碩士學位論文

증가하는 데이터베이스에 대한 다중 최소 임계치 기반 정규 패턴 마이닝

濟州大學校 大學院

컴퓨터工學科

崔 亨 吉

2013年 12月

증가하는 데이터베이스에 대한
다중 최소 임계치 기반 정규 패턴 마이닝

指導敎授 李 尙 俊

崔 亨 吉

이 論文을 工學 碩士學位 論文으로 提出함

2013年 12月

崔亨吉의 工學 碩士學位 論文을 認准함

審査委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

濟州大學校 大學院

2013年 12月

Regular Pattern Mining with Multiple Minimum Supports in Incremental Database

Hyong-gil Choi

(Supervised by professor Sang-Joon Lee)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Computer Engineering

2013. 12.

This thesis has been examined and approved.

Thesis director, _____

Thesis director, _____

Thesis director, _____

December 2013

Department of Computer Engineering

Graduate School

Jeju National University

목 차

표 목 차

그 림 목 차

국문초록

증가하는 데이터베이스에 대한 다중 최소 임계치 기반 정규 패턴 마이닝

컴퓨터공학과 최형길
지 도 교 수 이상준

기존의 전통적인 빈발 패턴 마이닝은 전체 트랜잭션 데이터베이스에 대하여 단일 최소 임계치를 기반으로 빈발 패턴을 마이닝한다. 단일 최소 임계치를 이용하는 방법은 아이템에 동일한 임계치가 적용되어 레어 아이템 문제(rare item problem)를 유발시킨다. 한편, 일정 주기마다 발생하는 패턴을 정규 패턴(regular pattern)이라고 한다. 실세계에서는 빈발한 패턴뿐만 아니라 점진적으로 증가하는 데이터 내에서 주기적으로 발생하는 패턴정보 처리의 중요성이 증가하고 있다. 본 논문은 레어 아이템 문제를 해결하고 정규 패턴을 마이닝하는 기법을 제안한다.

ABSTRACT

Regular Pattern Mining with Multiple Minimum Supports in Incremental Database

Choi, Hyong-Gil
Department of Computer Engineering
Graduate School
Jeju National University

Previous traditional frequent pattern mining methods discover all of the frequent patterns with a single minimum threshold for the whole database. By using the single threshold, they suffer from the rare item problem due to setting the same minimum support for all items. In real world applications, the significance of processing patterns that occur regularly in incremental database is increasing as well as the frequent patterns. In this paper, we propose a method that can be applied for regular pattern mining and also solve the rare item problem.

Ⅰ. 서 론

데이터 마이닝은 거대한 트랜잭션 데이터베이스 내에서 유용한 정보나 연관 규칙과 같은 의사 결정에 필요한 지식을 탐색하는 것을 의미한다. 새로운 기술과 온라인 커뮤니티 등의 발달에 의해 더 많은 데이터들이 발생한다. 따라서 비즈니스, 기업 업무 등 많은 분야에서 거대한 데이터베이스로부터 유용한 정보를 검색해 내는 기술의 필요성이 증가하였고 많은 분야의 데이터마이닝 관련 연구가 진행되고 있다.

연관 규칙이나 패턴 마이닝 또한 이러한 데이터 마이닝을 위한 중요한 기법 중 하나이다. 패턴 마이닝은 트랜잭션 내에서 어떤 아이템 a 가 나타날 때 같이 나타나는 아이템 패턴집합을 검색해내는 기법이다. 상품 판매 데이터에서 아이템 a와 연관되어 팔리는 상품은 무엇인지를 분석해내고 기업의 측면에서 연관된 상품을 가까이 배치할 수 있다. 기존에 진행된 연구의 알고리즘 [1, 2]은 데이터마이닝의 근간이 된 기법이다. 이 기법들을 좀 더 살펴보면 D를 트랜잭션 데이터베이스, X를 아이템 집합, Y를 하나의 아이템이라고 할 때 D 내 $X \cup Y$, 일때 $X \subseteq I$, $Y \subseteq I$, $X \cap Y = \emptyset$, 를 포함하는 트랜잭션들이 s%이면 $X \rightarrow Y$의 빈발도를 s라 한다. 이러한 기법을 이용한 마이닝은 사용자 정의 최소 임계치 그리고 최소 빈도수 이상의 신뢰도와 빈도수를 가지는 모든 연관 규칙이나 패턴을 찾는 것이며 많은 연구들 [1-5]이 진행되어 왔다. 일반적으로 패턴 마이닝 모델은 전체 데이터베이스에 대해 오직 하나의 최소 빈도수를 사용한다. 이는 암묵적으로 모든 아이템들이 동일한 특성을 가진다고 가정하는 것이지만 실세계 응용에서는 개별적인 특성을 가지고 있을 수 있으며 따라서 이를 고려할 필요가 있다. 또한 트랜잭션의 모든 아이템에 동일한 빈발도 임계치 적용으로 인해 높은 임계치가 적용시 빈발도가 낮은 레어 아이템(rare item)이 포함된 패턴을 마이닝하지 못하거나, 레어 아이템이 포함된 패턴을 마이닝하기 위해 임계치를 낮춤으

로 인해서 너무 많은 패턴이 발견되는 문제가 생기는데 이 문제를 레어 아이템 문제(rare item problem) [3, 4]라고 한다. 이런 문제를 해결하기 위해 다중 최소 임계치를 적용한 MMS (Multiple Minimum Supports) 모델이 제안되었다.

다중 최소 임계치를 고려한 연관 규칙 마이닝은 개별적인 아이템에 서로 다른 최소 임계치를 설정하여 드물게 발생하지만 중요한 아이템이 포함된 연관 규칙을 마이닝 한다. 레어 아이템 문제를 해결하기 위한 초기 연구들 [3-5]은 아이템들의 빈발도에 따라 데이터를 몇 개의 블록으로 분할하거나 관련 있는 레어 아이템들을 하나의 그룹으로 만드는 방법을 사용했다. 그러나 이러한 접근법들은 애드혹 그리고 근사적 방법이다. 이러한 문제를 해결하기 위해 다중 최소 임계치를 고려한 연관 규칙 마이닝 모델이 제안되었으며 이를 통해 아이템들의 개별적인 특성 및 임계치를 반영하여 연관 규칙을 마이닝 한다.

한편, 주기적으로 발생하는 패턴을 정규패턴(Regular Pattern) [6-8]이라고 한다. MMS 모델은 아이템의 빈발도에만 초점을 맞추고 있으므로 이러한 정규 패턴을 마이닝할 수 없다. 최근에는 보다 다양한 정보의 마이닝이 필요해지고 있으며 주말이나 연휴 같은 특정 기간에 수요가 늘어나는 상품, 정기적인 방문객, 이용자 수와 같은 정규 패턴 정보가 필요해지고 있다. 또한 기존의 다중 최소 임계치 기반의 패턴 마이닝 모델은 이러한 패턴을 마이닝 할 수가 없고, 실세계에서는 빈발한 패턴뿐만 아니라 주기적으로 발생하는 패턴정보의 필요성이 증가하고 있다 [8].

트랜잭션 데이터베이스의 크기는 정적이지 않고 상품의 판매 정보 등이 누적되면서 동적으로 꾸준히 커져간다. 따라서 본 논문은 트랜잭션 데이터베이스의 증가를 효과적으로 처리하면서, 주기적으로 발생하는 패턴을 마이닝하는 방법을 제안한다. 본 논문의 주요 기여는 다음과 같다

1. 연관 규칙을 마이닝 하기 위한 정보를 캡쳐 하는 트리 구조, RMIS-Tree를 제안한다. 구축된 RMIS-Tree는 기반으로 검색 공간을 감소시키고 다중 최소 빈도수와 정규도 모두를 고려한다.

2. 연관 규칙을 마이닝 하기 위한 데이터베이스의 성장 과정 중에서 다중 최소 빈도수 그리고 정규도를 고려하고 검색 공간을 효율적으로 감소시키기 위한 RMIS-Growth 알고리즘을 제안한다. 제안된 알고리즘의 적용을 통해 검색 공간

을 감소시킴으로써 효율적으로 연관 규칙을 마이닝 할 수 있다.

또한 본 연구가 제시하는 기법은 주기적으로 발생하는 질병의 발생 패턴, 주기적으로 의료시설에서 사용되는 약품, 정기적으로 팔리거나 특정 기간 동안에 팔리게 되는 상품들의 패턴과 같이 주기적으로 발생하는 패턴의 분석이 필요한 분야에 응용할 수 있다. 그리고 특정 기념일이나 성수기와 같은 단 기간에서 나타나는 레어 아이템을 높은 정규도 임계치와 아이템별 최소 임계치를 이용함으로써 극복 가능하다.

본 논문은 총 5장으로 구성되어 있다. 1장은 본 논문의 연구 배경과 목적에 대해 설명하고 2장은 기존에 연구된 관련 알고리즘에 대해 설명한다. 3장에서 본 논문이 제안하고자 하는 기법에 대해 소개하고 4장은 본 논문이 제안한 기법의 실험결과를 분석하고 5장에서 결론을 서술한다.

II. 관련 연구

본 장에서는 점진적으로 증가하는 데이터베이스에서 다중 최소 임계치를 기반으로 정규 패턴을 마이닝 하기 위한 제안하는 알고리즘과 관련 있는 기존의 알고리즘들에 대해 분석한다. 먼저, 전통적인 빈발 패턴 마이닝을 위한 알고리즘인 Apriori [1]와 전통적인 빈발 패턴 마이닝에서 Apriori의 단점을 해결하기 위해 제안된 트리 기반의 FP-Growth [2] 알고리즘에 대해 설명한다. 그리고 Apriori를 기반으로 하는 레어 아이템을 포함한 패턴들을 마이닝 하기 위한 다중 최소 빈발도 임계치 기반의 MSApriori [3] 알고리즘에 대해 분석하고 FP-Growth를 기반으로 제안된 레어 아이템이 포함한 중요 패턴들을 마이닝 하는 CFP-Growth [4] 그리고 CFP-Growth++ [5] 알고리즘들을 분석한다. 마지막으로 정규 패턴을 마이닝 하기 위한 알고리즘 [6-8]에 대해 설명한다.

2.1 Apriori

빈발 패턴을 마이닝 하기 위한 초기 해결책은 Apriori [1] 알고리즘이며, 단일 최소 빈발도 임계치를 기반으로 마이닝을 수행한다. 즉, Apriori 알고리즘은 사용자 정의 최소 빈발도 임계치 이상의 빈도수를 가지는 모든 중요 패턴들을 마이닝 하며, 마이닝 된 패턴은 빈발 패턴이라 한다. Apriori는 단일 최소 임계치를 적용한 모델 연관 규칙 마이닝 알고리즘으로 많은 연관 규칙이나 패턴 마이닝에 이용되었다. 또한, Apriori는 선험적 규칙을 이용하였다. 예를 들어, I = {a, b, c, d, e...}를 데이터베이스 내 아이템 집합이라고 할 때, I의 원소 중 빈발한 아이템들로 구성된 집합 F가 $F \subset I$ 이고 F = {a, b, c}라면 F의 원소로 구성된 아이템 집합은 빈발한 아이템 혹은 집합이 될 수 있다. 그러나 F의 원소로 구성된 아이템 집합에 빈발하지 않은 아이템이 추가가 된다면 그 집합은 더 이상 빈발한 아이템 집합이 될 수 없다. 이는 안티 모노톤 속성 [1]이라고 하며, 효율적인 패턴 마이닝을 위해 사용된다. 즉, 아이템 집합 X가 있을 때, 만약 X가 빈발하지 않은

아이템 집합이면 이를 포함하는 슈퍼 패턴(super pattern) 또한 빈발하지 않다는 것을 의미한다. 반대로 X가 빈발한 아이템 집합이면 이를 포함하는 슈퍼 패턴은 빈발 패턴이 될 수 있다. 이를 이용하면 마이닝 과정에서 생성된 비 빈발 아이템 집합에 대하여 더는 마이닝 과정을 진행하지 않음으로써 좀 더 효과적으로 검색 공간을 제거하여 효율적인 마이닝을 수행하는 것이 가능하다. Apriori의 패턴 마이닝 방식은 빈발한 다음과 같다. 먼저, 길이가 1인 빈발한 아이템으로 구성된 빈발한 아이템 집합을 데이터베이스 스캔을 통해 마이닝 한다. 다음에 이를 기반으로 길이가 2인 후보 아이템 집합들을 생성하고, 또 한 번의 데이터베이스 스캔을 통해 실제 길이가 2인 빈발 아이템 집합들을 찾아낸다. 이와 같은 방법으로, n-1 길이의 빈발한 아이템 집합을 구하고 그 집합의 원소를 조합하여 길이가 n인 후보 아이템 집합을 모두 구한다. 그 후에 데이터베이스를 스캔해서 길이 n인 아이템 집합들의 빈발도를 구하고 최소 빈발 임계치보다 크거나 같은 집합을 제외한 아이템 집합들을 제거하여 마이닝 과정을 진행한다. 그 후에 n을 1씩 증가시키며 위 과정을 더 이상 후보군을 생성하지 못할 때까지 반복한다. 따라서 이와 같은 방식을 사용하는 알고리즘에서는 길이가 가장 긴 빈발한 아이템 집합의 길이만큼 데이터베이스를 스캔해야 한다는 단점을 지닌다. 데이터베이스 스캔은 많은 시간을 소요하는 작업이므로 Apriori는 I/O에 대한 높은 코스트를 필요로 한다.

