 FP-트리와 유사하게 관리된다. 트리의 구성과정은 다음의 예제로 설명한다.

우선 표 1의 트랜잭션 데이터베이스와 유틸리티 표를 이용하여 첫 번째 트리(사전적 순서)의 구성 과정을 설명하면, 초기에 데이터베이스는 단지 4개의 트랜잭션만을 포함하고 있다. 트리의 구성 과정은 그림 2(a)와 그림 2(e)에서 기술된다. 표 1에서 T_1의 twu값은 34이다. 그림 2(a)는 T_1이 트리에 추가 되었을 때 사전적 순서



(a) Complete prefix-tree for L = {a,b,c,d}　　(b) Prefix-tree for the sample *DB*

그림 1 Prefix-트리의 구성 예

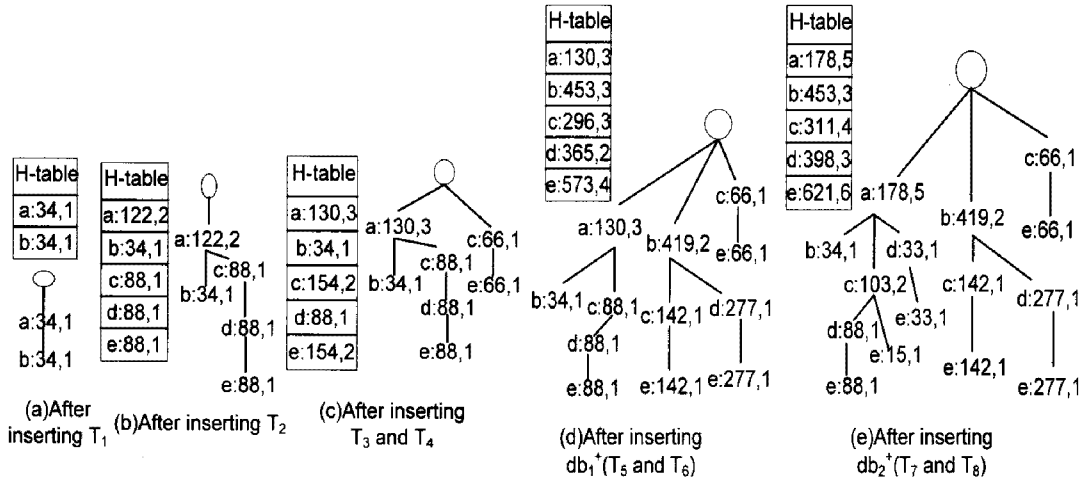


그림 2 사전적 순서에 따른 Prefix-트리의 구성 예

에 의해 첫 번째 노드가 twu=34와 tf=1을 가진 "a"가 되는 것을 보여준다. 두 번째 노드는 동일한 값을 가진 "b"가 된다. 그림 2(b)는 T_2가 트리에 추가 되었을 때를 보여준다. T_2의 tu값은 88이고, 사전적 순서에 따라 트리의 가지는 "a:88,1", "c:88,1", "d:88,1", "e:88,1"로 정의된다. 여기에서 "a"노드는 이미 존재하는 "a"노드와 공유되므로 twu값은 122(=34+88)가 되고, tf값은 2가 된다. 다른 항목들에 대해서도 그림 2(b)에서 보여주는 것과 같이 새로운 노드로 생성된다. T_3과 T_4도 동일한 방법으로 트리에 추가된다. 그림 2(d)와 그림 2(e)는 동일한 방법으로 데이터베이스 db_1+, db_2+가 트리에 추가되는 것을 보여준다.

트랜잭션이 삭제되는 경우에도 같은 방법으로 처리하게 된다. 즉 T_7을 삭제한다고 가정했을 때, 그림 2(e)의 경로 "a c e"로부터 T_7의 twu값과 tf값을 쉽게 감소시킬 수 있다. 그 다음에 경로에서 노드 e는 twu값과 tf값이 모두 0이 되어 가지로부터 삭제된다. 노드 a의 twu값과 tf값은 163과 4로 수정된다. 노드c는 노드a와 같은 방식으로 twu값과 tf값이 88과 1로 수정된다. Prefix-트리에서 항목들의 순서는 데이터의 추가, 삭제, 수정에 의한 항목들의 빈도변화에도 영향을 받지 않는다.

