KIPS Tr. Software and Data Eng.
Vol.5, No.3 pp.155~164 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.3.155

IRFP-tree: Intersection Rule Based FP-tree

Jung-Hun Lee†

ABSTRACT

For frequency pattern analysis of large databases, the new tree-based frequency pattern analysis algorithm which can compensate for the disadvantages of the Apriori method has been variously studied. In frequency pattern tree, the number of nodes is associated with memory allocation, but also affects memory resource consumption and processing speed of the growth. Therefore, reducing the number of nodes in the tree is very important in the frequency pattern mining. However, the absolute criteria which need to order the transaction items for construction frequency pattern tree has lowered the compression ratio of the tree nodes. But most of the frequency based tree construction methods adapted the absolute criteria. FP-tree[1] is typically frequency pattern tree structure which is an extended prefix-tree structure for storing compressed frequent crucial information about frequent patterns. For construction the tree, all the frequent items in different transactions are sorted according to the absolute criteria, frequency descending order. CanTree[2] also need to absolute criteria, canonical order, to construct the tree. In this paper, we proposed a novel frequency pattern tree construction method that does not use the absolute criteria, IRFP-tree algorithm. IRFP-tree(Intersection Rule based FP-tree). IRFP-tree is constituted with the new paradigm of the intersection rule without the use of the absolute criteria. It increased the compression ratio of the tree nodes, and reduced the tree construction time. Our method has the additional advantage that it provides incremental mining. The reported test result demonstrate the applicability and effectiveness of the proposed approach.

Keywords : Big Data Analysis, Data Mining, Frequent Pattern Analysis, FP-Tree, DRFP-Tree

IRFP-tree(Intersection Rule Based FP-tree): 메모리 효율성을 향상시키기 위해 교집합 규칙 기반의 패러다임을 적용한 FP-tree

이 정 훈†

요 약

대용량 데이터베이스의 빈도패턴 분석을 위해 기존의 Apriori 방식의 단점을 보완할 수 있는 새로운 트리 기반의 빈도 패턴 분석 알고리즘이 최근 다양하게 연구되고 있다. 그 중 FP-tree[1]는 이러한 빈도 패턴을 분석하기 위해 빈도 패턴을 표현하는 트리 구조로 단 두 번의 전체 데이터베이스 스캔을 통해 빠르게 트리를 구성할 수 있으며 FP-grwoth를 통해 빈도 패턴을 분석할 수 있다. 이처럼 빈도 패턴 트리의 노드 수는 트리 자체의 메모리 할당량과도 연관이 있지만 그 후 growth의 메모리 자원 소비 및 처리 속도에도 영향을 미치게 된다. 따라서 빈도 패턴 트리의 노드 수의 감소는 트리 자체뿐만 아니라 빈도 패턴 분석에 있어서도 매우 중요하다. 하지만 FP-tree는 전체 아이템 수 라는 고정된 기준 문제로 인해 충분한 노드 수의 압축률을 갖지 못하고 있다. 본 논문에서는 이러한 FP-tree의 문제를 보완하여 좀 더 노드 수를 감소시킬 수 있도록 교집합 규칙이라는 새로운 패러다임을 적용한 빈도 패턴 트리인 IRFP-tree를 제시하고 실험을 통해 그 성능에 대해 증명하였다.

키워드 : 빅 데이터 분석, 데이터 마이닝, 빈도패턴분석, FP-tree, DRFP-tree

1. 서 론

빈도 패턴 마이닝은 데이터 마이닝 분야에서 광범위하게 연구되어 왔다. 1993년에 소개된 이래로, Apriori[1, 2] 방법은 데이터 마이닝 분야에서 많은 관심을 받아왔다. 하지만 Apriori 기반의 접근방법들은 많은 후보 집합을 생성하고 잦은 데이터베이스 스캔으로 인해 비용이 많이 든다는 단점을 가지고 있다. 이러한 단점을 극복하기 위해 많은 논문에서 트랜잭션 데이터베이스로부터 빈도 아이템 집합들을 계산해 내는 새로운 데이터 구조를 제시하고 있다. 이러한 데이터 구조들 중 최근에 가장 인기 있는 것 중 하나가 FP-tree 구조이다.

† 정 회 원 : 동국대학교 전산원 컴퓨터해킹보안전공 교수
Manuscript Received : November 19, 2015
First Revision : February 29, 2016
Accepted : February 29, 2016
* Corresponding Author : Jung-Hun Lee(leeye123@naver.com)

FP-tree[3]는 Apriori 방식의 접근방법의 단점을 피하고 빈도 패턴들과 관련한 중요하고 양적인 정보를 저장하기 위한 확장된 prefix-tree 구조이다. FP-tree는 단 두 번의 전체 데이터베이스 스캔으로 트리를 생성함으로써 비용이 많이 들고 반복적인 데이터베이스 스캔을 피할 수 있다. 또한 FP-tree 기반의 마이닝 기법인 FP-growth를 이용하여 완전한 빈도 패턴들의 집합을 마이닝할 수 있다. FP-tree를 이용한 빈도 패턴 마이닝이 Apriori 알고리즘이나 그 밖에 최근의 새로운 패턴 마이닝 방법에 대한 연구들보다 매우 빠르다는 것은 [3]에서 제시한 연구에서 실험을 통해 확인되었다.

FP-tree는 전체 데이터베이스를 한번 스캔하여 데이터베이스 내에 아이템들의 발생 빈도 수를 구하고, 이를 이용하여 또 한번 데이터베이스를 스캔하며 트리를 구성한다. FP-Tree는 트리를 구성하기 전 전체 데이터베이스 스캔을 통해 아이템의 발생 빈도를 구해야 하기 때문에 두 번의 데이터베이스 스캔이 필요할 뿐만아니라 가변하는 데이터베이스에서 적용이 불가능하다. 이런 단점을 극복하기 위해 많은 빈도패턴 트리 생성 방법이 연구되었다. 기존의 FP-tree를 스트리밍 데이터베이스에서 적용 가능하도록 FP-stream[4] 구조가 제시되었고, 트리 생성 시 가지치기를 통해 조건 트리를 최소한으로 생성할 수 있도록 하는 COFI-tree[5, 6], 메모리 대신 DB를 사용하도록 하는 DRFP-tree[7, 8], FP-tree의 아이템 발생 빈도 수를 구하기 위한 데이터베이스 스캔을 줄이기 위한 방법으로 알파벳 순 특정 기준을 이용해 트리를 구성하는 CanTree[9] 등의 빈도패턴 트리 구조가 제시되었다. 이 중 DS-Tree[10]는 알파벳 순서나 사전편찬순서와 같은 단순한 기준을 이용하여 트리를 구성하기 때문에 단 한번의 데이터베이스 스캔으로 빈도 패턴 트리를 구성하여 FP-tree보다 비슷하거나 빠른 속도를 보장하며, 스트리밍 환경에서 빈도 패턴 마이닝을 위한 트리 구성이 가능하다.