표 1. 트랜잭션 데이터베이스

TID	Items
1	1 3 4
2	2 3 5
3	1 2 3 5
4	2 5

(1) 트랜잭션 데이터베이스

아이템	빈발도
1	2
2	3
3	3
5	3

(2) 빈발도

트랜잭션이 표 1의 왼쪽과 같이 구성이 되었고 최소 빈발도 임계치가 2라고 가정한다면 최소 빈발도 임계치인 2이상의 빈발도를 보인 아이템(n-1 , n = 2)의

빈발도는 표 1의 오른쪽과 같게 된다. 또한 빈발한 길이 1인 아이템의 집합은 {1, 2, 3, 5} 이 된다. 그리고 이 빈발한 아이템 집합으로부터 길이가 2인 후보군 집합은 {1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}가 된다. 그리고 데이터베이스를 스캔하고 각 후보군 집합의 빈도수를 검색한다. 검색 결과 {1, 2}의 빈도수는 1, {1, 3}은 2, {1, 5}는 1, {2, 3} 은 2, {2, 5} 는 2, {3, 5} 가 2로 최소 빈발도 임계치 2이상인 패턴은 {1, 3}, {2, 3}, {3, 5} 가 된다. 그리고 또 다시 이 패턴으로부터 길이가 3인 후보군 집합을 생성하고 데이터베이스를 검색한다. 검색 결과 {1, 3, 5} 가 2의 빈발도를 보이게 되고 이 패턴으로 길이가 4인 후보군을 생성하지 못하므로 빈발한 패턴 마이닝은 종료된다.

이 과정에서 Apriori 는 n-1 길이의 아이템 패턴으로부터 n 길이의 후보군 패턴을 생성하게 되는데 너무 많은 패턴을 생성하게 된다. 게다가 n 길이의 패턴을 검색하기 위해서 데이터베이스를 n번 검색하게 되는 문제와 단일 빈발 임계치로 인한 레어 아이템 문제가 있다. 레어 아이템 문제는 Apriori와 같은 전통적인 단일 임계치 모델에서 드물게 발생하는 중요한 아이템을 포함한 중요한 패턴을 마이닝 할 때 발생한다. 만약, 이러한 패턴을 마이닝 하기 위해서 최소 빈발도 임계치를 낮게 설정하면 드물게 발생하는 중요한 아이템을 포함한 중요 패턴을 마이닝 하는 것이 가능하지만, 동시에 너무 많은 의미 없는 패턴들이 마이닝 된다. 반대로 이러한 의미 없는 패턴들을 제거하기 위해 높은 최소 빈발도 임계치를 설정하면 드물게 발생하는 중요 아이템을 포함한 패턴들을 마이닝 하지 못하는 문제가 발생한다.

2.2 MSapriori

단일 최소 빈발도 임계치 기반의 전통적인 빈발 패턴 마이닝에서 발생하는 레어 아이템 문제를 해결하기 위해서 다중 최소 임계치(Multiple Minimum Supports)를 기반으로 하는 모델이 제안되었다. MSapriori [3]는 Apriori를 기반으로 다중 최소 임계치 모델을 적용함으로써 레어 아이템 문제를 해결하였다. 이 모델은 트랜잭션 데이터베이스에 등장하는 모든 아이템에 각각의 최소 임계치를

부여함으로써 아이템들의 특성과 다양한 빈발도를 반영할 수 있게 된다. 아이템 집합 I = {a, b, c, d, e...} 이고 각 아이템에 MIS 값이 할당되었다면 아이템 집합 I의 MIS 값은 min[MIS(a), MIS(b), MIS(d)..] 와 같다. 예를 들어 MIS(빵) = 2%, MIS(우유) = 0.1%, MIS(쥬스) = 0.2% 이라고 가정하고 아이템 집합 {빵, 쥬스} 의 빈발도가 0.15% 라면 이 아이템 집합은 min(2, 0.2) 보다 작으므로 비 빈발 집합이다. 그러나 아이템 집합 {빵, 우유} 의 빈발도가 0.3% 라면 이 아이템 집합은 min(2, 0.1) 보다 크므로 빈발한 집합이다.

Apriori는 단일 임계치를 적용한 반면 MSapriori는 다중 최소 임계치를 반영하였으므로 안티 모노톤 속성이 더는 성립하지 않는다. 즉 빈발한 아이템 집합의 일부 부분집합은 빈발하지 않게 될 것이다. 만약 아이템 집합 I = {a, b, c, d} 이고 각각 MIS(a) = 10%, MIS(b) = 20%, MIS(c) = 30%, MIS(d) = 35% 이고 {b, c}의 빈발도가 13%, {b, d}의 빈발도가 14% 를 보였다면 이 두 아이템 집합은 MIS(b)의 값보다 작은 빈발도를 보이므로 모두 비 빈발하게 되지만 {a, b, d}의 빈발도가 8%를 보였다고 한다면 집합 {a, b, c}는 mis(a)보다 높은 빈발도를 보였으므로 빈발 집합이 된다. MSApriori 알고리즘은 Apriori를 기반으로 레어 아이템 문제는 해결하였으나, 많은 수의 후보 패턴을 생성하고 최대 길이가 n인 패턴을 마이닝 하기 위해 n 번의 데이터베이스 스캔을 필요로 한다는 단점을 가진다.

2.3 FP-Growth

Apriori가 가진 문제점을 해결하기 위해 트리 구조 기반의 FP-Growth 알고리즘 [2]이 제안되었다. FP-Growth와 Apriori는 빈발 패턴을 마이닝 하기 위한 잘 알려진 알고리즘들로서 많은 알고리즘들이 이를 기반으로 한다. FP-Growth는 Apriori와 달리 후보군을 생성하는 대신 FP-Tree를 생성함으로써 많은 후보군이 생성되는 문제와 데이터베이스 스캔 횟수가 많아지는 문제를 해결하였고 데이터 마이닝의 많은 연구에 이용되고 있다. FP-Growth는 먼저 첫 번째 데이터베이스 스캔을 통해서 각 아이템의 빈발도를 구한다. 그리고 두 번째 데이터베

이스 스캔에서 최소 임계치보다 큰 아이템들을, 즉 빈발한 아이템들을 구해진 빈발도의 역순으로 아이템이름과 빈발도로 구성된 테이블을 생성한다. 이 단계에서는 FP-Tree를 구축하기 위해, 데이터베이스 각 트랜잭션에 대해 빈발하지 않은 아이템들을 제거하고 아이템들을 빈발도 내림차순으로 정렬하여 트리에 삽입한다. 삽입되는 각 트랜잭션은 FP-Tree에서 하나의 경로가 된다. 예를 들어, 표 2의 트랜잭션 데이터베이스에서, 첫 번째 데이터베이스 스캔을 통해 구해진 각 아이템의 빈도수는 오른쪽 헤더 테이블과 같다. 또한, 이를 빈도수의 내림차순으로 정렬하면 {B, D, A, E, C}가 된다. 따라서 표 2의 첫 번째 트랜잭션은 {B, D, A, E}로 정렬되어 FP-Tree에 삽입된다. 같은 방법으로 모든 트랜잭션들은 표 2의 정렬된 순서(Ordered Items)와 같이 정렬되어 FP-Tree에 삽입된다.

표 2. 트랜잭션 데이터베이스, 헤더 테이블

TID	ITEMS	Ordered ITEMS
1	E A D B	B D A E C
2	D A C E B	B D A E C
3	C A B E	B A E C
4	B A D	B D A
5	D A C E B	D A E
6	D B	B D
7	A D E	D A E
8	B C	B C

(1) 트랜잭션 데이터베이스

아이템	빈발도
B	6
D	6
A	5
E	4
C	3

(2) 헤더 테이블

표 2의 오른쪽에 있는 헤더 테이블은 FP-Tree를 구성하기 위한 헤더 테이블이다. 헤더 테이블은 각 아이템의 이름과 각 아이템별 빈발도로 구성되고 표 2의 트랜잭션 데이터베이스에 대한 헤더 테이블은 오른쪽과 같이 생성이 된다. 다음 단계로 FP-Tree를 구성하기 위해 먼저 Root 노드를 하나 생성하고 현재 노드 N으로 한다. 그리고 데이터베이스의 각 트랜잭션의 아이템을 헤더 테이블에 존재하지 않는 아이템은 제외하고 헤더 테이블의 순서대로 정렬한다. 정렬된 아이템을 순서대로 현재 노드의 자식 노드들 중에 그 노드가 존재하는지 확인하고 없

으면 노드 이름과 빈발 횟수 1을 표시한다. 만약 현재 노드에 그 노드가 존재한다면 새로 노드를 추가하는 대신 빈발 횟수를 1 증가시키고 그 노드를 새로운 현재 노드 N으로 한다.

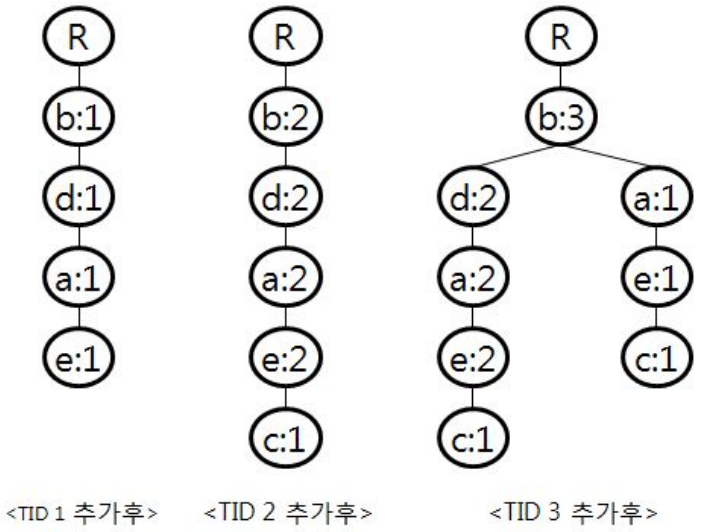


그림 1. FP-Tree의 변화도

그림 1은 이 과정을 보이고 있다. 첫 번째 트랜잭션 {E, A, D, B} 는 헤더 테이블의 순서에 의해 {B, D, A, E}로 정렬된다. 그리고 현재 노드인 Root에 자식 노드가 없으므로 트랜잭션의 아이템 B는 Root의 자식 노드로 새로 생성하고 빈발도를 1로 표시한다. B는 새로운 현재 노드가 되며 B의 자식 노드가 없으므로 D는 B의 자식 노드로 새로 생성이 된다. A와 E에도 동일한 과정을 반복한다. 2번째 트랜잭션 {D, A, C, E, B}는 {B, D, A, E, C}로 정렬되고 Root로부터 트리에 추가한다. Root의 자식 노드들 중에 B가 존재하므로 B의 빈발 횟수를 1증가시켜서 2로 변경하고, B의 자식 노드들 중에 D가 존재하므로 D의 빈발 횟수도 1 증가시킨다. 이 과정을 모든 트랜잭션에 반복한다. 이 과정이 모두 끝나면 FP-Tree는 그림 2와 같게 된다.