FP-트리[3]에서와 같이 항목들의 삽입 순서를 항목 출현 빈도(frequency)의 내림차순에 따라 트리를 구성한다면 Prefix-트리에서 공유하는 Prefix를 최대화하여 좀 더 작은 Prefix-트리를 얻을 수 있다는 것을 알 수 있다. 두 번째로 고려하는 Prefix-트리는 트리의 항목 노드들이 트랜잭션 빈도의 내림차순에 따라 배열되는 트리이다. 이 Prefix-트리는 FP-트리의 구성 방법에 따라 구성할 수도 있고, 한 번의 데이터베이스 스캔을 통하여 구성하려면 경로 조정 기법(path adjusting method) [16]을 사용하여 기존의 첫 번째 방식의 사전적 순서에 의한 트리로부터 재 구성할 수 있다. 이에 대한 자세한 방법은 CP-트리[20]를 참고하기로 한다.

그림 2(c)에서 보여주는 것과 같이 트랜잭션 T_4까지의 삽입은 사전적 순서에 따른 트리 모습과 같으나 그림 3(a)에 나타난 Header Table과 같이 항목들의 tf값(즉 빈도수) 순서가 달라지면 새로운 순서인 "a c e b d" 순서로 트랜잭션들을 삽입한다. 그림 3(c)는 최종적으로 T_8까지 삽입된 후의 트리 모습을 보여준다.

FP-트리[3]는 트리가 단지 빈발한 항목들만을 포함하기 때문에 FP-Growth알고리즘에 의한 마이닝 작업이 매우 효과적으로 처리되는 것을 보여준다. 그 이유는 트리의 어떤 가지(branch)에도 빈발한 항목들 사이에 빈발하지 않은 항목들이 나타나지 않기 때문이다. 그러

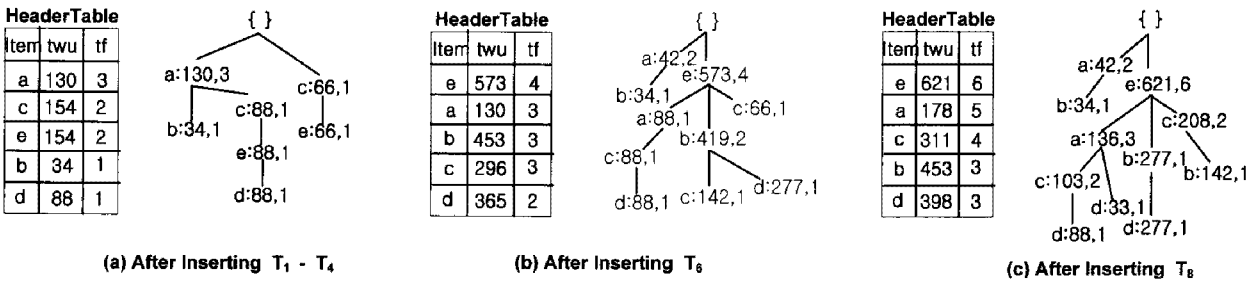


그림 3 항목 빈도수의 내림차순에 따른 Prefix-트리의 구성 예

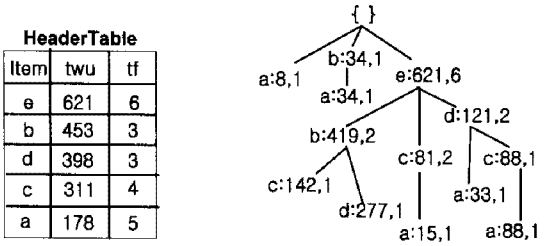


그림 4 항목의 twu 내림차순에 따른 Prefix-트리의 구성 예

나 twu값을 고려하는 유틸리티 마이닝 과정에서는 앞에서 언급된 두 가지 형태의 Prefix-트리에서는 낮은 twu 값을 갖는 항목들이 높은 twu 값을 갖는 항목들 앞에 나타날 수 있다. 예를 들어 표 1에서 *minutil*=198.9라고 하면 "a"는 낮은 twu값을 갖는 항목이다. 그러나 위의 트리 구조에서는 "a" 노드가 높은 twu값을 갖는 항목들인 "b", "c", "d" 앞에 나타날 수 있다. 따라서 세 번째로 사용될 수 있는 Prefix-트리 구조로 twu값의 내림차순에 따라 구성된 Prefix-트리를 생각할 수 있다. 이 트리 역시 항목들의 빈도수 순서에 따른 트리처럼 재구성을 통하여 만들어질 수 있다.