이러한 연구들은 대부분 트리를 구성할 때 아이템 발생 빈도나 알파벳 순서, 사전편찬순서 등의 절대적인 기준을 이용하여 트랜잭션 내 아이템들을 정렬하고 트리를 구성한다. 하지만 절대적인 기준을 이용해 트리를 구성할 경우 트랜잭션이 입력될 때 마다 변하는 아이템들의 빈도 수 변화를 충분히 반영하지 못해 충분히 높은 압축률을 얻을 수 없다. Goethals[13], Vaarandi[14], 그리고 Buehrer[15] 등이 연구한 바에 따르면 아주 큰 데이터베이스를 주 기억장치에서 마이닝하기는 어려움이 있다고 하였다. 이처럼 메모리상에서 처리되는 FP-tree기반의 빈도패턴분석 방법에 있어서 메모리 효율성은 매우 중요한 요소가 아닐 수 없다.

본 논문에서는 절대적인 아이템 정렬 기준을 이용하지 않고 각 트랜잭션이 입력되어 트리를 구성할 때 마다 변하는 아이템 발생 빈도가 반영될 수 있도록 교집합 규칙이라는 새로운 패러다임을 이용하여 트리를 생성하고, 단 한번의 데이터베이스 스캔만으로 빠른 속도로 기존의 FP-tree에 비해 높은 압축률을 보이며 가변하는 데이터베이스에서 빈도 패턴 마이닝을 위한 트리를 구성할 수 있는 새로운 방법을 제시하고 실험을 통해 기존의 방법들과 비교 분석하였다.

2장에서는 본 논문과 관련된 연구들 특히 FP-Tree 및 DSTree에 대해 살펴보고, 3장에서 기존 연구의 문제점을 분석한 후 4장에서 우리가 제시하는 새로운 빈도 패턴 트리에 대해 설명하고 5장에서 실험을 통해 새로운 방법과 기존 방법을 비교 분석하였다.

2. 관련 연구

Apriori는 연관규칙에서 가장 대표적인 알고리즘으로 이진 연관규칙에 대한 빈발항목집합을 찾아내는데 유용한 알고리즘이다. Apriori 알고리즘은 실제 발생되는 빈발 패턴을 찾아내는 데 가장 좋은 성능을 보여준다. 하지만 반복적으로 후보 항목집합들을 생성하고 데이터베이스를 스캔 하면서 지지도를 계산하기 때문에 처리 시간이 많이 소비된다. 이를 보완하기 위해 후보집합의 수를 줄이거나 데이터베이스의 스캔 횟수를 줄이기 위한 여러 연구가 진행되어 왔다. 그 중 FP-Growth 알고리즘은 후보집합을 생성하지 않고 단 두 번의 데이터베이스 스캔만으로 빈발패턴 분석이 가능하여 주목을 받고 있다.

FP-Tree 알고리즘은 기존의 여러 연관규칙 알고리즘들과는 달리 Apriori 방법을 사용하지 않는다. 빈발 항목집합들에 대한 중요하고 양적인 정보를 FP-Tree라 부르는 확장된 prefix tree 구조에 저장한다. FP-Tree는 두 번의 데이터베이스 스캔만으로 빈도 패턴 트리를 생성한다. 첫 번째 데이터베이스 스캔을 통해 각 아이템들의 빈발횟수를 카운트하여 아이템의 우선순위를 결정한다. 두 번째 데이터베이스 스캔을 통해 입력되는 각 아이템 집합을 빈발횟수를 이용하여 정렬한다.

아래 FP Tree를 생성하는 간단한 예가 나타나 있다. Fig. 1-(a)는 FP Tree를 생성하기 위한 예제 트랜잭션 데이터베이스이다. 이 트랜잭션 데이터베이스는 다섯 개의 트랜잭션을 가지고 있다. 트리 생성할 때 최소지지도는 3으로 한다. 각 행은 한 개의 트랜잭션 내에 동시에 발생한 아이템들의 집합으로 구성되어 있고, TID로 구분된다. 이 예제 트랜잭션 데이터베이스를 이용하여 FP-Tree를 생성하기 위해, 우선 아이템들의 빈도수를 구해야 한다. 데이터베이스에 표현된 아이템들의 수를 카운트하기 위해 처음으로 데이터베이스를 한 번 스캔한다. 그리고 나서, 빈도 아이템 리스트를 만들기 위해 아이템들을 빈도수가 많은 순서대로 최소 지지도 이상인 아이템들만 FP Tree를 생성하는데 사용된다. Fig. 1-(c)는 최소지지도 이상의 빈도수를 가진 아이템들을 빈도수가 많은 순서대로 정렬한 것이다.

다음 단계는, FP-tree를 생성하기 위해 트랜잭션 데이터베이스를 두 번째로 스캔 한다. 루트부터 시작해서 전위 트리 방식으로 루트 하위 트리로 하나씩 트랜잭션들을 추가한다. 각 트랜잭션을 읽은 후, 그 아이템들을 빈도수의 역순으로 재정렬한다. 최소지지도에 미치지 못하는 아이템들은 고려하지 않는다. Fig. 1-(b)는 Fig. 1-(a)의 지지도 3 미만인 아이템들을 생략하고 정렬한 후의 트랜잭션을 보여준다.

TID	Items
100	F,A,C,D,G,I,M,P
200	A,B,C,F,L,M,O
300	B,F,H,J,O,W
400	B,C,K,S,P
500	A,F,C,E,L,P,M,N

(a) Transaction database

TID	Items
100	F,C,A,M,P
200	F,C,A,B,M
300	F,B
400	C,B,P
500	F,C,A,M,P

(b) Ordered and Truncated Transactional Database

Items	Count
F	4
C	4
A	3
M	3
P	3
B	3

(c) Item list ordered by frequency

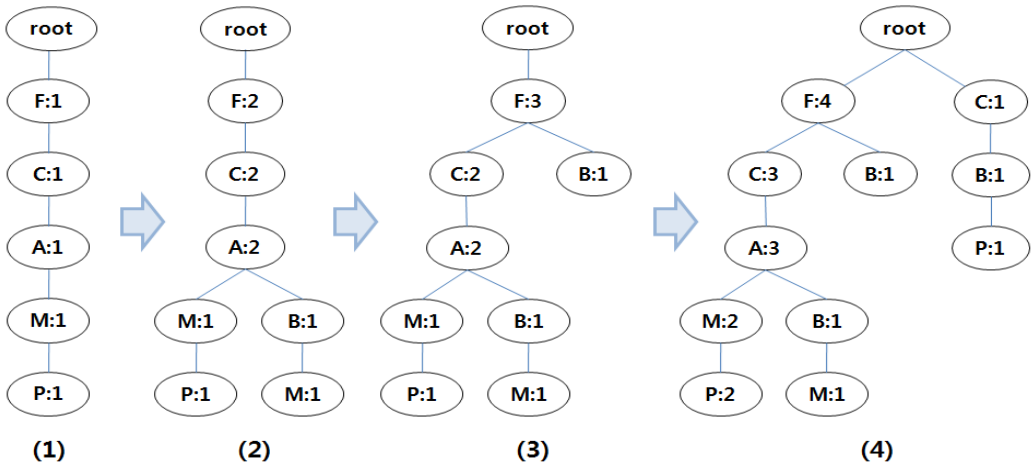


Fig. 1. Steps of FP-tree construction

Fig. 1의 (1)~(4)는 FP-tree를 위해 Fig. 1-(b)의 다섯 개의 트랜잭션을 더하기 위한 네 단계의 FP-Tree 생성 절차를 보여준다. Fig. 2는 Fig. 1-(a)의 트랜잭션 데이터베이스에 대한 최종적인 FP-tree와 그 헤더 테이블을 보여준다.