FP-Tree 생성이 끝나면 이 트리와 헤더 테이블을 이용해서 빈발한 집합을 찾아낸다. 헤더 테이블은 빈발도의 내림차순으로 정렬이 되어 있으므로 테이블의 가장 아래에 있는 아이템부터 상향식으로 검색한다. 헤더 테이블의 가장 밑에 있

는 아이템 C가 포함된 트리의 경로를 Root까지 따라 올라가면서 해당 경로의 아이템과 빈발도를 모두 모은다. 이때 각 경로의 아이템의 빈발도는 C의 빈발도와 같다. 모은 아이템의 빈발도가 지정한 최소 임계치보다 작은 아이템을 제외하고 다시 빈발도 순으로 내림차순 정렬하고 c의 조건부 FP-Tree를 구성한다. 이 조건부 FP-Tree가 단일 경로이면 해당 경로의 아이템들을 조합하여 빈발 패턴을 마이닝하고 단일경로가 아니라면 조건부 FP-Tree를 구성하는 것을 반복하며 빈발한 패턴을 마이닝 한다. FP-Growth는 데이터베이스 스캔 횟수를 줄여주고 데이터베이스를 트리로 압축해주는 효율적인 면을 보여주지만 단일 임계치로 인해 레어 아이템의 특성을 반영하지 못한다.

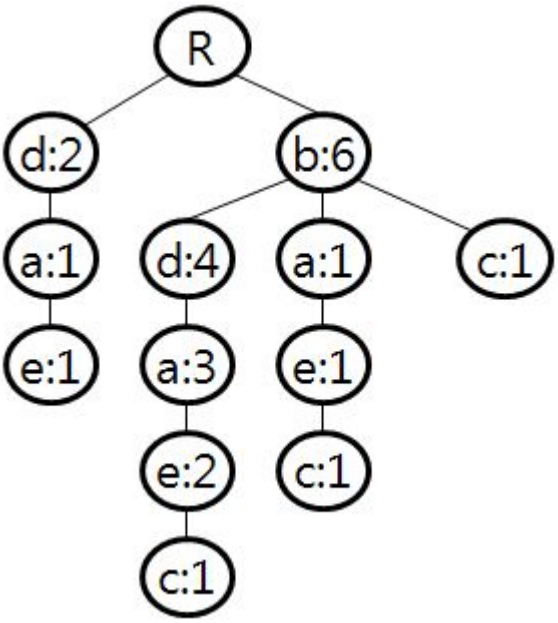


그림 2. 전체 트랜잭션 처리후 FP-Tree

2.4 CFP-Growth

CFP-Growth [4]는 FP-Growth 기반의 알고리즘으로, Apriori 기반의 MSApriori 알고리즘이 지닌 많은 수의 후보 패턴 생성과 다중 데이터베이스 스캔 문제를 해결하기 위해 제안되었다. CFP-Growth는 FP-Growth 알고리즘에 MMS 모델을 적용함으로써 단일 임계치로 인한 레어 아이템 문제를 해결했다.

CFP-Growth 는 MSapriori 처럼 각각의 아이템에 MIS(Multiple Item Supports) 값을 부여한다. CFP-Growth는 1회의 데이터베이스 검색만을 수행한다. CFP-Growth는 FP-Tree와 유사하지만 조금 다르게 빈발도가 아닌 MIS 값의 내림차순으로 정렬된 아이템 이름과 MIS 값으로 구성된 헤더 테이블을 생성한다. 그리고 데이터베이스를 스캔하며 각 트랜잭션을 헤더 테이블의 아이템 순으로 정렬하고 FP-Tree 와 같은 방법으로 MIS-Tree를 생성한다. 트리 생성이 끝나면 헤더 테이블의 MIS 값보다 작은 빈발도를 보인 아이템을 헤더 테이블과 트리 상에서 제거한다. 가지치기가 끝난 후 트리 상에서 노드가 제거됨으로 인해 똑같은 이름의 자식 노드를 가지고 있는 노드가 있다면 그 노드의 자식 노드들 중에 같은 이름의 노드를 합치는 합병(Merge) 작업을 수행한다. 그리고 빈발 패턴이 안티모노톤 속성을 만족하지 않기 때문에, CFP-Growth는 빈발 패턴을 모두 찾기 위해 각각 아이템의 조건부 패턴 베이스가 공집합이 될 때까지 접미 패턴을 만든다. 이러한 공집합이 될 때까지 CFP-Growth 알고리즘이 마이닝 과정을 반복하여 수행함으로써 너무 많은 시간이 소요된다는 문제가 발생한다.

2.5 CFP-Growth++

CFP-Growth++ [5]는 CFP-Growth의 문제점인 빈발한 패턴을 만들 수 없는 아이템이 테이블 상단부에 존재할 가능성이 있고, 이 아이템들로 빈발 패턴을 생성하려고 시도하게 되는 문제점을 해결하였다. 즉, CFP-Growth 알고리즘이 마이닝 과정을 조건부 패턴 베이스가 공집합이 될 때까지 반복함으로써 많은 마이닝 실행 시간을 소비하는 문제를 해결하기 위해서 CFP-Growth++ 알고리즘이 제안되었다. CFP-Growth++는 먼저 각 아이템을 MIS의 내림차순으로 정렬하고 각각의 빈발도와, MIS 값을 MIS-list에 기록한다. 그리고 데이터베이스를 스캔하며 MIS-list에 각 아이템의 빈발도를 기록하고 트랜잭션의 아이템을 MIS-list의 순으로 정렬하고 FP-Tree와 유사한 방법으로 MIS-tree에 각 아이템의 이름과 해당 경로에서 나타난 아이템들의 빈발도를 1씩 증가시키며 표시한다. 데이터베이스 스캔이 끝난 뒤 MIS-list를 아래서부터 검색해서 MIS 값보다 작은 빈발도

를 보인 아이템을 MIS-list와 MIS-Tree로부터 제거한다. 그리고 CFP-Growth와 같이 자식 노드들 중에 같은 이름을 가진 노드들을 합치는 합병 작업을 수행한다. 합병 작업이 끝나면 MIS-list 내에서 빈발도가 MIS 값보다 작은 값을 보인 아이템들을 대상으로 그 아이템이 트리 상에서 마지막 노드인 것들을 삭제하고 MIS-Tree 최소화시키고 트리 생성이 종료된다. 그리고 MIS-list의 하단에 있는 아이템부터 빈발한 패턴을 검색한다. CFP-Growth++는 CFP-Growth의 문제점을 해결하였으나 정규 패턴을 마이닝 하지 못한다.

2.6 정규 패턴 마이닝

정규 패턴 마이닝은 데이터베이스 내에서 주기적으로 등장하는 패턴을 마이닝 하기 위한 기법이다.

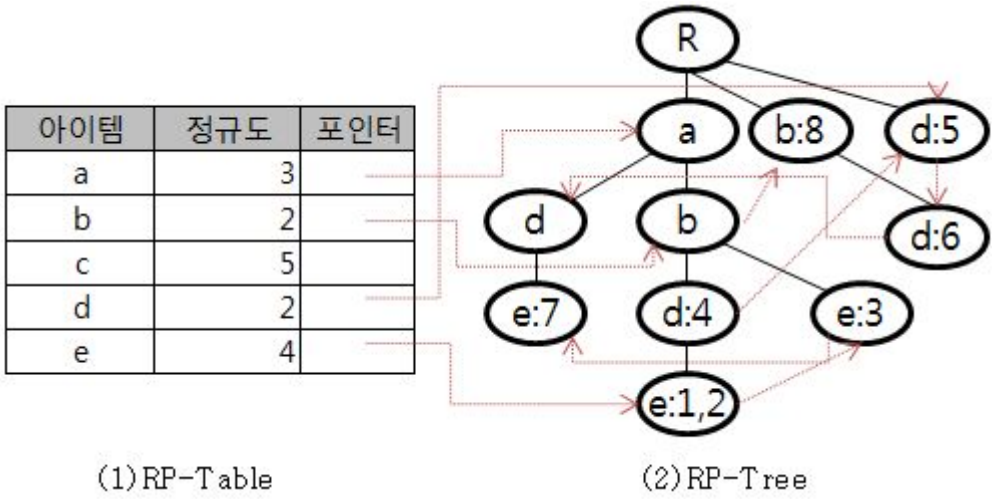


그림 3. RP-Table 과 RP-Tree

정규 패턴 마이닝은 먼저 트랜잭션 데이터베이스를 스캔하고 각 아이템의 빈발도 대신에 아이템이 등장한 주기를 검사한다. 아이템의 주기는 첫 번째 등장한 트랜잭션 번호와 아이템이 등장한 트랜잭션 간의 차, 트랜잭션의 총 개수 - 최종적으로 등장한 트랜잭션 번호 중 가장 큰 수가 된다. 예를 들어 만약 트랜잭션의

총 개수가 10이고 아이템 a가 트랜잭션 번호 {1, 3, 5}에서 등장했다고 가정한다면 이 아이템 a의 주기는 (1 - 0 = 1), (3 - 1 = 2), (5 - 3 = 2), (10 - 5 = 5)중에 가장 큰 수인 5가 된다. 그리고 최대 임계 빈발도 보다 더 큰 주기를 가지는 아이템을 제외하고 나머지 아이템을 지정된 순서에 따라서 정렬하고 아이템 이름, 주기로 구성된 RP-Table을 생성한다. 그리고 다시 한 번 데이터베이스를 스캔하며 각 트랜잭션의 아이템들 중 정규 항목이 아닌 것을 제거하고 남은 아이템들을 테이블의 순서대로 정렬하고 FP-Tree와 유사한 방법으로 트리를 구성한다. 이 때 노드에는 아이템 이름만을 기록하되 마지막 아이템은 그 트랜잭션 번호를 함께 기록한다. 그림 3은 표 1의 데이터베이스를 정규도 임계치를 4로 가정하고 RP-Tree로 구성한 결과이다. RP-Tree 생성이 끝나면 정규 패턴을 마이닝하기 위해 RP-Table의 가장 아래에 있는 아이템부터 상향식으로 조건부 트리를 생성한다. 생성한 조건부 트리로부터 다시 한 번 각 아이템의 정규도를 구해고 최대 임계 빈발도보다 낮은 아이템을 조합하여 정규 패턴을 마이닝 한다. 그러나 정규 패턴 마이닝은 특정 부분에서만 나타나는 아이템을 마이닝 하지 못하므로 정규도에 의한 레어 아이템 문제가 발생한다.

III. 제안하는 마이닝 기법

본 장에서 정규 패턴을 마이닝 하기 위한 데이터 구조 RMIS-Tree와 RMIS-Tree로부터 정규 패턴을 마이닝 하는 기법 RMIS-Growth을 제안한다. 본 장의 예제는 설명을 위해 임의로 만들어졌으며 RMIS-Tree의 구축에 대해서 설명하고, RMIS-Growth가 증가하는 데이터베이스를 처리하고 마이닝하는 기법에 대해서 자세히 설명한다.

3.1 RMIS-Tree 구성

최초의 RMIS-Tree 의 구성은 트랜잭션 데이터베이스를 한 번 스캔하여 구성된다. 데이터베이스의 각 트랜잭션의 아이템들은 RMIS-Tree에 노드로 삽입되고 헤더 테이블에 반영된다. 이후에 데이터베이스에 증가된 트랜잭션이 발생한다면 전체 데이터베이스를 다시 스캔하지 않고 추가된 트랜잭션만 스캔을 수행하게 된다. RMIS-Tree는 데이터베이스를 경량화하여 트리 구조로 표현하고 정규 패턴 마이닝에 사용된다.

1) 헤더 테이블

헤더 테이블은 데이터베이스 내에 등장한 아이템들에 대한 정보를 저장하고 정규 패턴 마이닝에 중요한 역할을 한다. 헤더 테이블은 각 아이템의 이름, MIS 값, 빈발도, 정규도, 최종적으로 등장한 트랜잭션 번호, 마이닝 대상 여부, 그 아이템이 처음 트리 내에서 나타난 노드의 포인터로 구성된다. MIS 값은 아이템별 최소 빈발 임계치를 저장하고 빈발도는 트랜잭션 데이터베이스에서 발생한 빈발도를 저장한다. 정규도는 해당 아이템이 트랜잭션 데이터베이스에서 등장하는 최대 주기를 저장하게 되고 최종 TID는 해당 아이템이 등장한 최종 트랜잭션

의 번호를 저장한다. 마이닝 여부는 마이닝을 수행할 시에 해당 아이템의 마이닝 여부를 저장하고 포인터는 각 아이템이 트리에서 첫 등장한 노드를 지정한다. 헤더 테이블의 각 행은 MIS 값의 내림차순으로 정렬되고, 이 순서와 동일하게 모든 트랜잭션의 아이템도 정렬한다.