4.2 마이닝 알고리즘 및 성능 분석

Prefix-트리의 구성 및 유틸리티 패턴 마이닝을 위한 알고리즘은 그림 5에 나타난 단계별 작업을 통하여 이루어진다. 수행 과정은 먼저 트리의 세가지 구성 방법 중 임의의 한 형태로 Prefix-트리를 구성하고 사용자가 입력한 임계값 δ에 따라 마이닝을 수행한다. 이때 입력된 δ 값에 따라 매번 Prefix-트리를 새로이 구성하는 것이 아니라 이전에 수행한 마이닝 결과값을 적절히 활용하여 불필요한 계산을 할 필요가 없도록 하고 있다. 마이닝 과정은 유틸리티 임계값(minutil)과 트리의 각 노드에 저장된 twu 값을 이용하여 패턴 확장(pattern-growth) 방식으로 수행되므로 단계별(level-wise) 후보 검증 방법의 문제점이 발생하지 않게 된다.

FP-Growth 알고리즘과 같은 패턴 확장 방식의 마이닝 과정에서 특정 항목에 대한 조건-트리(conditional tree)를 구성할 경우 빈발하지 않은 항목들이 트리 중간에 나타나게 되면 이를 제거하는 과정이 필요하게 된다.

```
Input:  DB, group of db', db and dbmodα, δ,
Output: High Utility Patterns
begin
    foreach transaction Ti do
        Sort the items inside Ti according to the current sort order
        Insert/Delete/Modify Ti into the tree
        Update the twu and tf in the header table H
        /* The following if block is only applicable for tf-ordered tree and twu-ordered tree */
        Perform tree-restructuring operation to restructure the tree and header table H end
        if Ti is the last transaction of DB or any db'/db /dbmod then
            while there is a mining request from user do
                Input δ from the user
                if previous δ > current δ or this is the first δ value for this Prefix-tree then
                    foreach item α of H do
                        if twu (α) ≥ minutil then
                            Create Prefx-tree PTα with its header table HTα for item α
                            Call Mining (PTα, HTα, α)
                        end
                    end
                    if this is the first δ value for this Prefix-tree then
                        scan the database a second time to calculate high utility patterns from the candidate patterns
                    end
                    else
                        scan the database for only those candidate patterns which are not covered by the previous
                        result
                    end
                end
                else
                    Find the high utility patterns from the previous result without mining and second database scan
                    end
                end
            end
    end
end

Procedure Mining(T,H,α)
begin
    foreach item β of H do
        if twu(β) < minutil then
            Delete β from T and H to create conditional tree and header table
        end
    end
    Let CT be the Conditional tree of α created from T
    Let HC be the Header table of Conditional tree CT created from H
    foreach item β in HC do
        Add pattern αβ in the candidate pattern list
        Create Prefx-tree PTαβ and Header table HTαβ for pattern αβ
        Call Mining (PTαβ , HTαβ , αβ)
    end
end
```

그림 5 Prefix-트리 구성 및 유틸리티 패턴 마이닝 알고리즘

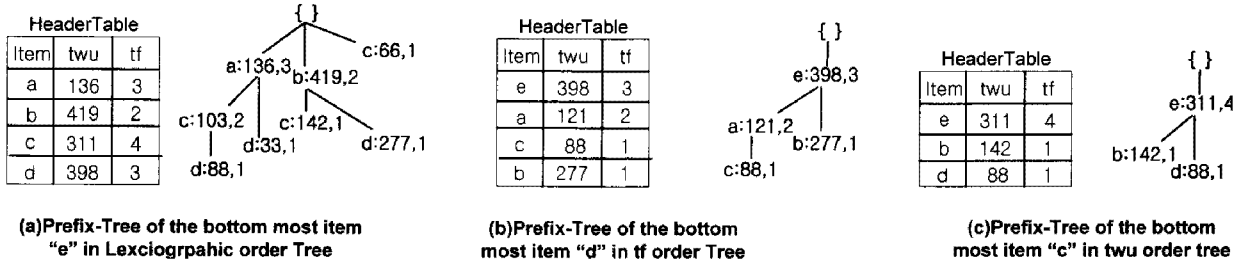
이 과정은 조건-트리의 구성 시간을 느리게 하여 결국 전체적인 마이닝 속도에 영향을 미치게 된다. twu 값을 이용하는 유틸리티 Prefix-트리에서도 조건-트리를 만드는 과정에서 낮은 twu 값을 갖는 노드가 높은 twu 값을 갖는 노드들 사이에 놓이게 되면 마이닝 속도가 떨어지게 된다. 예를 들면 minutil 값이 198.9일 경우 그림 2(e)와 그림 3(c)에서 처럼 낮은 twu 값 178을 갖는 항목 "a"가 높은 twu 값을 갖는 노드들 위에 놓이게 되면 높은 twu 값을 갖는 항목을 기반으로 하는 조건-트리를 구성하는 과정에서 불필요한 "a" 항목을 여러 번 제거해야 하므로 마이닝 성능이 떨어지게 되는 것이다. 그림 5에 나타난 알고리즘의 수행 과정은 그림 3~그림 7에 나타난 예제를 통하여 세가지 다른 형태의 Prefix-트리에 대한 구성 과정과 마이닝 절차를 설명하기로 한다.