이와 같이 FP-tree는 Apriori에 비해 데이터베이스의 잦은 스캔을 줄여준다. 또한 후보집합을 생성하지 않기 때문에 대용량의 데이터로부터 빈발항목집합을 찾아내는데 유용하다. 하지만 트리의 깊이가 깊어지고 노드의 수가 많아질 경우 메모리크기의 의존성이 크고, 마이닝에 많은 처리 시간이 소비될 수 있다.

FP-Tree가 소개된 이래로 COFI-tree[5, 6], DRFP-tree[7, 8], CanTree[9], DSTree[10], AFOPT-tree[11], CATS Tree[12] 등 많은 빈도 패턴 마이닝을 위한 tree구조가 제시되었다.

FP-Tree는 전체 데이터베이스를 읽고 아이템의 빈발 횟수를 이용하여 트리를 구성하기 때문에 고정된 데이터베이스에만 적용되는 데이터의 마이닝 기법으로 스트리밍 데이터베이스에서는 사용할 수 없다. 이러한 FP-tree의 제약 조건을 벗어나기 위해 Carson Kai-Sang Leung, Quamrul I .Khan은 [10]에서 스트리밍 데이터베이스에서의 마이닝을 위한 DS-Tree라는 연관규칙 마이닝 방법을 제시하고 있다. DS-Tree는 아이템 빈발횟수를 이용하지 않고 알파벳 순서나 사전편찬순서 등 아이템 특성에 맞도록 사용자가 정한 기준으로 아이템들을 정렬한다. 정렬 후 윈도우 크기만큼 트랜잭션의 집합인 배치를 읽어와 트리를 생성한다.

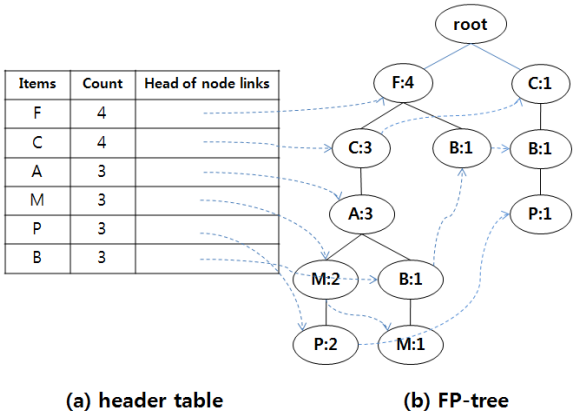
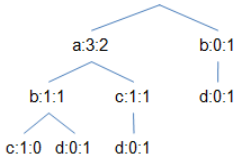


Fig. 2. The FP Tree built based on the data in Fig. 1-(a)

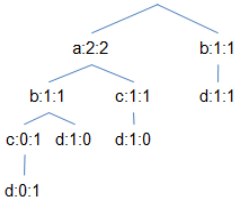
DSTree는 고정된 데이터베이스뿐만 아니라 데이터 스트림 환경에서도 적용할 수 있는 트리 구성 방법이다. FP-tree에서 아이템 빈발 횟수를 아이템 정렬 기준으로 이용한 것에 반해, DSTree에서는 트리 구성을 위한 기준으로 알파벳 순이나 사전 편찬 순과 같은 단순한 기준을 이용하고 있다. 따라서 아이템 정렬 기준을 찾기 위한 데이터베이스 스캔이 생략될 수 있기 때문에 단 한 번의 데이터베이스 스캔으로 트리 구성이 가능하며, 데이터 스트림 환경에서도 트리 구성이 가능하다. 또한 단 한 번의 데이터 스캔만이 필요하기 때문에 트리를 구성하는데 걸리는 시간을 매우 줄일 수 있

Example 1 Consider the following stream of transactions:

Batch	Transactions	Contents
first	t1	{a, b, c}
	t2	{a}
	t3	{a, c}
second	t4	{a, c, d}
	t5	{b, d}
	t6	{a, b, d}
third	t7	{b, d}
	t8	{a, b, c, d}
	t9	{a, c}



(a) At time T(The DSTree capturing 1st & 2nd batches)



(b) At time T'(The DSTree capturing 2nd & 3rd batches)

Fig. 3. The DSTrees after each batch of transactions is added for stream mining

다. 아래 DSTree를 구성하는 방법을 설명하였다.

Fig. 3은 DSTree를 구성하는 한 가지 예이다. Fig. 3의 Example 1은 DSTree 예제 트랜잭션 데이터베이스이다. 트리를 구성하기 위해 최소지지도는 3으로, 윈도우 사이즈는 2개의 배치로, 각 배치는 3개의 트랜잭션을 포함하도록 구성하였다. 아이템 정렬 기준은 알파벳 순서로 하였다. Fig. 3의 오른쪽 상단의 트리는 Time T일 때의 DSTree를 보여준다. Time T는 첫 번째와 두 번째 배치가 들어와 트리로 구성하는 시기이다. 우선 첫 번째 배치의 트랜잭션 순서대로 트리를 구성하며 아이템 우측에 발생 빈도를 표기한다. 그 후 두 번째 배치의 트랜잭션을 순서대로 입력하여 트리를 구성하며, 그 발생 빈도는 첫 번째 배치의 아이템 발생 빈도 수와 구분하여 그 뒤에 입력한다. Time T'가 되어 세 번째 배치가 들어오면 첫 번째 배치의 아이템 발생 빈도를 삭제하고, 두 번째 배치의 아이템 발생 빈도를 좌측으로 한 칸 이동시킨 후, 세 번째 배치의 트랜잭션 순서대로 트리를 구성하며 아이템 발생 빈도를 저장한다. 이처럼 DSTree의 경우 가장 최근의 윈도우 크기만큼의 데이터에 대해서만 정보를 저장하며, 오래된 정보는 무시한다.

3. 기존 연구들의 문제점 분석

앞서 언급한 바와 같이 기존에 연구된 FP-tree 기반의 빈도 패턴 트리 구조는 고정된 기준을 이용하여 트리를 생성하기 때문에 충분한 노드의 압축률을 얻을 수 없다. 트리의 노드 수 증가는 메모리의 소비의 증가뿐만 아니라 FP Growth를 실행할 때 방문하는 경로를 증가시키며, 트리 생성 비용 증가시키기 때문에 가능한 한 높은 압축률의 트리를 구성할 필요가 있다. 트리를 생성할 때 지속적으로 변하는 아이템 빈도 수를 반영할 수 있다면 노드의 최적화를 통해 트리의 전체 노드 수를 줄일 수 있을 것이다.