표 3. 헤더 테이블

ITEM	MIS	S(빈발도)	R(정규도)	L(최종TID)	M(마이닝)	P(포인터)
I_1	4	5	2	7	T	
I_2	3	1	4	5	F	
I_3	2	2	3	4	T	

2) RMIS-Tree의 구성 요소

그림 4와 같이 RMIS-Tree는 노드들과 링크들로 이루어진다. 각 노드 N은 아이템 이름이 표시되고, 만약 트랜잭션의 마지막 아이템이면 트랜잭션 번호가 함께 표시된다. 각 노드의 링크는 마이닝 과정에서 빠르게 트리를 탐색하기 위해 필요한 요소로 다른 노드의 포인터이다. 이 링크들은 노드 N의 부모 노드 포인터, 자식 노드들의 포인터, 트리 내에서 같은 이름을 가진 다음 노드의 포인터로 구성한다.

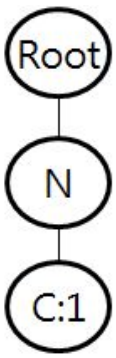


그림 4. RMIS-Tree

3.2 RMIS-Tree 구축

RMIS-Tree는 1회의 데이터베이스 스캔 후 생성된다. 표 4와 표 5는 트랜잭션 번호 TID와 아이템 집합으로 구성된 데이터베이스와 MIS 값의 한 예이다.

표 4. 트랜잭션 데이터베이스

T-ID	Transaction	T-ID	Transaction
1	A, B, E	5	C, D ,E
2	B, C, D	6	A, C, E
3	A, D	7	B, E, F
4	C, D, E	8	A, B, C, E

표 5 아이템별 MIS 값

아아템	A	B	C	D	E	F
MIS	3	5	2	4	4	4

위 트랜잭션 데이터를 경량화하여 RMIS-Tree를 생성하는 알고리즘을 소개한다. 먼저 최상위 루트 노드 R를 생성하고 이 노드를 null 또는 R로 표기한다. 그리고 전체 데이터베이스를 스캔하고 각 트랜잭션의 아이템을 헤더 테이블의 아이템 순으로 정렬한다. 정렬된 아이템을 차례대로 헤더 테이블에서 그 아이템을 찾아 빈발도를 1 증가시키고 새로운 주기를 계산하여 헤더 테이블에 적용한다. 아이템의 주기는 처음 등장한 트랜잭션 번호, 등장한 트랜잭션 간의 차, 데이터베이스의 전체 트랜잭션의 수와 마지막으로 등장한 트랜잭션의 차중 가장 큰 값이 그 아이템의 주기가 된다. 그리고 현재 트랜잭션 번호를 그 아이템의 마지막으로 나타난 트랜잭션의 번호에 기록한다. 헤더 테이블에 각 아이템 정보의 갱신이 완료되면 정렬한 아이템을 Tree에 삽입한다. 노드 삽입은 root 노드로부터 시작하여 각 아이템을 정렬된 순서대로 링크로 연결한다. 먼저 만약 현재 노드의 자식 노드들 중에 아이템 I의 이름과 같은 이름의 노드가 있는지를 확인하고, 없다면 새로운 노드를 생성하여 그 노드를 현재 노드로 한 뒤 다음 아이템으로 넘

어간다. 반면에 자식 노드들 중에 아이템 I와 같은 이름의 노드가 있다면, 그 자식 노드가 현재 노드가 되고 다음 아이템으로 넘어간다. 그리고 정렬된 아이템들 중에 마지막 아이템은 그 트랜잭션의 번호를 “노드 이름:트랜잭션 번호” 와 같이 함께 기록하고 포인터를 추가한다. 포인터는 마이닝 시에 전체 트리를 검색하지 않고 다음 경로를 찾아가기 위해 반드시 필요하다. 헤더 테이블의 아이템 I의 포인터는 그 아이템 I가 트리 상에서 처음 나온 항목의 포인터이며, 다시 그 아이템 I의 노드의 포인터는 아이템 I가 다음에 나타난 노드의 포인터이다. 이 포인터가 null이면 더 이상 트리에는 그 아이템 I로 끝나는 경로가 없음을 뜻한다.

Algorithm 1 - RMIS-Tree 생성

Input : Transaction Database DB

Output : RMIS-Tree, Header Table

```
Create root of a RMIS-Tree T, and label it as "null"
foreach Transaction T in DB
    sort all items in T according to order of header table
    foreach Item I in T
        increase support of I by 1 in header table
        set regularity of I by call caculateReg(reg, lastTransaction num,
                                                current Transaction num
        set last transaction num by current transaction num
        let the sorted items in T be [p|P], where p is the first element
        and P is the remaining list.
        call insert_tree(Tree, I)
    end foreach
end foreach
```

Procedure cacluateReg(regulatiry, last TID num, current Tid Num

---

```
if last transaction num = 0 then
    regularity = current Transaction num
```

```
else if (last transaction num - current Transaction num) > regularity
    regularity = last transaction num - current transaction num
return  regularity
```

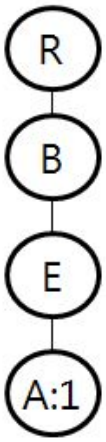
```
Procedure insert_tree([p|P], T)
  while p.size() > 0
    set current node = T
    if current node has child node C that C.name = p.name then
      current node = C
    else
      create node N and named p
      add N to child node of C
    end if
    if p is last item then
      add transaction num to N
      if pointer of N in header table then
        add pointer of N to header table
      else
        set pointer of node P = pointer of N in header table
        while P != nulll
          set P = P.nextnode
        end while
        set P.nextnode = pointer of N
      end if
    end if
  end while
```

Example 1.

표 4의 데이터베이스를 이용하여 RMIS-Tree를 구축하는 예를 설명한다.

먼저 TID 1의 아이템 {A, B, E}는 MIS 값의 내림차순으로 정렬된 헤더 테이블의 순서대로 {B, E, A}로 정렬된다. 그리고 헤더 테이블의 아이템 B, E, A의 주기는 현재 트랜잭션 번호 1 - 0 = 1이 되고 각 아이템의 빈발도는 1씩 증가하여 1이 되며 최종 등장 트랜잭션 번호는 현재 트랜잭션 번호인 1이 된다. 초기 상태의 Tree는 아무것도 삽입되지 않았으므로 root 노드만이 존재한다. 그러므로 정렬된 아이템 집합 {B, E, A}는 루트 노드로부터 차례대로 새로운 노드가 생성이 되고 링크가 연결될 것이다. TID 1번 트랜잭션이 트리에 추가된 결과는 root 노드의 자식 노드로 B가 추가되고 B의 자식 노드로 E, E의 자식 노드로 A가 추가되고, A에는 현재 트랜잭션 번호인 1번이 함께 표시된다. 그리고 아이템 A의 포인터를 추가한다. 포인터의 추가는 헤더 테이블로 시작해서 그 포인터가 null 이 나올 때까지 계속해서 따라가고 그 null인 포인터에 그 노드의 포인터를 저장함으로써, 헤더 테이블의 포인터를 따라가면 그 아이템의 경로를 모두 찾을 수 있다. 현재 헤더 테이블의 항목 A의 포인터가 null이므로 헤더 테이블의 A의 첫 노드 포인터가 저장된다. 다음으로 두 번째 트랜잭션 {B, C, D}는 아이템들의 정렬 순서에 따라 {B, D, C}로 정렬된다. 이 아이템들의 빈발도를 헤더 테이블에서 1씩 증가시킨다. 그리고 B의 주기는 현재 트랜잭션 - 마지막 트랜잭션 = 1로 현재 주기 값보다 크지 않으므로 변하지 않게 되고 D와 C는 현재 트랜잭션 - 헤더 테이블의 최종 등장 트랜잭션 번호의 결과인 2로 설정된다. 그리고 이 아이템들을 Tree에 추가한다. 이 아이템들 중 첫 아이템인 B가 root 노드의 자식 노드 중에 같은 이름의 노드가 있는지 확인하고 root의 자식 노드 중 이름이 B인 노드가 존재하므로 현재 노드를 B로 설정하고 다음 아이템 D가 B노드의 자식 노드들 중에 존재하는지 확인한다. D 노드는 B의 자식 노드 중에 존재하지 않으므로 D 노드를 생성하고 D를 현재 노드인 B의 자식 노드로 추가하고 새로 추가된 노드들은 자식 노드들이 없으므로 그다음 아이템들은 모두 새로 생성하여 자식 노드로 추가하고 링크로 연결하게 된다. 그리고 마지막 아이템 C는 현재 트랜잭션 번호인 2를 함께 표시하고 포인터를 추가한다. 그림 5 는 현재까지의 트리 생성 과정을 보이고 있다.

ITEM	MIS	S	R	L
B	5	1	1	1
D	4	0		0
E	4	1	1	1
F	4	0		0
A	3	1	1	1
C	2	0		0



ITEM	MIS	S	R	L
B	5	2	1	2
D	4	1	2	2
E	4	1	1	1
F	4	0		0
A	3	1	1	1
C	2	1	2	2

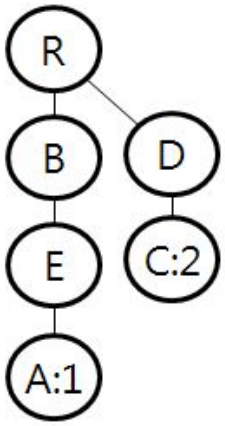


그림 5. 트랜잭션 1번과 2번을 트리에 삽입한 결과

이와 같은 과정을 모든 트랜잭션에 수행한다. 그림 6은 모든 데이터베이스를 스캔하고 트리를 완성시킨 결과이다.

ITEM	MIS	S	R	L
B	5	4	5	8
D	4	4	4	5
E	4	6	4	8
F	4	1	3	7
A	3	4	3	8
C	2	5	2	8

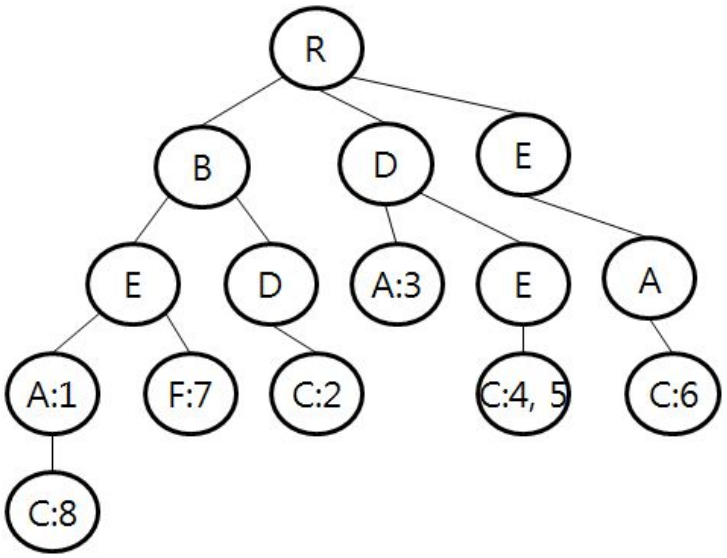


그림 6. 완성된 RMIS-Tree

3.3 RMIS-Tree의 증가된 트랜잭션 삽입

RMIS-Tree는 증가된 트랜잭션을 삽입하기 위해 데이터베이스를 새로 스캔하지 않고 간단한 절차로 Tree에 삽입한다.

표 6. 증가된 트랜잭션

T-ID	Transaction	T-ID	Transaction
9	B, D	11	A, B, D
10	A, B, F	12	E, F

표 6은 증가된 트랜잭션이 있는 데이터베이스의 예이다. RMIS-Tree는 노드 가지치기(Pruning)과 합병 작업을 하지 않으며 헤더 테이블에 각 아이템이 등장한 최종 트랜잭션 번호가 저장되어 있다. 따라서 추가된 트랜잭션은 최초 트리 구성 방법과 동일한 방법으로 증가된 트랜잭션들을 삽입한다. 그림 7은 추가된 트랜잭션이 삽입된 트리의 모습과 헤더 테이블이다.