그림 6은 세가지 다른 항목 순서에 따라 구성된 Prefix-트리에서 최하위 항목에 대한 조건 Prefix-트리의 구성 모습을 나타낸 것이다. 그림에서 알 수 있듯이 twu 값의 순서로 구성된 Prefix-트리의 조건-트리가 작은 형태를 보이고 있다. 패턴 확장 방식으로 마이닝을 하는 경우 대부분의 시간이 각 항목들에 대하여 재귀적으로 조건-트리를 구성하고 또 이를 탐색(traversal)하는 데에 소요된다. 따라서 작은 조건-트리를 생성할 경우 마이닝의 속도가 향상될 수 있다. 그림 7은 twu 값 순서로 구성된 Prefix-트리(그림 4)에 대하여 항목 "c", "d", "b"에 대한 재귀적인 조건-트리를 구성한 결과를 나타낸다. 항목 "c"에 대하여 조건-트리를 구성하는 방법을 단계적으로 설명하면 우선 그림 4의 Prefix-트리에서 "c"를 포함하는 Prefix-경로만을 모으고 twu값과 tf 값도 경로에 나타난 값만을 더하여 나타낸다(그림 7(a)). 이렇게 만들어진 항목 "c"의 prefix-트리에서 twu 값이 minutil=198 보다 작은 항목들은 유틸리티 패턴에 포함될 수 없으므로 트리에서 제거하고 또한 항목 "c"도 제거하면 조건-트리가 그림 7(b)와 같이 나오게 된다. 이 과정에서 패턴 "c"와 "ce"가 유틸리티 패턴인 것을 찾아낼 수 있다. 조건-트리를 구성하여 빈발 패턴을 찾아내는 방법은 [3]에 자세히 설명되어 있다.

위에 언급하였듯이 twu값의 내림차순에 따라 구성된 Prefix-트리가 최적의 유틸리티 마이닝 성능을 보일 수 있음을 알 수 있다. 다음 절에서는 이러한 사실을 여러 형태의 실험을 통하여 분석한 내용을 기술한다.

5. 실험 결과

우리가 제안한 Prefix-트리 구조들의 성능을 평가하기 위해서 IBM synthetic datasets(T10I4D100K,T40I10D100K)과 실환경의 데이터집합(mushroom, retail, kosarak)들로 몇 가지 실험을 수행하였다. 실환경과 관련된 데이터집합은 Frequent itemset mining dataset repository(http://fimi.cs.helsinki.fi/data/)와 UCI Machine Learning Repository(http://kdd.ics.uci.edu/)로부터 가져왔다. 그러나 이 데이터집합들은 각각의 트랜잭션에 대하여 유틸리티 값을 제공하지 못하므로 기존의 패턴 마이닝[9,11,14] 기법의 성능 평가에서처럼 각 트랜잭션에서 항목의 양과 항목의 이윤값을 임의의 숫자들을 생

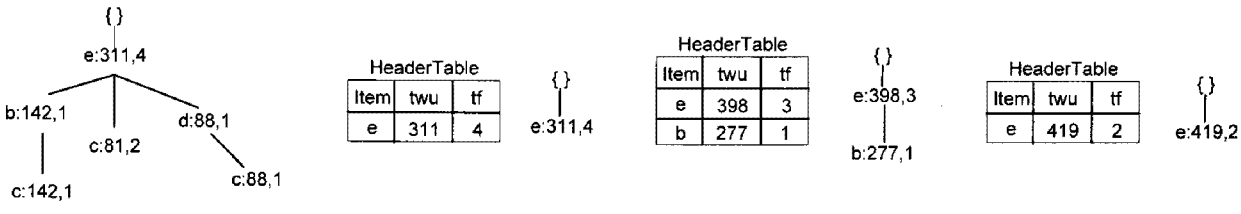


(a)Prefix-Tree of the bottom most item "e" in Lexciogrpahic order Tree

(b)Prefix-Tree of the bottom most item "d" in tf order Tree

(c)Prefix-Tree of the bottom most item "c" in twu order tree

그림 6 세가지 순서에 따른 Conditional Prefix-트리



(a) Prefix-Path for an item "c"