Fig. 4는 고정적인 기준으로 트리를 생성하는 FP-Tree의 문제에 대해 그림으로 나타낸 것으로 Fig. 4-(a)는 실험 데이터를 이용해 만든 FP-Tree의 예를, Fig. 4-(b)는 가변하는 트랜잭션 아이템 빈도 수를 반영하여 트리를 재 생성 했을 경우 압축률을 보여주고 있다. FP-Tree는 트리를 생성하기 전 한 번만 DB내 전체 아이템 수를 계산하여 빈도 아이템 리스트를 만든다. 이 빈도 아이템 리스트는 트리 생성하기 전에 만들어져 트리가 다 만들어 질 때까지 변하지 않는다. 하지만 특정 트랜잭션까지 트리 생성 후 남은 트랜잭션들의 아이템 빈도 수는 처음 계산한 아이템 발생 빈도 수와 다를 수 있다. Fig. 4-(a)에서 T600까지 트리가 생성된 이후 남은 트랜잭션들의 아이템 발생 빈도 수를 계산한 결과 아이템 발생 빈도순서는 B>C>D>F에서 C>D>B>F로 변한 것을 볼 수 있다. 이처럼 변화된 아이템 정렬 기준을 이용해 트리를 생성한 결과, T700에서 T900까지 트리 생성 노드 수는 FP-tree의 {B : 1, C : 1, D : 1, C : 2, D : 1, F : 1}의 6개에서 {C : 3, D : 2, B : 1, F : 1}의 4개로 줄어든 것을 볼 수 있다. 이는 고정된 기준을 이용해 트리를 생성하기 때문에 생기는 문제로, 빈도 수 뿐만 아니라 알파벳 순서나 사전편찬순서 등 트리 생성 전 만들어진 하나의 기준을 이용해 전체 트리를 생성하기 때문에 발생하는 문제이다. 트랜잭션을 입력할 때마다 변하는 아이템 발생 빈도를 트리 생성에 반영할 수 있다면 FP-Tree 이상의 압축률을 가진 트리를 만들 수 있을 것이다.

DSTree는 알파벳 순서와 같은 단순한 정렬기준을 이용하여 한 번의 스캔으로 트리를 구성한다. 특별한 추가 작업 없이 알파벳 순서와 같은 단순 기준으로 트리를 구성하기 때문에 실행 속도가 매우 빠르다. 또한 스트리밍 데이터 마이닝이 가능하다. 하지만 이 역시 고정 기준을 이용해 트리를 생성하기 때문에 충분히 높은 압축률을 보장하지 않는다. 또한 DSTree는 최근의 윈도우사이즈만큼의 데이터만을 유지하기 때문에 오래된 데이터에 대해서는 유효성을 보장하지 않는다.

이 논문에서는 고정된 기준을 이용하여 아이템 정렬 순서를 정하지 않고, 가변하는 아이템 빈도 수를 반영할 수 있

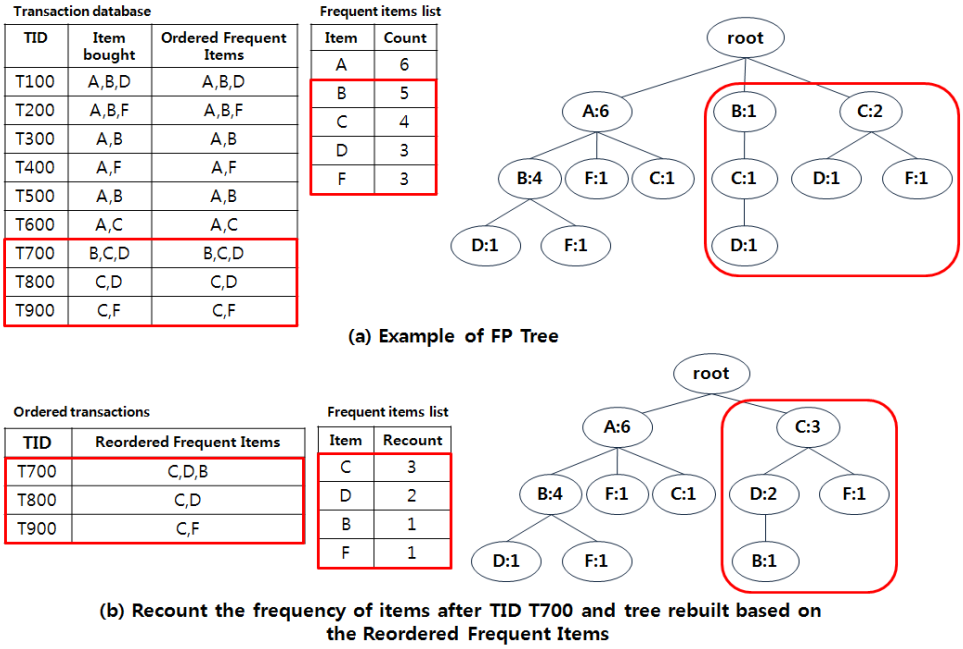


Fig. 4. FP-tree constructed based on fixed criteria may not always be minimal

는 규칙 기반의 새로운 트리 생성 방법을 소개한다. 또한 하위 경로 내 빈도수가 동일한 아이템들을 하나의 노드로 표현하는 집합 노드 방식을 제시하여 노드의 수를 좀 더 줄일 수 있도록 하였다. 이 두 가지 제시된 방법을 통해 지속적으로 데이터 입력이 일어나는 incremental 데이터베이스에서도 트리 생성이 가능하도록 하고, 트리의 압축률을 향상시키면서 기존의 FP-tree보다 빠른 처리 시간을 보장하는 트리 생성 방식을 제시한다.

4. 우리가 제시하는 방법

기존의 FP-Tree는 발생빈도수라는 절대적인 기준을 통해 트리를 생성하고 있다. 하지만 이 논문에서는 교집합 규칙을 이용하여 트리를 생성하는 새로운 방법을 제시하고 있다.

교집합 규칙 기반의 IRFP-tree는 특정한 기준을 이용해 아이템을 정렬하여 트리를 구성하는 방식이 아닌, 각 트랜잭션이 입력될 때 마다 교집합을 이용하여 교차하여 발생된 아이템들을 묶어 상위 노드로, 빈번하지 않게 발생한 아이템들의 집합을 하위 노드로 하여 트리를 생성해 나가는 방식이다. 하나의 노드에는 하나의 아이템만 포함될 수도 있고, 하나의 노드에 여러 아이템들이 포함될 수도 있다. 이 방법은 트랜잭션 내 아이템들을 재정렬 할 필요가 없으며, 계속되는 트랜잭션의 유입도 문제가 되지 않기 때문에 점진적 데이터 마이닝(incremental datamining)에도 적용이 가능하다. 새로운 하위 트리가 생성될 경우 아이템 집합들 중에서 가장 교집합이 많은, 다시 말하면 입력된 트랜잭션까지 가장 빈도수가 높은 아이템 집합이 상위 노드가 되도록 유도하기 때문에 트랜잭션이 입력될 때 마다 그 때 까지의 아이템 빈도수가 지속적으로 적용되어 노드 최적화가 가능하다.