ITEM	MIS	S	R	L
B	5	7	5	11
D	4	6	4	7
E	4	7	4	8
F	4	3	3	7
A	3	6	3	11
C	2	5	4	8

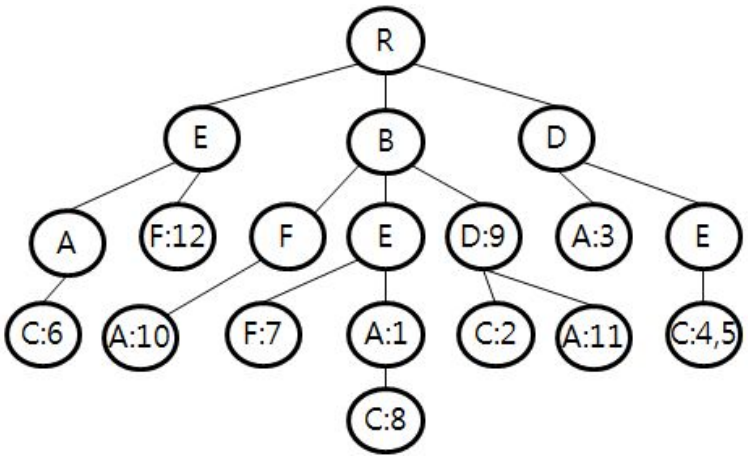


그림 7. 증가된 트랜잭션이 반영된 RMIS-Tree

3.4 정규 패턴 마이닝

트리 생성이 완료되면 이 트리로부터 정규 패턴을 마이닝 한다. RMIS-Tree는 추가적인 트랜잭션을 반영하기 위해 최초 트리를 구축시에는 노드 가지치기와 합병 작업을 하지 않는다. 그러나 마이닝을 수행하기 전에 빈발한 정규 패턴을 만들지 못하는 항목을 골라내고 그 결과를 헤더 테이블에 마이닝 대상인지를 저장한다. 그림 6의 헤더 테이블에서 최대 주기를 4라고 가정하고, 최대 주기 4보다 높은 아이템 B와 MIS 값 보다 낮은 빈발도를 보인 F를 마이닝 대상에서 제외한다.

표 7. 마이닝 여부 결정 후의 헤더 테이블

ITEM	MIS	S	R	L	M
B	5	7	5	11	F
D	4	6	4	7	T
E	4	7	4	8	T
F	4	3	3	7	F
A	3	6	3	11	T
C	2	5	4	8	T

마이닝 대상이 결정이 되면 헤더 테이블의 맨 아래 있는 항목 C부터 상향식으로 빈발한 정규 패턴을 마이닝 한다. 마이닝 과정은 먼저 헤더 테이블에 있는 아이템 I의 포인터를 따라서 트리 내에서 아이템 I가 포함된 경로를 찾은 후, 그 경로를 이용하여 그 아이템 I의 조건부 패턴 베이스(Conditional Pattern Base)를 생성하고, 이 조건부 패턴 베이스로부터 조건부 트리(Conditional Tree)를 생성한 뒤 조건부 트리로부터 모든 생성 가능한 패턴을 만든다.

Algorithm 2

Input : RMIS-Tree, Item I

Output : the complete set of all I's conditional pattern bases and complete

```
        set of all frequency regular patterns
foreach Item I in Header Table
    Genarate pattern β = i ∪ α with support = i.support And reg = i.reg
    Generate β's conditional pattern base and β's conditional tree T
    if T != null then
        call RMIS-Growth(T, β)
    end if
end foreach
```

Procedure RMIS-Growth

```
    foreach Item I in header table
        Genarate pattern β= i ∪ α with support = i.support And reg = i.reg
        Generate β's conditional pattern base and β's conditional tree T
        if T != null then
            if T is single path P then
                foreach combination of the Item in the P
                    generate pattern with regularity and support
                end foreach
            end if
        else
            call RMIS-Growth(T, β)
        end if
  end foreach
```

아이템 C가 포함된 경로는 {E, A, C:6}, {E, A, C:8}, {D, C:2}, {E, F, C:4}, {D, E, C:5}로 그림 8의 좌측의 와 같게 된다. 이 집합중 8번 트랜잭션 {B, E, A, C}는 아이템 B는 마이닝 대상이 아니므로 제외되었다. 이 집합들은 아이템 C의 경로이므로 아이템 C를 반드시 포함하게 되고 헤더 테이블의 순서에 따라 정렬

되므로 이 집합들의 마지막 아이템은 반드시 아이템 C가 된다. 이 경로들을 다시 트리로 만들면 이 트리를 C의 조건부 패턴 베이스(Conditional Pattern Base)가 된다.

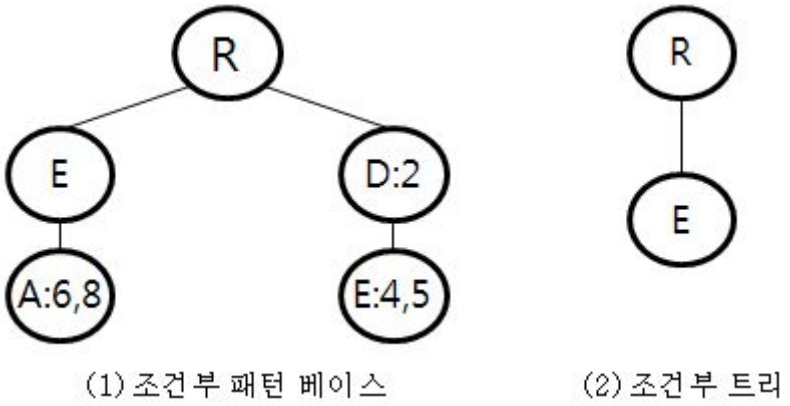


그림 8. 아이템 C의 조건부 패턴 베이스와 조건부 트리

C의 조건부 패턴 베이스가 생성이 되면 C의 조건부 트리(Conditional Tree)를 생성하기 위해서 생성된 C의 조건부 패턴 베이스로부터 각 항목의 주기 값을 다시 계산한다. 아이템 A는 트랜잭션 번호 6, 8번에서 나타났으므로 주기는 6, 아이템 D는 2번 트랜잭션에서만 나타났으므로 10, 아이템 F는 4번 트랜잭션에서만 나타났으므로 8, 아이템 E는 4, 5, 6, 8번 트랜잭션에서 나타났으므로 주기는 4가 된다. 따라서 아이템 C의 조건부 트리는 그림 8과 같이 생성된다. 그리고 이 조건부 트리는 단일 경로이므로 이 경로에서 생성 가능한 모든 패턴을 생성하고 “아이템 이름:빈발도, 최대 주기”로 표현하면 {EC:4,4}, {C:5,4} 의 패턴이 마이닝 된다. 이 과정을 헤더 테이블의 아이템들 중 패턴을 만들 수 있는 모든 아이템에 대해서 수행하고 모든 패턴을 마이닝 한다.

Ⅳ. 성능 평가

4.1 실험 환경 및 실험 데이터

본 장에서는 제안한 알고리즘인 RMIS-Growth의 성능을 평가하기 위해 비교 알고리즘들과의 다양한 실험을 수행한다. 다중 빈발도 임계치 기반의 정규 패턴을 마이닝 하기 위한 알고리즘이 존재하지 않으므로 정규 패턴 마이닝 알고리즘 [8]과 성능을 비교한다. 모든 성능 평가 실험은 펜티엄 코어II 듀오 2.8GHz, 2GB 메모리, 윈도우 XP 서비스 팩 III 환경에서 수행되었다. 본 논문에서 제안된 알고리즘인 RMIS-Growth는 Java JDK 1.6에서 구현되었다. 성능 평가 실험을 위한 일반적인 설정은 다음과 같다. 먼저, 데이터베이스에 대한 스캔을 통해 RMIS-Tree 그리고 RP-Tree를 구축한다. 다음에, 구축된 RMIS-Tree 내 노드들을 MIS 내림차순으로 정렬함으로써 트리를 재구축한다. 마지막으로, 레어 패턴들을 포함한 모든 정규 패턴들을 마이닝 한다. 표 8은 성능 평가 실험에서 사용되는 실제 데이터 셋들의 특성을 보여준다. 실험에 사용한 데이터셋 Chess, Mushroom, T10I4D100K, Retail 데이터 셋들은 기존 연구 [1-8]에서 많이 활용되고 있는 일반적인 데이터셋이다. T10I4D100K는 가상 데이터셋 생성기 [1]를 통해 생성했으며 나머지 데이터셋은 FIMI 저장소로부터 획득하였다. Chess 그리고 Mushroom은 밀집(dense) 데이터 셋들이고, T10I4D100K, Retail은 비밀집(sparse) 데이터 셋이다. 즉, Chess 그리고 Mushroom은 비슷한 형태의 긴 트랜잭션들로 구성돼 있으며, T10I4D100K, Retail은 길이가 짧고 서로 비슷하지 않은 형태의 트랜잭션들로 이뤄져 있다. Chess 데이터 셋은 체스 게임에서 말의 움직임에 대한 데이터로서 평균 트랜잭션 길이가 길며, 세 개의 데이터 셋들 중에서 가장 적은 트랜잭션 그리고 아이템으로 이루어져 있다. 또 다른 밀집 데이터 셋인 Mushroom은 23개의 버섯 품종에 대한 데이터로서 일반적인 밀집 데이터 셋들과 마찬가지로 평균 트랜잭션 길이가 길며, Chess 데이터 셋보다 더 많은 트랜잭션 그리고 아이템을 가지고 있다. 즉, Chess 데이터 셋보다 더 큰 크기를 가

지고 있는 밀집 데이터 셋이다. 마지막으로, T10I4D100K, Retail 데이터 셋은 Chess 그리고 Mushroom 데이터 셋들과는 달리 평균 트랜잭션 길이가 짧은 비밀집 데이터 셋으로서 두 실제 밀집 데이터 셋들보다 훨씬 많은 트랜잭션을 포함하고 있다. 다시 말해서, 데이터베이스 크기가 가장 큰 데이터 셋이다. 또한, 아이템 역시 더 많이 포함하고 있으며, 평균 트랜잭션 길이가 짧으므로 각 트랜잭션의 형태는 비슷하지 않은 형태를 가지고 있다.

표 8. 데이터셋

데이터셋	트랜잭션수	평균 트랜잭션 길이	개별 아이템 수	크기(MB)
Chess	3196	37	75	0.34
Mushroom	8124	23	119	0.56
T10I4D100K	100,000	10	1,000	3.92
Retail	88,162	10.3	16,470	4

본 논문에서 제안된 알고리즘인 RMIS-Growth는 다중 최소 빈발도 임계치를 기반으로 정규 패턴을 마이닝 한다. 본 장의 성능 평가 실험에서 사용되는 실제 데이터 셋들인 Chess, Mushroom, T10I4D100K, Retail은 다중 최소 빈발도 임계치 정보를 포함하지 않는다. 따라서 이전의 다중 최소 빈발도 임계치 기반의 알고리즘 논문들 [3, 4]에서 성능 평가 실험을 위해 사용한 아래의 공식을 사용하여 각 아이템의 최소 아이템 빈발도 임계치를 생성한다. 아래의 공식에서, 임의의 아이템 a_i의 최소 아이템 빈발도 임계치는 $f(a_i)$를 그 아이템의 데이터베이스 내 빈도수라고 할 때 그 빈도수에 σ를 곱한 값으로 계산한다. 만약, 계산된 아이템의 a_i의 최소 아이템 빈발도 임계치 $M(a_i)$가 최소 임계치 MIN보다 작으면 $MIS(a_i)$는 MIN으로 설정된다. 즉, 각 아이템의 빈도수에 빈도수 반영 비율 σ를 곱한 값이 MIN보다 작으면 그 아이템의 최소 아이템 빈발도 임계치는 MIN, 그렇지 않으면 그 곱한 값이 설정된다. 다시 말해서, MIN 값은 각 아이템의 최소 아이템 빈발도 임계치가 너무 작은 값으로 설정되지 않도록 제어하기 위한 값이다. 예를 들어, 아이템들 A 그리고 B의 데이터베이스 내 빈도수가 각각 20 그리

고 10이고, σ와 MIN이 각각 0.1 그리고 0.01이라고 하면, 200개의 트랜잭션을 가지고 있는 데이터베이스에서는 MIN 값이 2가 된다. 또한, A 그리고 B의 M 값은 각각 20 × 0.1 = 2 그리고 10 × 0.1 = 1이다. 따라서 MIN 값보다 큰 M 값을 가지는 아이템 A의 최소 아이템 빈발도인 MIS(A)는 2이고, 반대인 M 값을 가지는 아이템 B의 최소 아이템 빈발도 MIS(B)는 2로 설정된다.

$$\mathrm{MIS}(a_i) = \begin{cases} \mathrm{M}(a_i) & \mathrm{M}(a_i) > MIN \\ MIN & \text{otherwise} \end{cases}$$
$$\mathrm{M}(a_i) = \sigma \times \mathrm{f}(a_i)$$

또한, 전통적인 일반 빈발 패턴 마이닝, 즉, 단일 최소 빈발도 임계치 기반의 모델에서는 하나의 임계치 값을 사용한다. 만약 위 공식에서 σ에 의해 모든 아이템들이 MIN 값을 최소 아이템 임계치로 설정되면 다중 최소 빈발도 임계치 기반의 알고리즘들도 단일 모델과 동일하다. 반대로 σ이 각 아이템들이 서로 다른 최소 아이템 임계치를 가지도록 설정되면, 단일 모델과는 다른 다중 최소 빈발도 임계치 기반으로 동작한다. 따라서 σ은 아이템들의 최소 아이템 빈발도 임계치 값을 제어하기 위한 목적으로 사용된다.