(b) Conditional-Tree of item "c"

(c) Prefix and Conditional-Tree of item "d"

(d) Prefix and Conditional-Tree of item "b"

그림 7 twu 값 순서로 구성된 Prefix-트리에서의 마이닝 과정

성하였다. 실환경 데이터베이스는 항목들이 대부분이 낮은 이윤값을 가지기 때문에 우리는 로그정규분포[14,15]를 사용해서 이윤값들을 생성하여 데이터들로부터 일관된 결과를 얻을 수 있었다. Synthetic 데이터 집합인 T10 I4D100K는 희박성(loose)의 특성을 갖는 데이터집합이고 Mushroom데이터집합은 고밀도(dense)의 특성을 갖는 데이터집합이다. 각각의 데이터 집합에 대하여 제안한 방법들과 기존의 Two-Phase[14], FUM과 DCG+[7]과의 성능을 비교 분석한다. 프로그램들은 Microsoft Visual C++ 6.0에서 작성되었으며, Windows XP운영 체계(pentium dual core 2.13 GHz CPU, 1GB main memory)에서 수행하였다.

5.1 대화형 마이닝에 대한 효율성 비교

그림 8은 Mushroom 데이터집합을 사용할 경우 각 알고리즘에서 minutil 값의 변화에 따른 후보 패턴의 수를 비교한 것이고, 그림 9는 마이닝 수행 시간을 비교한 결과를 보여준다. 본 논문의 알고리즘은 Prefix-T로 표시하기로 하고, 항목 순서에 따른 세가지 트리 구조를 사용할 경우 각각 Lex. Order, tf Order, twu Order로 표시한다. 우리가 제안한 세가지 트리 구조들은 모두 동일한 개수의 후보들을 갖는다. 그림 9에서 알 수 있듯이 마이닝 시간은 제안하는 기법들이 기존 방법보다 모든 minutil 값에 대하여 빠른 성능을 보여주고 있으며 제안하는 기법중에서도 twu 값 순서로 구성된 Prefix-트리를 사용할 경우가 최적의 성능을 보여주는 것을 알 수 있다. 또한 대화형 마이닝을 할 경우에는 제안하는 기법은 첫 번째 minutil 값 이후로는 Prefix-트리를 재구성할 필요가 없게 된다. minutil 값이 감소함에 따라 새로운 마이닝 작업이 필요하지만, minutil 값이 0.25일 경우의 후보 패턴들은 minutil 값 0.3에서의 후보 패턴들의 초집합(superset)이므로 minutil 값0.3후보들을 처리하기 위한 두 번째 데이터베이스 스캔 시간을 쉽게 줄일 수 있다. 높은 tu를 갖는 항목 집합들은 이미 minutil 값 0.3에서 계산되어졌기 때문에 두 번째 데이터베이스 스캔시에는 minutil 값 0.25를 위해 새롭게 추가된 후보들만을 스캔하면 되기 때문이다. 최상의 경우는 올림차순으로 마이닝 minutil 값을 변경할 때이다. 그러면 minutil 값 0.2의 후보들은 minutil 값 0.1 후보들의 부분집합이므로 마이닝 작업과 두 번째 데이터베이스의 스캔 없이 이전 결과만으로 결과를 얻을 수 있다. 이 경우에 첫 번째 minutil 값을 처리한 후에 다음 minutil 값을 위한 계산 시간은 첫 번째 처리 시간과 비교해볼 때 무시해도 좋을 정도로 작다. 또한 Two-Phase와 우리가 제안한 알고리즘의 수행 시간 차이는 minutil 값이 감소함에 따라 커진다. 표 2는 Mushroom 데이터집합에서 minutil 값이 0.3일 경우 세가지 트리 구조에 대하여 단계별 작업의 수행 시간의 분포를 보여주고 표 3은 트리의 크기를 나타내는 사용 메모리의 크기를 비교한 것이다.