Fig. 5와 Fig. 6은 Fig. 1-(b)의 트랜잭션 데이터베이스 예를 이용하여 IRFP-tree를 생성하는 방법을 하나는 집합 표현 방식으로, 하나는 트리 구조로 나타낸 것이다. Fig. 1-(ii)의 트랜잭션 100, 200이 들어온 경우, Fig. 5-(ii)와 같이 두 집합의 교집합인 {a,c,f,m}은 두 번 씩 발생되었고,

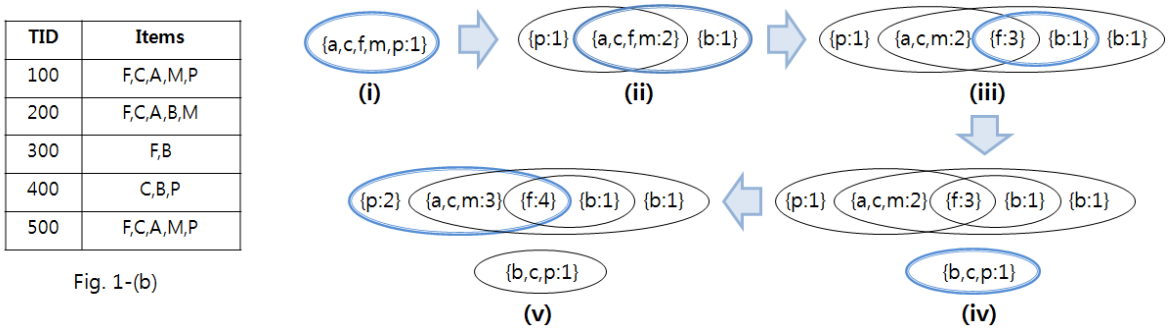


Fig. 1-(b)

Fig. 5. Construction of intersection rule based IRFP-tree

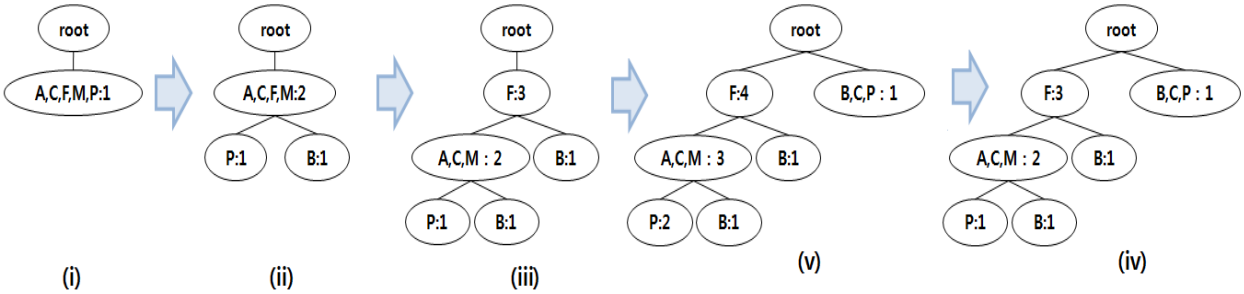


Fig. 6. Tree representation of IRFP-tree

{p}와 {b} 아이템 집합은 한 번 씩 발생되었다. 이를 트리를 이용해 나타내면 Fig. 6-(ii)와 같이 표현할 수 있다. 이처럼 트랜잭션이 유입될 때 마다 교집합을 이용해 아이템 집합들을 묶어 나가면 최종적으로 Fig. 7과 같은 트리를 생성할 수 있다.

Fig. 6에서와 같이 교집합 규칙을 이용한 IRFP-tree는 하나의 노드가 아이템 집합을 갖는 형태의 트리를 가지게 된다. 이처럼 집합으로 표현된 노드 방식은 Growth를 수행할 때 방문하는 노드의 수를 감소시키는 이점을 줄 수 있다. 또한 특정 기준이 아닌 규칙 기반의 트리 이기 때문에 구축된 트리에서 트랜잭션의 추가 및 삭제가 용이하다.

하지만 FP-tree와 정당한 노드 수의 비교를 위해 우리는 다음과 같이 하나의 노드가 하나의 아이템을 갖는 형태의 IRFP-tree 또한 생성하였다.

Fig. 7에서는 규칙 기반의 새로운 패러다임을 적용시켜 고정된 기준 문제를 해결한 IRFP-tree가 집합 방식을 사용하지 않는다 하더라도 기존의 FP-tree에 비해 노드 수를 더욱 감소시켰다는 것을 보여주기 위해 집합방식을 사용하지 않고 하나의 노드가 하나의 아이템을 표현하는 FP-tree 표현 방법에 맞추어 트리를 표현한 것이다. Fig. 7의 IRFP-tree에서 점선으로 나타난 노드들의 묶음은 IRFP-tree의 집합표현이다.

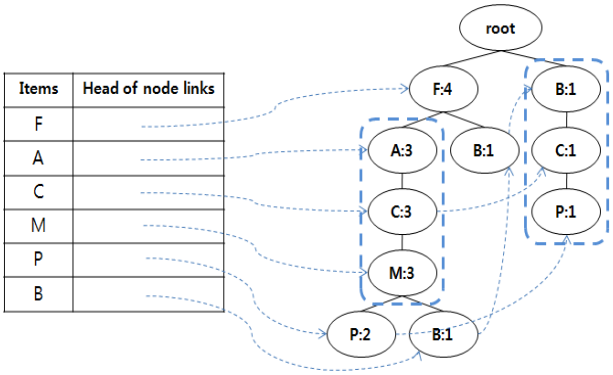


Fig. 7. IRFP-tree constructed based on Fig. 1-(b)

IRFP-tree를 생성하기 위한 알고리즘은 Pseudo Code 1.에 나타나 있다.