한편, 정규 패턴을 마이닝 하기 위한 알고리즘들은 최대 정규도 임계치 λ를 사용하며, 이는 데이터베이스 내 트랜잭션과 정규도의 곱으로 계산된다. 예를 들어, Chess 의 정규도 0.1%의 λ 는 0.001에 Chess 의 전체 트랜잭션 수인 3196을 곱한 결과인 3이 된다.

제안한 알고리즘인 RMIS-Growth의 성능을 평가하기 위해 트리 구성 시간, 마이닝 소요 시간, 발견된 유효 패턴 수의 관점에서 실험을 수행한다. 트리를 구성하는 데 걸린 시간과 유효 패턴들을 마이닝 하는 데 소요된 시간은 제안한 알고리즘의 효율성을 보여주며, 발견된 패턴의 개수는 얼마나 효과적으로 중요하지 않은 패턴들을 제거하여 유효 패턴을 마이닝 하는 것을 보여준다. 따라서 트리를 구성하는 시간과 마이닝을 수행하는 데 걸린 시간은 빠를수록 효율적인 알고리즘이며, 더 많은 불필요한 패턴이 제거되어 적은 수의 패턴을 마이닝 할수록 좋

다. 그 이유는 마이닝 되는 패턴이 너무 많으면 많을수록 의사 결정을 하는 데 필요한 중요한 정보를 얻기 위한 분석이 어려워지기 때문이다.

4.2 성능 평가 실험

4.2.1 Chess 데이터 셋에 대한 성능 평가 실험 결과

본 부분에서 밀집 데이터 셋인 Chess에 대한 성능 평가 실험을 수행한다. 성능 평가는 위에서 언급한 것과 같이 트리를 구성하는 데 걸린 시간, 유효 패턴을 마이닝 하기 위해 걸린 시간, 그리고 마이닝 된 패턴의 개수에 대하여 수행된다. 실험을 위해 σ는 0.1%로 설정되었다.

먼저 비교 알고리즘들이 Chess 데이터 셋에 대한 스캔을 통해 트리를 구축하는 데 사용한 시간에 대한 성능 평가 결과를 그림 9에서 보인다. 그림 9에서, 제안한 RMIS-Growth 알고리즘이 비교 알고리즘인 비교 RP-Tree보다 더 빠른 속도로 트리를 구축한 것을 알 수 있다. RMIS-Growth는 데이터베이스를 1번만 스캔하기 때문에 트리 구성에서 빠른 속도를 보였으며 λ에 큰 영향을 보이지 않는 거의 일정한 트리 구성 속도를 보인다.

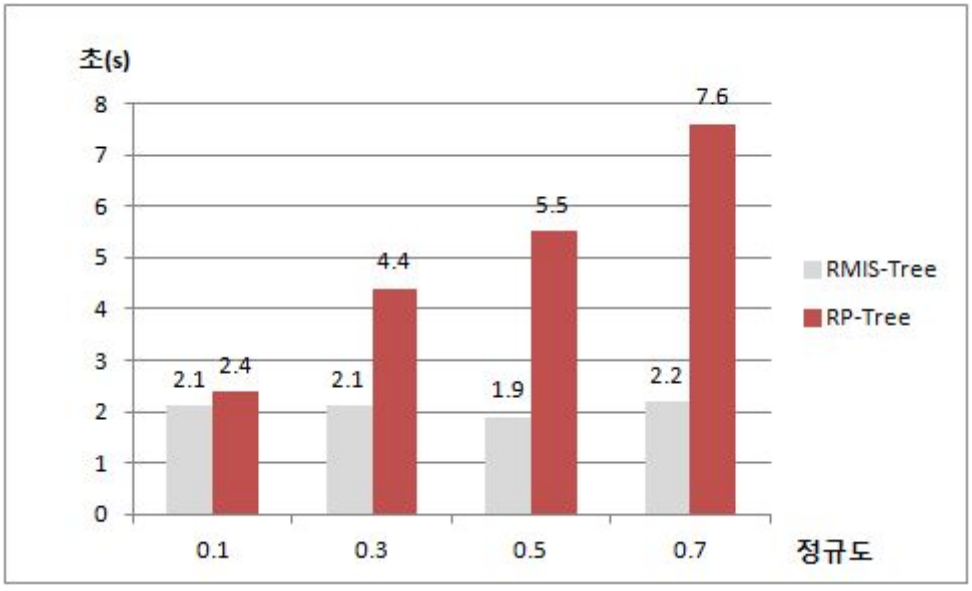


그림 9. Chess 트리 구성 소요 시간(초)

다음으로 그림 10에서 구축된 RMIS-Tree 그리고 RP-Tree에서 각 비교 알고리즘이 Chess 데이터 셋에서 마이닝 한 패턴의 개수에 대한 결과를 보인다.

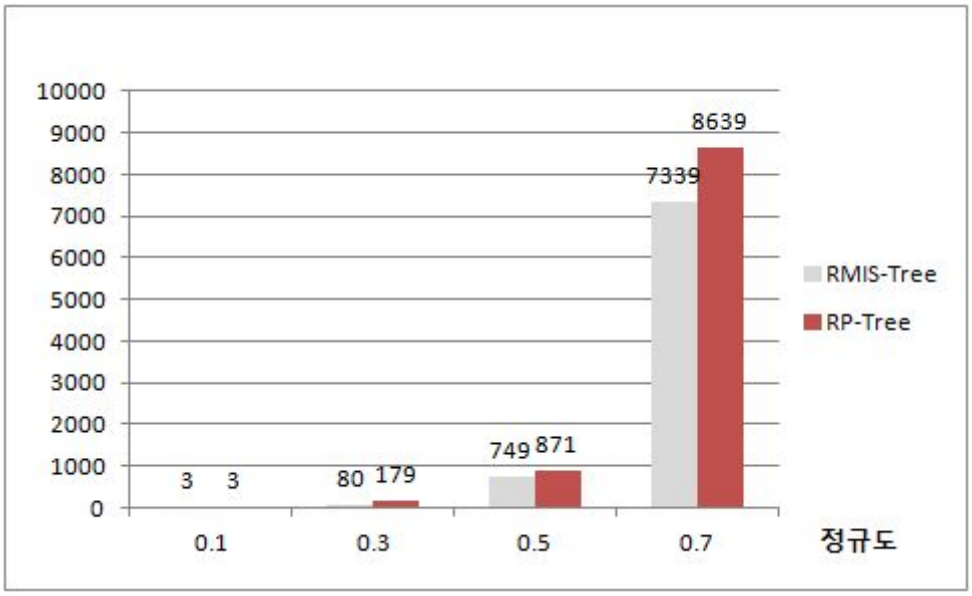


그림 10. Chess 마이닝 패턴 수

4.2.2 Mushroom 데이터 셋에 대한 성능 평가 실험 결과

본 부분에서 또 다른 밀집 데이터 셋인 Mushroom에 대한 비교 알고리즘들의 성능을 평가하기 위한 실험을 수행한다. 실험은 Chess 데이터 셋과 마찬가지로 비교 알고리즘들이 트리를 구축하는 데 소요한 시간, 구축된 트리로부터 유효한 패턴을 마이닝 하는 데 필요한 실행 시간, 그리고 마이닝 결과로 추출된 패턴의 개수에 대하여 수행된다. 실험을 위해 σ는 0.1%로 설정하였다.

그림 11은 Mushroom 데이터 셋에 대한 스캔을 통해 비교 알고리즘들이 트리를 구축하는 데 사용한 실행 시간에 대한 실험 결과를 보여준다. RMIS-Tree는 Chess와 유사하게 정규도에 영향을 받지 않는 트리 구축 시간을 보였으며 RP-Tree 보다 빠른 결과를 보였다.

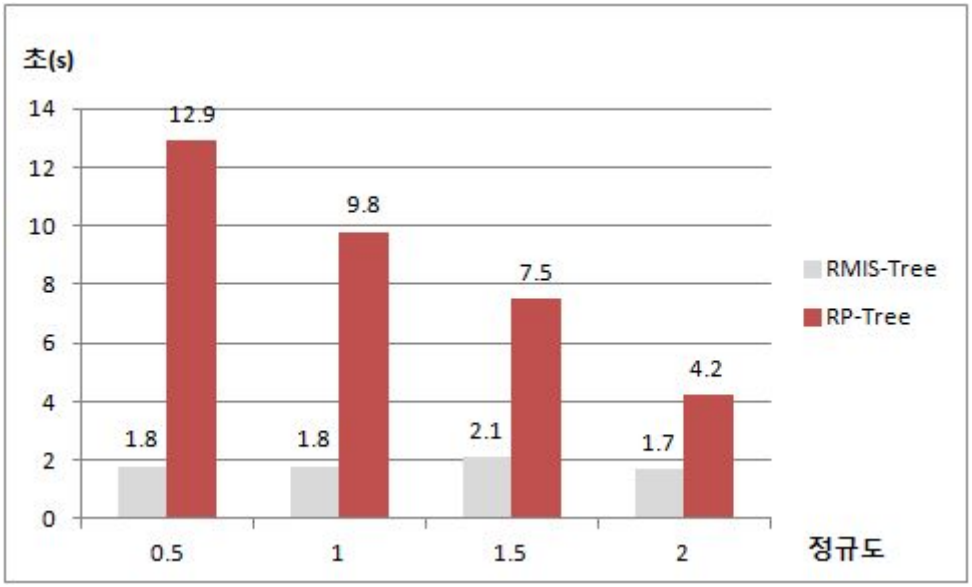


그림 11. Mushroom 트리 구성 소요 시간(초)

다음으로 그림 12는 구축된 트리에서 비교 알고리즘들이 유효한 패턴을 마이닝 하는 데 소요한 실행 시간에 대한 결과이다.

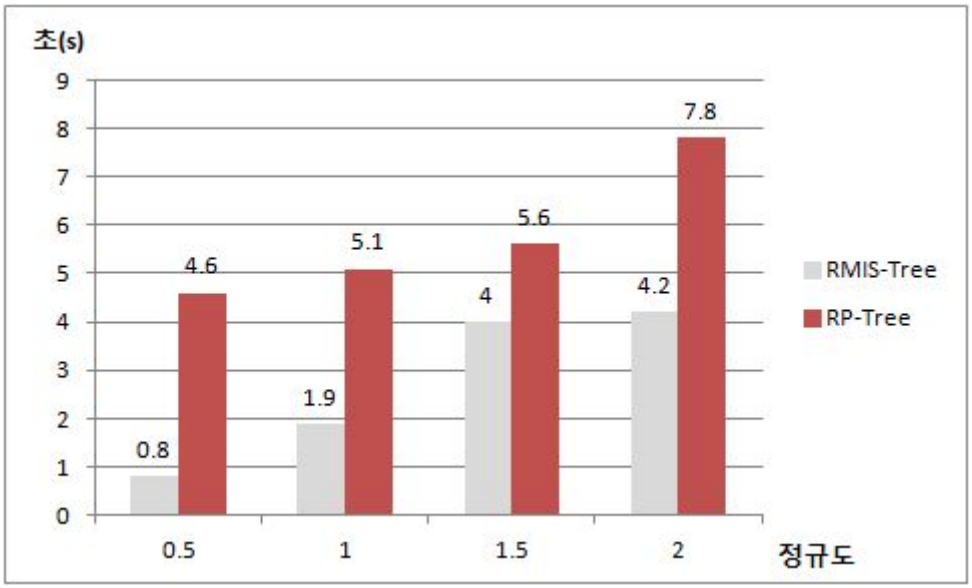


그림 12. Mushroom 마이닝 소요 시간(초)

마지막으로 그림 13에서 각 비교 알고리즘이 Chess 데이터 셋에서 발견한

유효한 패턴의 개수에 대한 결과를 보인다.