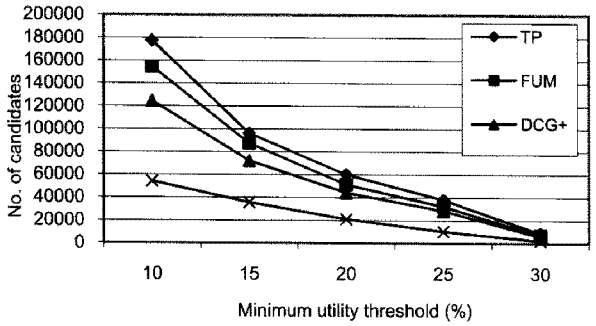


그림 8 Mushroom데이터집합에 대한 후보 항목집합의 개수 비교

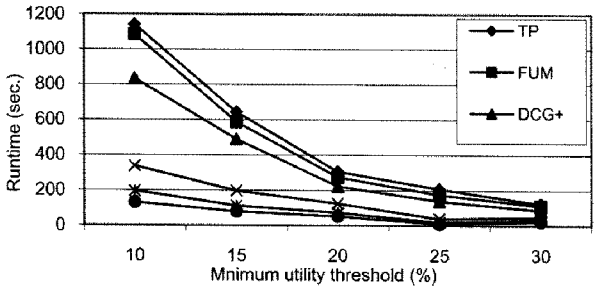


그림 9 Mushroom데이터집합에 대한 마이닝 시간 비교

표 2 Mushroom 데이터집합으로 δ=0.3에 대한 시간(Sec.) 분포

	Lex. Order	Tf Order	$IHUP_{TWU}$-Tree
Construction	5.715	5.593	5.608
Reconstruction	0	0.828	0.782
Mining	30.755	14.309	5.05
2nd DB Scan	11.15	11.06	11.83
Total	47.62	31.79	23.27

표 3 세가지 Prefix-트리의 (MB) 비교

	$IHUP_{L}$-Tree	$IHUP_{TF}$-Tree	$IHUP_{TWU}$-Tree
Mushroom	0.721	0.415	0.486
T10I4D100K	15.274	12.733	12.912
Kosarak	208.237	178.134	183.856

5.2 점증적 마이닝에 대한 효율성 비교

점증적 마이닝에 대한 실험은 제안한 세 가지 Prefix-트리의 효율성을 비교 분석하고 기존 방법과의 성능적 특징을 비교하기 위하여 Kosarak(30.5MB) 데이터집합을 가지고 시험하였다. 데이터 집합은 990,00건의 트랜잭션들과 41,270건의 서로 다른 항목들로 구성되었다. 첫 번째로 우리는 이 데이터집합에서 20만 건의 트랜잭

션들로 Prefix-트리를 만든 다음 유틸리티 한계값 0.05로 마이닝 작업을 수행하였다. 다음 또 다른 20만건의 트랜잭션들을 트리에 추가한 후 동일한 한계값을 가지고 마이닝 작업을 수행하였다. 동일한 방법으로 Kosarak 데이터집합의 남은 트랜잭션들을 트리에 추가하고 한계값 0.05로 마이닝 작업을 수행하였다. 그림 10은 이 결과를 보여준다. 각각의 트랜잭션을 추가한 후 tf Order와 twu Order인 경우는 마이닝 작업 전에 트리를 재구성하게 된다. 따라서 데이터베이스가 증가함에 따라 트리 구성과 마이닝 시간이 증가한다는 것을 그림 10에서 알 수 있다. 우리는 모든 트랜잭션을 추가 처리한 후에 그 트리에 삭제 명령을 수행하였다. 삭제 트랜잭션 크기는 10만 건이다. 첫 번째로 10만 건 트랜잭션을 삭제한 후에 한계값 0.05를 가지고 마이닝 작업을 수행하였다. 동일한 삭제 작업을 4번 더 반복 수행하였고 이 결과는 그림 11에서 보여준다. 트랜잭션을 삭제한 후에도 tf Order와 twu Order인 경우는 마이닝 작업 전에 트리를 재구성하게 된다. 따라서 데이터베이스가 감소함에 따라 트리 구성과 마이닝 시간이 감소됨을 그림 11에서 알 수 있다.