Algorithm: IRFP Tree Builder
Input: transaction database
Output: IRFP Tree

```
1.  PROCEDURE IRFPTreeBuilder(transactionDatabase DB)
2.      NODE node = NULL;
3.      for all transactions t ∈ DB
4.          treeConstruct(t, node);
5.  PROCEDURE treeConstructor(items, currentNode)
6.      if(child node.items ∩ items == ∅ )
7.          addNode(items, currentNode);
8.      else
9.          if (child node == items)
10.             childNode.frequency ++;
11.         else if (child node.items ⊃ items)
12.             splitNode(items, childNode);
13.         else if (child node.items ⊂ items)
15.             childNode.frequency ++;
16.             restItems = items - childNode.items;
17.             treeConstructor(restItems, childNode);
18.         else
19              splitAndAddNode(items, childNode);
20. PROCEDURE addNode(items, node)
21.     node.child ← items;
22. PROCEDURE aplitNode(items, node)
23.     node.child ← node - items;
24.     node = node ∩ items;
25.     node.frequency++;
26. PROCEDURE splitAndAddNode(items, node)
27.     node.child ← node - items;
28.     node.child ← itmes - node;
29.     node = node ∩ items;
30.     node.frequency++;
```

Pseudo Code 1. IRFP-tree Builder

FP-tree의 생성 알고리즘에선 트리 생성 전 처리 단계인 전체 DB를 스캔하여 아이템의 빈도수를 계산하고 그를 이용하여 아이템 순위를 정하고 각 트랜잭션의 아이템들을 아이템 빈도 수의 역순으로 정렬한 후 트리를 구성해야 하는

비용이 필요하다. 하지만 IRFP-tree는 그러한 트리 구성의 전 처리 단계가 필요치 않고 바로 DB의 트랜잭션을 읽어 들이며 트리의 생성을 진행하기 때문에 전처리 비용을 줄일 수 있다. 또한 FP-tree의 경우 DB로부터 하나의 트랜잭션을 읽어와 트랜잭션 내의 아이템들을 하나씩 읽어나가며 트리를 구성하지만, IRFP-tree는 한 트랜잭션을 읽어와 트랜잭션 단위로 트리를 구성하기 때문에 FP-tree에 비해 시간 비용을 줄일 수 있었다.

다음 장 에서는 앞서 살펴본 IRFP-tree의 개별 노드 표현방법 및 집합표현방법을 이용하여 실험을 통해 IRFP-tree와 FP-tree, DS-tree의 트리 구성 시간 및 노드 최적화 율에 대해 좀 더 객관적이고 정당하게 비교 분석해 보았다.

5. 실 험

이 장에서는 우리가 제시한 IRFP-Tree의 성능을 기존의 FP-tree, DS-Tree와 실험을 통해 비교해 보고 그 결과에 대해 분석하였다. 샘플 데이터베이스를 이용하여 앞서 살펴본 각각의 트리를 생성하고, 이 때 각 트리의 생성 시간과 생성된 노드의 수를 비교하였다. 똑같은 빈도 패턴을 찾아낼 경우, 트리 생성 속도가 빠르고 생성된 노드의 수가 적을수록 성능이 좋은 알고리즘으로 평가하였다.

실험을 위해 사용된 샘플 데이터베이스의 구성은 다음과 같다. 아이템은 총 26가지이며, 이 26가지의 아이템을 랜덤하게 발생하여 하나의 아이템 집합을 만든다. 이 하나의 아이템 집합을 하나의 트랜잭션이라 할 때, 트랜잭션의 개수에 따른 트리 성능 변화의 비교를 위해 트랜잭션의 수를 증가시켜 가며 총 T10I4D100K과 유사한 양의 데이터를 이용하여 실험하였다.

우선, 본 실험에 앞서 논문에서 문제 제기한 기존 FP-tree의 고정된 기준 문제에 대해 실험해보았다. 고정된 기준을 이용하여 FP-tree를 구성할 경우 실제로 노드 압축률에 대한 문제가 발생되는지 알아보기 위해, 트리를 생성할 때 아이템 빈도수의 변화에 대해 반영한 개선된 FP-tree를 구현하여 새로 개선된 FP-tree가 기존의 FP-tree에 비해 어느 정도의 노드 압축률을 보이는지 실험하였다. 이를 평가하기 위해 실험 데이터베이스를 이용하여 기존의 FP-tree와 개선된 FP-tree, 그리고 우리가 제시하는 IRFP-tree 알고리즘을 이용하여 트리를 생성하며 각각의 트리에 생성된 노드의 수를 비교하였다. 실험을 위해 10,000의 트랜잭션을 1000개씩 증가시켜 가며 18,000개까지의 데이터 파일을 생성하며, 각 트랜잭션의 개수마다 랜덤하게 50가지의 데이터파일을 생성하여 총 450개의 데이터를 파일을 이용해 각각의 트리를 생성하였다.

Fig. 8은 기존의 FP-tree와 개선된 FP-tree, 우리가 제시하는 IRFP-tree 알고리즘을 이용하여 트리를 생성한 경우 각각 트리의 노드 수를 그래프로 보여주고 있다. IRFP-tree의 경우는 좀 더 정당한 비교를 위하여 하나의 노드가 하나의 아이템만을 표현하도록 개별 노드 트리를 사용하였다. 그래프의 가로축은 실험을 위해 생성한 데이터베이스를 구성하는 트랜잭션의 수를 나타내고, 좌측 축은 노드의 수를, 우측 축은 전체 생성된 아이템의 수를 보여준다. 각 트랜잭션 개수 크기의 50개의 데이터파일을 실험 데이터베이스로 만들어 트리를 구축하여 그 평균 노드 수 및 평균 아이템의 수를 그래프로 표현하였다. Fig. 4에서와 같이 FP-tree와 수정된 FP-tree의 생성된 노드의 수는 어느 정도 차이를 보인다. 이러한 차이는 트랜잭션의 수가 증가할수록 노드 수의 차이가 커진다. 반면 논문에서 제시한 IRFP-tree의 경우 역시 고정 기준의 문제가 해결되었기 때문에 수정된 FP-tree와 유사한 노드 수를 생성한다.

Fig. 9는 실험을 통해 나타난 FP-tree 대비 IRFP-tree의 노드 감소 비율을 그래프로 표현한 것이다. 그래프의 가로축은 트랜잭션의 수를, 세로축은 FP-tree에 비해 IRFP-tree의 노드 감소 비율을 백분율로 나타낸 것이다. 그래프에서

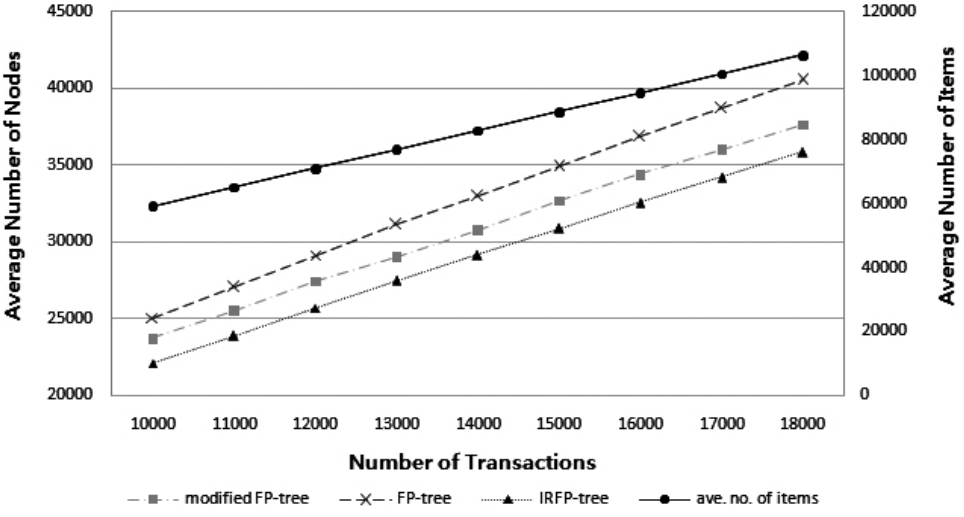
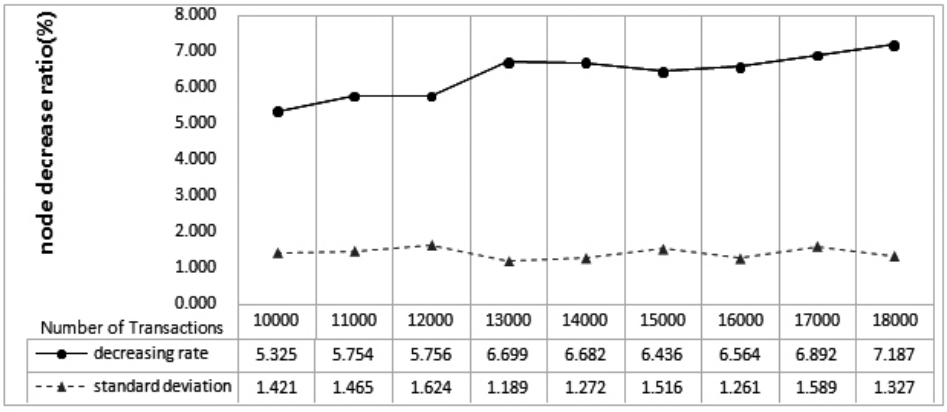