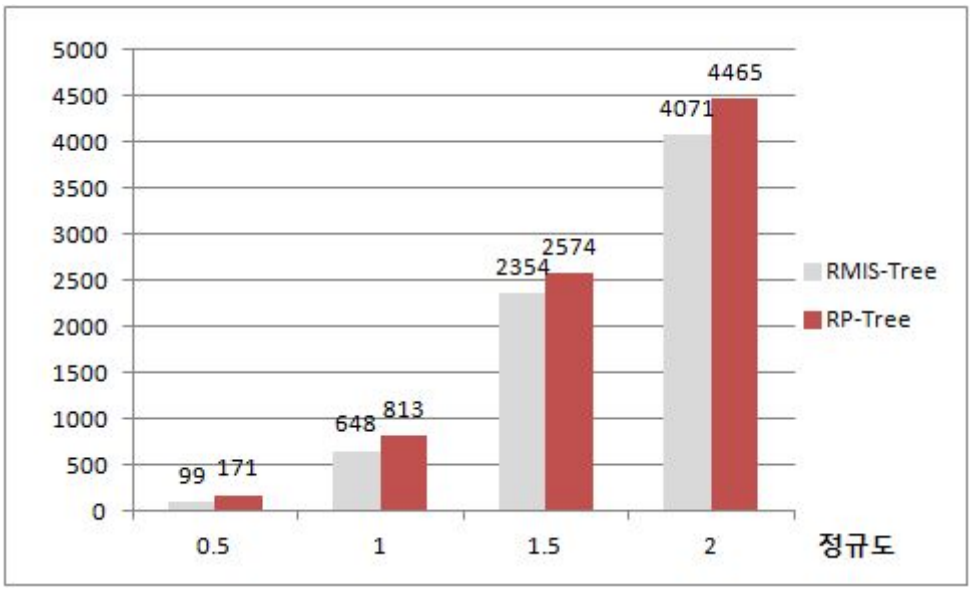


그림 13. Mushroom 마이닝 패턴 수

그림 13의 밀집 데이터 집합에 보인 결과에서 제안한 알고리즘이 트리 구축, 마이닝 수행, 유효 패턴 마이닝에 적절함을 볼 수 있다.

4.2.3 T10I4D100K 데이터 셋에 대한 성능 평가 실험 결과

본 부분에서 비밀집 데이터 셋인 T10I4D100K에 대한 성능 평가 실험을 수행한다. 위의 실제 밀집 데이터 셋들에 대한 실험과 마찬가지로 본 실험에서도 비교 알고리즘들이 트리를 구축하고 유효 패턴을 마이닝 하기 위해 사용한 각각의 실행 시간과 마이닝 결과로 추출된 패턴의 개수에 대한 실험을 수행한다. 실험을 위해 σ는 0.1%로 설정되었다.

먼저 비교 알고리즘들의 T10I4D100K 데이터 셋을 스캔하여 마이닝을 수행하기 위한 트리를 구축하는 데 걸린 시간에 대한 성능 평가 결과를 그림 14에서 보인다. 그 실험 결과에서, RMIS-Tree는 정규도에 거의 영향을 받지 않는 반면 RP-Tree는 정규도가 높아질수록 트리 구축에 소요되는 시간이 늘어나는 것이

보인다.

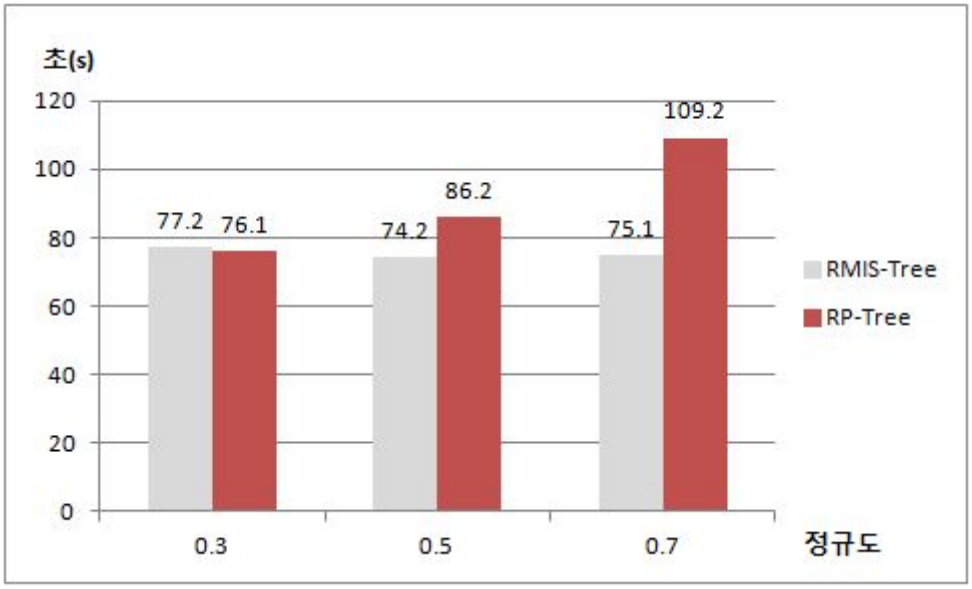


그림 14. T10I4D100K 트리 구성 소요 시간(초)

그림 15에서 구축된 RMIS-Tree 그리고 RP-Tree에서 패턴을 마이닝 하는 데 걸린 시간에 대한 실험 결과를 보인다.

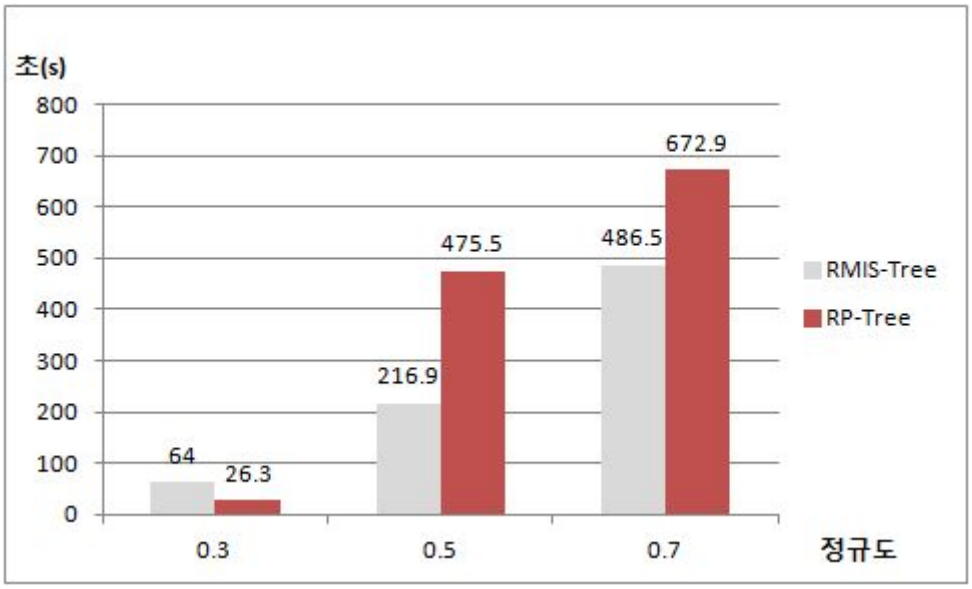


그림 15. T10I4D100K 마이닝 소요 시간(초)

적은 패턴이 발견되는 낮은 정규도에서 트리의 크기가 작으므로 RP-Tree가 더 빠른 마이닝 속도를 보였으나 높은 정규도가 될수록 RMIS-Tree가 더 빠른 속도를 보였다. 마지막으로 그림 16에서 각 비교 알고리즘이 T10I4D100K 데이터 셋에서 발견한 유효한 패턴의 개수에 대한 결과를 보인다.

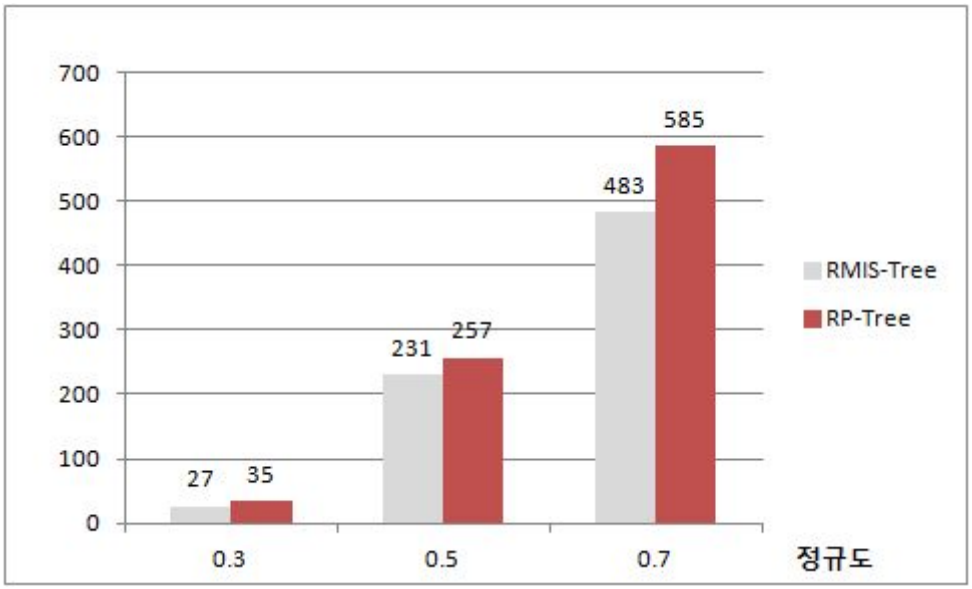


그림 16. T10I4D100K 마이닝 패턴 수

4.2.4 Retail 데이터 셋에 대한 성능 평가 실험 결과

본 부분에서 또 다른 비밀집 데이터 셋인 Retail에 대한 성능 평가 실험을 수행한다. 위의 실험과 마찬가지로 본 실험에서도 비교 알고리즘들이 트리를 구축하고 유효 패턴을 마이닝 하기 위해 사용한 각각 의 실행 시간과 마이닝 결과로 추출된 패턴의 개수에 대한 실험을 수행한다. 실험을 위해 σ는 0.1%로 설정되었다.

먼저 비교 알고리즘들의 Retail 데이터 셋을 스캔하여 마이닝을 수행하기 위한 트리를 구축하는 데 걸린 시간에 대한 성능 평가 결과를 그림 17에서 보인다.

실험 결과에서, RMIS-Tree는 비슷한 소요 시간을 보인 반면 RP-Tree는 소요 시간이 점차 증가하는 것이 보인다.

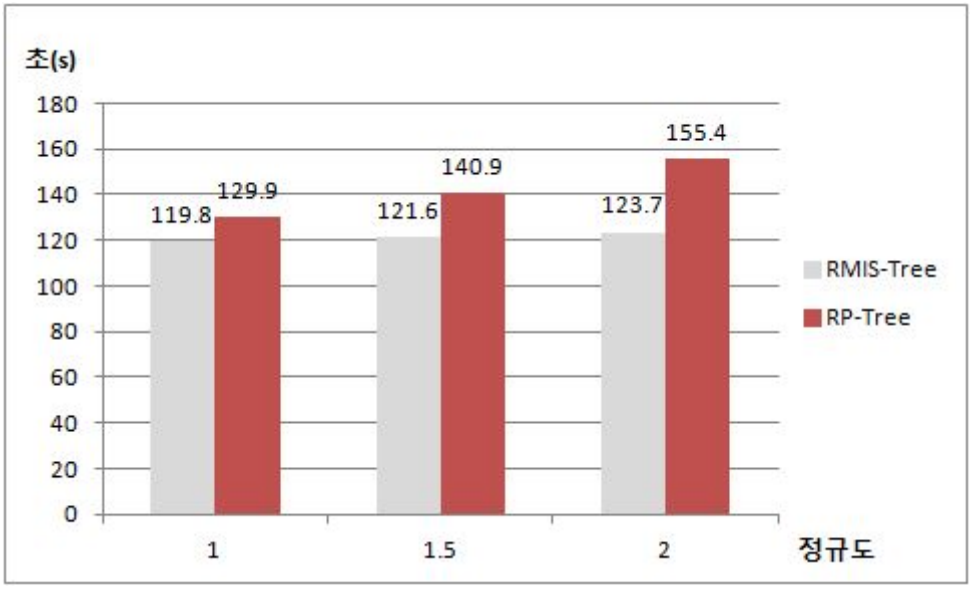


그림 17. Retail 트리 구성 소요 시간(초)

그림 18에서 구축된 RMIS-Tree 그리고 RP-Tree에서 유효한 패턴을 마이닝 하는 데 걸린 시간에 대한 실험 결과를 보인다.