우리가 제안한 세가지 Prefix-트리들은 Kosarak 데이터집합에서 서로 다른 41270개의 항목들을 효과적으로 처리하였다. 그림 10에서는 제안한 기법이 Kosarak

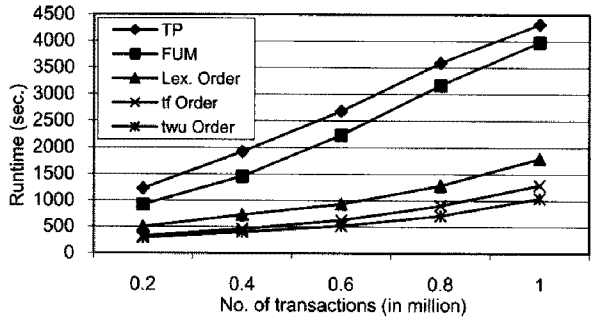


그림 10 Kosarak 데이터집합으로 db+=0.2M에 의한 데이터베이스 증가

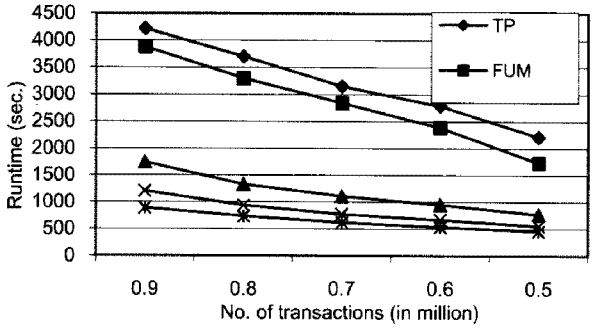


그림 11 Kosarak 데이터집합으로 db-=0.1M에 의한 데이터베이스 감소

표 4 세가지 Prefix-트리의 (MB) 비교

	Lex. Order	tf Order	twu Order	FUM
mushroom	0.721	0.415	0.486	2.256
T10I4D100K	15.274	12.733	12.912	27.49
retail	16.761	13.067	13.819	31.73
kosarak	208.237	178.134	183.856	454.296
Chain-sore	272.94	236.875	245.671	528.348

데이터집합의 100만 건의 트랜잭션과 41,270개의 서로 다른 항목들을 처리하는 좋은 확장성을 보여주고 있다. 표 4는 여러 데이터집합의 모든 트랜잭션들로 트리를 구성한 후 사용된 메모리 크기를 비교 정리한 것이다.

6. 결 론

본 논문의 주요 공헌은 현재 이용 가능한 메모리 크기 범위 내에서 대량 데이터 삽입, 삭제, 수정 작업들이 빈번히 발생하는 점증적 데이터베이스를 처리할 수 있는 새로운 유틸리티 패턴 마이닝 알고리즘을 고안한 것이 될 수 있다. 제안한 기법에서는 항목의 사전적 순서로 구성하는 Lex. Order, 항목의 빈도수에 따라 구성한 tf Order, 그리고 twu 값의 순서에 따른 twu Order의 세가지 다른 prefix-트리를 사용하고 있다. Lex. Order 트리의 특징은 매우 단순하고 쉽게 구성될 수 있으며, 많은 데이터의 삽입, 삭제, 수정이 발생 하여도 트리의 재구성 작업이 필요 없다는 것이다. tf Order의 경우는 트리 재구성은 필요하지만 적은 메모리를 필요로 장점이 있고, twu Order의 경우는 최적의 마이닝 성능을 보여주는 장점이 있는 것으로 분석되었다. 본 논문에서는 기존 연구의 문제점인 후보생성 및 검사 전략 없이 가능한 패턴증가(pattern-growth) 접근방식을 유틸리티 마이닝에 도 사용할 수 있음을 입증하였으며, 제안한 모든 트리 구조는 한번의 구성으로 여러 차례의 마이닝이 가능하여 대화식 마이닝에 적합하고, 대량의 트랜잭션들이 삽입, 삭제가 발생하여도 점진적으로 마이닝이 가능할 뿐만 아니라 많은 수의 서로 다른 항목들을 처리하는데도 효과적이고 확장 가능하다는 것을 보여주었다.

참 고 문 헌

[1] R. Agrawal, T. Imieliński and A. Swami, "Mining association rules between sets of items in large databases," *Proc. of the 12th ACM SIGMOD Int'l Conf. on Management of Data*, pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, Sep. pp.487-499, 1994.