Fig. 8. Average number of nodes at various sizes of transactions



Number of Transactions	10000	11000	12000	13000	14000	15000	16000	17000	18000
decreasing rate	5.325	5.754	5.756	6.699	6.682	6.436	6.564	6.892	7.187
standard deviation	1.421	1.465	1.624	1.189	1.272	1.516	1.261	1.589	1.327

Fig. 9. IRFP-tree's node number decrease ration against FP-tree

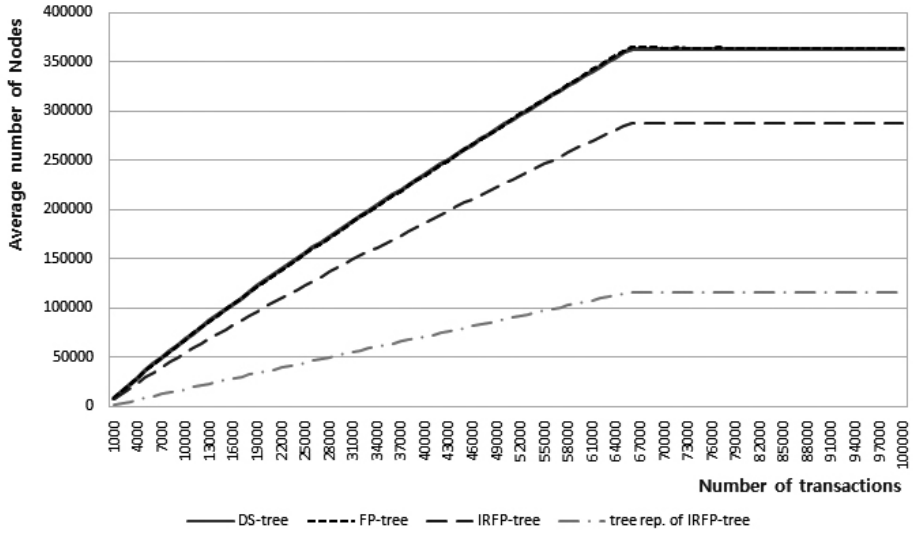


Fig. 10. Average number of nodes in each tree

와 같이 기존의 FP-tree를 이용하면 데이터베이스의 데이터베이스의 크기가 커질수록 노드의 압축률이 떨어지는 것을 볼 수 있다. 또한 감소량의 표준편차에서 보는 바와 같이 IRFP-tree를 이용하면 일시적인 현상이 아닌 안정적으로 노드의 압축률을 증가시켜 주는 것을 확인할 수 있다. 이처럼 빅 데이터에서의 패턴 분석에 IRFP-tree를 이용한다면 기존의 FP-tree에 비해 노드의 압축률을 증가시킬 수 있을 것으로 보인다.

앞서 기존의 FP-tree가 가진 고정된 기준 문제에 대한 실험을 통해 변화하는 빈도 수를 반영한다면 노드의 압축률을 줄일 수 있다는 것을 확인하고 이 때 특정 기준이 아닌 교집합 규칙 기반의 IRFP-tree 알고리즘을 이용하여 이러한 교집합 문제를 해결할 수 있음을 실험을 통해 알아보았다.

이제 본격적으로 우리가 제시하는 IRFP-tree와 기존의 FP-tree 그리고 DS-tree 알고리즘을 이용한 트리 생성을 통해 각 트리의 노드 압축률 및 트리 생성 속도를 실험을 통해 살펴보았다. 실험 데이터베이스는 1000개의 트랜잭션으로 시작하여 그 수를 1000개씩 증가시켜가며 1,000,000개까지의 트랜잭션을 이용해 트리를 생성하였을 때 각각의 트리에 생성되는 노드의 수와 처리속도를 비교 실험하였다.

Fig. 10은 트랜잭션 수 증가에 따른 각 트리에 생성된 노드의 수이다. 각 트랜잭션 개수마다 50개의 랜덤한 데이터 파일을 생성하여 실험한 후 그 평균을 그래프에 기록하였다. 그래프의 좌측 축은 각 트리의 생성된 노드 수를, 하단의 축은 트랜잭션의 수를 나타낸다. Fig. 3의 그래프에서 보는 바와 같이 전체 아이템 빈도 수라는 고정된 기준을 이용하는 FP-tree와 알파벳이나 사전편찬순서 등의 고정된 기준을 이용하는 DS-tree의 경우 거의 유사한 개수의 노드를 생성하고 있음을 볼 수 있다. 하지만 우리가 제시하는 교집합 규칙 기반의 FP-tree는 앞선 트리들에 비해 더 적은 수의 노드를 생성하고 있고, IRFP-tree의 특성 중 하나인 하나의 노드가 여러 개의 아이템을 가질 수 있는 집합 표현방식을 사용할 경우 IRFP-tree가 절대적인 노드 수의 감소를 보여준다. 이러한 노드 수 감소를 통해 메모리 자원의 소비를