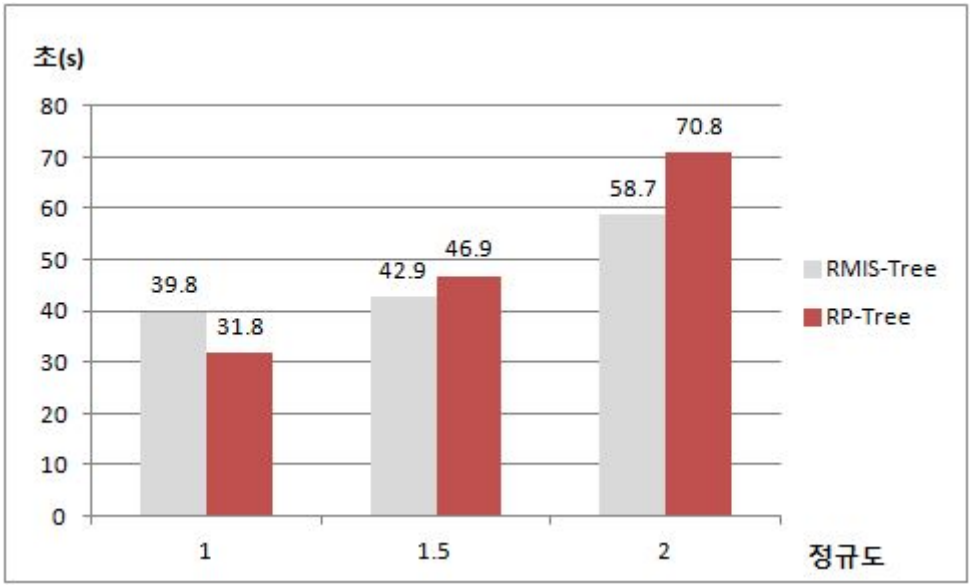


그림 18. Retail 마이닝 소요 시간(초)

RMIS-Tree는 적은 유효 패턴이 마이닝 되는 낮은 정규도에서는 더 많은 시

간이 마이닝에 소요되었으나 정규도가 높아지면서 더 효율적인 면을 보였다. 마지막으로 그림 19에서 각 비교 알고리즘이 Retail 데이터 셋에서 발견한 유효한 패턴의 개수에 대한 결과를 보인다.

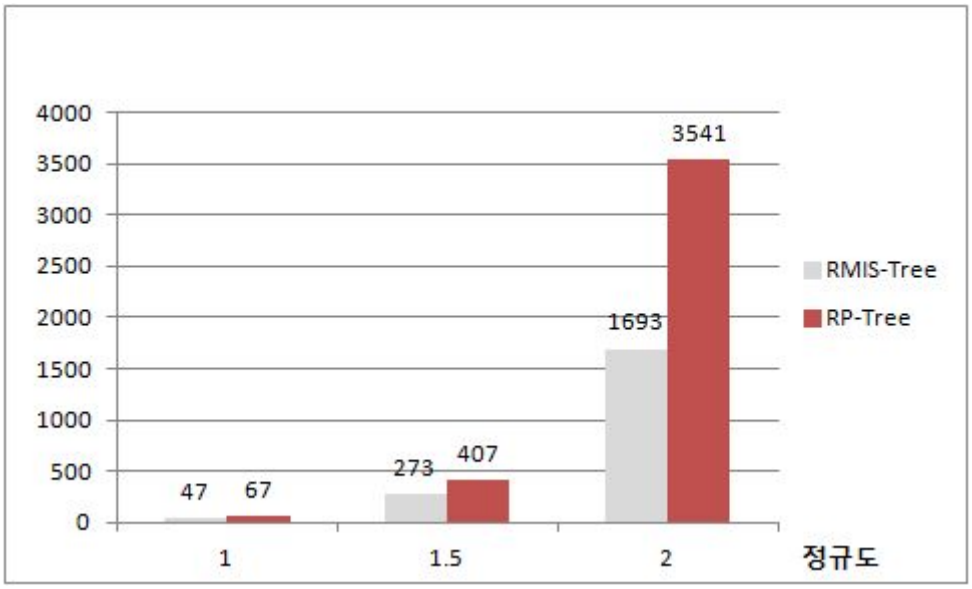


그림 19. Retail 마이닝 패턴 수

그림 20은 증가하는 T10I4D100K 데이터셋을 반영한 시험 결과이다. 데이터셋은 20%씩 증가하고 각 구간별로 트리 구축에 소요된 시간을 보인다.

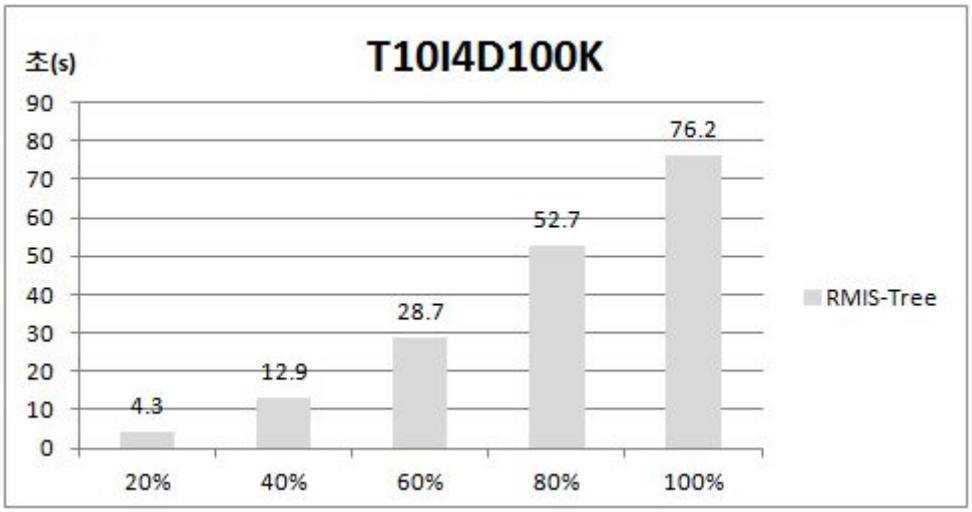


그림 20. 증가하는 T10I4D100K 데이터셋에 대한 트리 구성 시간

그림 21은 증가하는 T10I4D100K 데이터셋을 반영한 시험 결과이다. 데이터셋은 20%씩 증가하고 각 구간별로 트리 구축에 소요된 시간을 보인다.

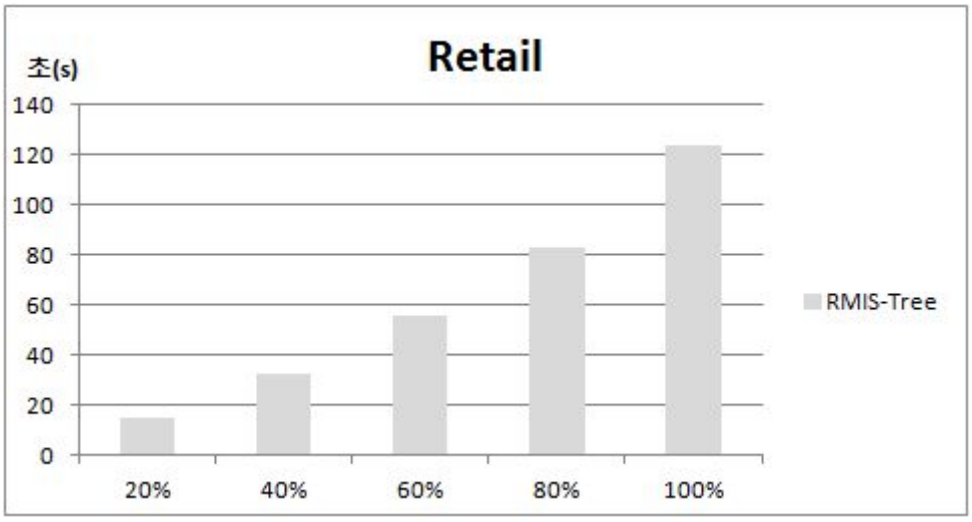


그림 21 증가하는 Retail 데이터셋에 대한 트리 구성 시간

RMIS-Grow가 새로운 데이터 구조 RMIS-Tree를 이용하여 증가하는 데이터베이스에 대한 트리 구성과 마이닝에 적합함을 보였다. RMIS-Tree는 높은 MMS와 너무 낮은 정규도 임계치로 인해 매우 적은 패턴이 마이닝되는 조건에서 더 많은 시간이 소모된다. 그러나 일정 수 이상의 패턴이 마이닝되는 조건에서 보다 효율적인 면을 보인다. 그리고 패턴이 불필요한 패턴이 더 많이 제거되어 적은 패턴이 마이닝 되고, σ 값을 조절함으로써 마이닝 되는 패턴 수를 조절함으로써 의사 결정을 하는데 필요한 분석에 더 적절하다.

Ⅴ. 결론

본 논문에서는 정규 패턴을 마이닝 하기 위한 새로운 기법인 RMIS-Growth를 제안하였다. 기존의 패턴 마이닝 기법들이 레어 아이템 문제를 유발시키는 문제를 해결하기 위해 제안한 RMIS-Growth는 패턴의 정규도 뿐만 아니라 다중 최소 임계치를 적용하였다. RMIS-Growth는 밀집 데이터셋인 Chess, Mushroom와 비밀집 데이터셋인 T10I4D100K, Retail에서 일정한 트리 구축 시간을 보였다. 그리고 일정 수 이상의 패턴이 마이닝 되는 임계치에서 기존 알고리즘 보다 빠른 패턴 마이닝 시간을 보임으로써 정규 패턴 마이닝에 적합함을 보였다. 그리고 RMIS-Growth는 기존 패턴 마이닝 기법보다 불필요한 패턴을 더 많이 걸러낸다. 기존 알고리즘의 80% 이하의 패턴을 마이닝하므로 의사 결정을 위한 분석에도 더 적절하다. 또한 점진적으로 증가하는 데이터베이스에서도 RMIS-Growth는 정규 패턴을 마이닝에 적합함을 보였다. 이를 위해 한 번의 데이터베이스 스캔으로 구축된 트리는 재구축 되며, 점진적으로 추가된 트랜잭션 정보는 전체 데이터베이스에 대한 추가적인 스캔 없이 효율적으로 반영된다. 본 연구에서 제안한 RMIS-Growth를 이용함으로써 지속적으로 누적되는 판매정보, 주기적으로 발생하는 질병 등의 데이터베이스 내에서 중요한 정규 패턴 마이닝과 의사 결정에 기여할 것으로 기대한다.

참고문헌

[1] R. Agrawal, R Srikant, “Fast Algorithms for Mining Association Rules” Proc. of the 12th ACM SIGMOD Int. Conf. on Management of Data, May 1993, pp. 207-216.

[2] J. H., J. Pei, Y. Yin, R. Mao “Mining Frequent Patterns without Candidate Generation” Data Mining and Knowledge Discovery, Volume 8, 2004, pp. 53-87.

[3] R. Agrawal, P. E. Stolorz, G. P. Shapiro, “Database Methods for Data Mining,” Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD-98), 1998.

[4] Y. -H. Hu, Y. -L. Chen “Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism” Decision Support Systems, Volume 42 Oct, 2006, pp. 1-24.

[5] R. U. Kiran, P. K. Reddy “Novel Techniques to Reduce Search Space in Multiple Minimum Supports-Based Frequent Pattern Mining Algorithms” Proc. of the 14th Int. Conf. Extending Database Technology, 2011 pp. 11-20.

[6] S. K. Tanbeer, C. F. Ahmed, B. -S. Jeong “Mining Regular Patterns in Data Streams” Proc. of the 15th Int. Conf. on Database Systems for Advanced Applications, 2010, pp. 399-413.

[7] S. K. Tanbeer, C. F. Ahmed, B. -S. Jeong “Mining Regular Patterns in Incremental Transactional Databases" Proc. of the 12th Asia-Pacific Web Conference, 2010, pp. 375-377.

[8] S. K. Tanbeer, C. F. Ahmed, B. -S. Jeong, Y. -K. Lee “Mining Regular Patterns in Transactional Databases” IEICE Transactions on Information and Systems Vol. E91.D, 2008 No.11 pp. 2568-2577.