[3] J. Han, J. Pei, Y. Yin and R. Mao, "Mining

frequent patterns without candidate generation: a frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol.8, pp.53-87, 2004.
[4] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, vol.59, pp.603-626, 2006.
[5] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong and Y.-K. Lee, "Mining high utility patterns in incremental databases," *Proc. of ICUIMC*, pp.653-663, Feb. 2009.
[6] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, vol.59, pp.603-626, 2006.
[7] U. Yun, "WIS: Weighted interesting sequential pattern mining with a similar level of support and/or weight," *ETRI Journal*, vol.29, no.3, pp.336-352, Jun. 2007.
[8] X.Li, Z.-H. Deng and S. Tang, "A fast algorithm for maintenance of association rules in incremental databases," Advanced Data Mining and Application (ADMA 06), vol.4093, pp.56-63, Jul. 2006.
[9] S. Zhang, J. Zhang and C. Zhang, "EDUA: An efficient algorithm for dynamic database mining," *Information Science*, vol.177, pp.2756-2767, 2007.
[10] J. Hu and A. Mojsilovic, "High utility pattern mining: A method for discovery of high utility item sets," *Pattern Recognition*, vol.40, pp. 3317- 3324, 2007.
[11] Y. Liu, W.-K. Liao, A. Choudhary, "A fast high utility itemsets mining algorithm," *Proc. 1st Intl. Conf. on Utility-Based Data Mining*, pp.90-99, Aug. 2005.
[12] F. Tao, "Weighted association rule mining using weighted support and significant framework," *Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp.661- 666, 2003.
[13] B. Barber and H.J. Hamilton, "Extracting share frequent itemsets with infrequent subsets," *Data Mining and Knowledge Discovery*, vol.7, pp.153-185, 2003.
[14] Y. Liu, W.-K. Liao and A. Choudhary, "A Two phase algorithm for fast discovery of high utility of itemsets," *Proc. of the 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining(PAKDD'05)*, pp.689-695, May 2005.
[15] Y. Liu, W.-K. Liao, A. Choudhary, "A fast high utility itemsets mining algorithm," *Proc. 1st Intl. Conf. on Utility-Based Data Mining*, pp.90-99, Aug. 2005.
[16] J.-L. Koh, S.-F. Shieh, "An efficient approach for maintaining association rules based on adjusting FP-tree structures," *Proceedings of the DASFAA'04*, pp.417-424, 2004.
[17] X.Li, Z.-H. Deng and S. Tang, "A fast algorithm for maintenance of association rules in incremental databases," *Advanced Data Mining and Application (ADMA 06)*, vol.4093, pp.56-63, Jul 2006.
[18] C. K.-S. Leung Q.I. Khan, Z. Li and T. Hoque, "CanTree: a canonical-order tree for incremental frequent-pattern mining," *Knowledge and Information Systems*, vol.11, no.3, pp.287-311, 2007.
[19] A. Erwin, R.P. Gopalan, N.R. Achuthan, "CTU-Mine: an efficient high utility itemset mining algorithm using the pattern growth approach," *Proc. of the Seventh IEEE Int. Conf. on Computer and Information Technology (CIT'07)*, pp.71-76, Oct. 2007.
[20] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong and Y.-K. Lee, "CP-tree: A tree structure for single pass frequent pattern mining," *Proc. of the 12th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'08)*, May 2008.

정 병 수
서울대학교 공과대학 컴퓨터공학과에서 학사. 한국과학기술원 전산학과에서 전산학 석사. Georgia Institute of Technology, College of Computing에서 박사 학위를 취득하고, 경희대학교 전자정보대학 컴퓨터공학 전공에서 부교수로 재직 중이며, 현재 웹 로그 분석, 스트림 데이터 마이닝, 패턴 탐색 기법 등을 연구 중이다. 관심 분야는 데이터 마이닝, 모바일 데이터베이스, 플래쉬 메모리 저장시스템 등이다.

아메드 파한
방글라데시 다카 대학에서 학사 및 석사 학위를 취득하고 현재 경희대학교 전자정보대학 컴퓨터공학과에서 박사 과정에 있으며 현재 데이터 마이닝, 지식 탐사 기법 등을 연구 중이다. 관심 분야는 스트림 데이터 마이닝, 순차적 패턴 탐색, 유틸리티 패턴 탐색 등이다.

이 인 기
이화여자대학교 컴퓨터공학과에서 학사와 석사를 취득하고 KTF 정보시스템실에서 CRM 개발을 하였다. 현재는 이화여자대학교 컴퓨터공학과에서 박사과정에 있다. 관심분야로는 데이터마이닝, 온톨로지, 지식기반 시스템 등이다.

용 환 승
서울대학교 컴퓨터공학과에서 학사. 석사 및 박사학위를 취득하고, 한국전자통신연구소에서 연구원, IBM T.J. Watson 연구소에서 방문연구원으로 근무했었으며 현재 이화여자대학교 컴퓨터공학과에서 교수로 재직중이다. 관심분야는 온톨로지 기반 정보시스템, OLAP 및 데이터 마이닝, 유비쿼터스 데이터베이스 등이다.