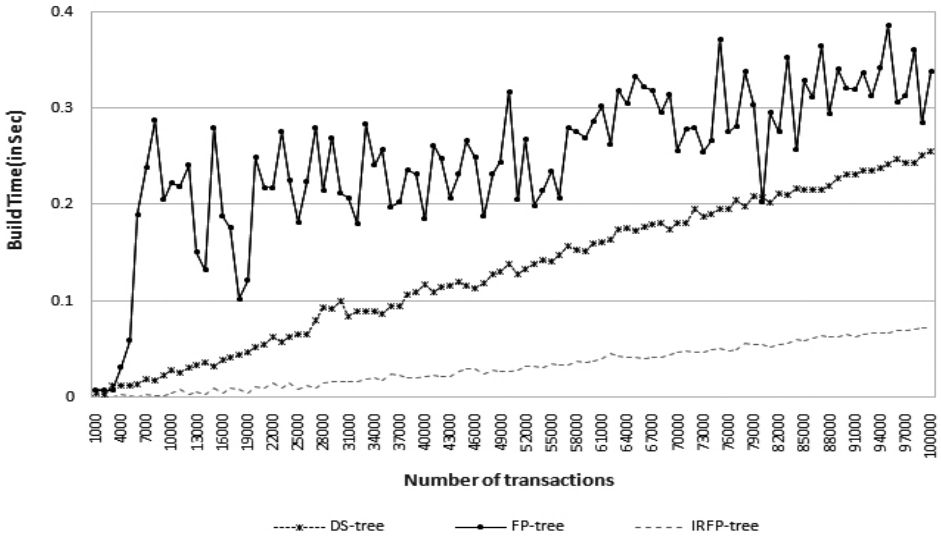


Fig. 11. Build time of each tree on various size of transactional databases

줄여줄 수 있고, growth를 통한 패턴 분석에 사용되는 노드의 수를 감소시킴으로써 growth의 메모리 자원 소비를 줄이고 그 속도를 향상시키는데 도움이 될 것이다.

Fig. 11은 실험 데이터베이스를 이용해 트리를 생성할 때 걸리는 시간을 그래프로 표현한 것이다.세로축은 시간을 나타내고 가로축은 사용된 트랜잭션의 수를 나타낸다. Fig. 4의 그래프에서 보여지는 바와 같이 기존에 가장 빠른 성능을 보이는 DS-tree에 비해서도 IRFP-tree 알고리즘을 이용한 트리 생성 속도가 더 빠르고 트랜잭션 크기가 커질수록 트리 생성 속도에서 큰 차이를 보이고 있다.

이처럼 우리가 제시한 IRFP-tree는 기존의 트리들에 비해 노드의 압축률을 증가시켜주며 그 처리속도 또한 감소시켜줄 수 있다는 것을 실험을 통해 확인하였다.

6. 결론 및 향후 연구과제

IRFP-tree는 교집합 규칙이라는 새로운 트리 생성 패러다임을 이용하여 생성되는 노드수를 최소화 하고 그 처리시간 또한 줄여줄 수 있다. 규칙 기반의 알고리즘이기 때문에 트리 생성 후 새로운 트랜잭션이 생성된 경우 추가 처리가 가능하여 스트리밍 데이터에서도 유용하게 사용될 수 있으며, 또한 DRFP-tree와 같이 트리를 생성하며 각 노드를 집합으로 표현하는 것 역시 가능하다. 이러한 장점을 통해 IRFP-tree는 빅 데이터에서 빠른 데이터 분석을 돕고 메모리에 의존적인 FP-tree의 문제를 일부 해결 가능할 것으로 보인다.

현재는 트리 생성 알고리즘만을 제시하였기 때문에 트리 생성 후 패턴을 분석하는 Growth 알고리즘은 기존의 FP-Growth를 이용하고 있다. 하지만 IRFP-tree에 적합한 Growth 방법을 이용하여 빅 데이터 분석을 위한 좀 더 빠르고 효율적인 빈도 패턴 분석이 가능하도록 새로운 Growth 방법에 대해 연구하고 있으며 실험 단계에 있다. 이러한 빈도 패턴을 표현하는 트리 생성 방식의 제시 및 새로운 Growth방법의 제시를 통해 좀 더 효과적이고 효율적인 빅 데이터 분석에 기여할 수 있을 것이다.

References

[1] R. Agrawal, T. Imieliski, and A. Swami, “Mining association rules between sets of items in large databases,” in *Proc. ACM SIGMOD Int. Cont. Manage. Data*, pp.207-216, 1993.

[2] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proc. Int. Conf. Very Large Data Bases*, pp.487-499, 1994.

[3] Jiawei Han, Jian Pei, and Yiwen Yin, “Mining Frequent Patterns without Candidate Generation,” in *ACM-SIGMOD, Dallas*, 2000.

[4] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S. Yu, “Mining Frequent Patterns in Data Streams at Multiple Time Granularities; Data Mining: Next Generation Challenges and Future Directions,” *AAAI/MIT*, 2003.

[5] O. R. Zaïane and M. El-Hajj, “COFI Approach for Mining Frequent Itemsets Revisited,” *Proc. ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery*, pp.70-75, 2004.

[6] O. R. Zaïane and M. El-Hajj, “Cofi-tree mining: A new approach to pattern growth with reduced candidacy,” in *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementation(FIMI'03)*, 2003.

[7] M. Adnan and R. Alhajj, “DRFP-tree: Disc-resident frequent pattern tree,” *Appl. Intell.*, Vol.30, No.2, pp.84-97, 2009

[8] M. Adnan and R. Alhajj, “A bounded and Adaptive Memory-based Approach to Mine Frequent Patterns From Very Large Databases,” *IEEE Transactions on Systems, Man, and*

Cybernetics, Part B(2011), pp.154-172.

[9] C. K. -S. Leung, Q. I. Khan, and T. Hoque, "CanTree:A Tree Structure for Efficient Incremental Mining of Frequent Patterns," *Proc. IEEE Int. Conf. Data Mining*, pp.274-281, 2005.

[10] C. K. -S. Leung, and Q. I. Khan, "DSTree:a tree structure for the mining of frequent sets from data streams," in *Proc. IEEE ICDM*, pp.928-932, 2006.

[11] G. Liu, H. Lu, J. X. Yu, W. Wang, and X. Xiao, "AFOPT: An efficient implementation of pattern growth approach," in *Proc. FIMI*, 2003.

[12] Cheung, William and Osmar R. Zaiane, "Incremental mining of frequent patterns without candidate generation or support constraint," *DatabaseEngineering and Applications Symposium, Proceedings. Seventh International. IEEE*, pp.111-116, 2003.

[13] B. Goethals, "Memory issues in frequent itemset mining," in *proc. ACM SAC*, pp.530-534, 2004.

[14] R. Vaarandi, "A breadth-first algorithm for mining frequent patterns from event logs," in *Proc. IEEE INTELLCOMM*, pp.274-281, 2004.

[15] G. Buehrer, S. Parthasarathy, and A. Ghoting, "Out-of-core frequent pattern mining on a commodity PC," in *Proc. 12th ACM SIGKDD Int. Conf.* KDD, pp.86-95, 2006.

이 정 훈

e-mail : leeye123@naver.com

2005년 동국대학교 컴퓨터공학과(학사)

2007년 동국대학교 컴퓨터공학과(석사)

2011년 동국대학교 컴퓨터공학과(박사)

2011년~2012년 동국대학교 산업기술연구원

2014년~2015년 동국대학교 IT융합교육센터

2015년~현 재 동국대학교 전산원 컴퓨터해킹보안전공 교수

관심분야 : SW품질평가, 빅데이터, 데이터마이닝