Vol.5, No.12 pp.623~634 pISSN: 2287-5905 https://doi.org/10.3745/KTSDE.2016.5.12.623

PPFP(Push and Pop Frequent Pattern Mining): A Novel Frequent Pattern Mining Method for Bigdata Frequent Pattern Mining

Lee Jung-Hun† · Min Youn-A††

ABSTRACT

Most of existing frequent pattern mining methods address time efficiency and greatly rely on the primary memory. However, in the era of big data, the size of real-world databases to mined is exponentially increasing, and hence the primary memory is not sufficient enough to mine for frequent patterns from large real-world data sets. To solve this problem, there are some researches for frequent pattern mining method based on disk, but the processing time compared to the memory based methods took very time consuming. There are some researches to improve scalability of frequent pattern mining, but their processes are very time consuming compare to the memory based methods. In this paper, we present PPFP as a novel disk-based approach for mining frequent itemset from big data; and hence we reduced the main memory size bottleneck. PPFP algorithm is based on FP-growth method which is one of the most popular and efficient frequent pattern mining approaches. The mining with PPFP consists of two setps. (1) Constructing an IFP-tree: After construct FP-tree, we assign index number for each node in FP-tree with novel index numbering method, and then insert the indexed FP-tree (IFP-tree) into disk as IFP-table. (2) Mining frequent patterns with PPFP: Mine frequent patterns by expending patterns using stack based PUSH-POP method (PPFP method). Through this new approach, by using a very small amount of memory for recursive and time consuming operation in mining process, we improved the scalability and time efficiency of the frequent pattern mining. And the reported test results demonstrate them.

Keywords : Big Data Mining, Frequent Pattern Mining, FP-Growth, IFP-Tree, PPFP

PPFP(Push and Pop Frequent Pattern Mining): 빅데이터 패턴 분석을 위한 새로운 빈발 패턴 마이닝 방법

이 정 훈† · 민 연 아††

요 약

현존하는 빈발 패턴 마이닝 방법은 대부분 시간 효율성을 목표로 하고, 물리적 메모리 사용에 매우 의존적이다. 하지만 빅데이터 시대가 도래함에 따라 실제 세상의 데이터베이스는 급속도로 증가하고 있으며, 그에 따라 기존의 방법으로 현실적인 거대한 양의 데이터를 마이닝하기에 물리적 메모리 공간이 부족한 실정이다. 이러한 문제를 해결하기 위해, 빈발 패턴 마이닝의 메모리 의존성을 줄이기 위한 보조저장장치 기반의 연구들이 진행되었으나, 메모리 기반의 방법들에 비해 처리 시간이 너무 많이 소비된다는 한계가 있었다. 따라서 확장성을 가지며, 기존의 디스크 기반의 방법들에 비해 시간효율성을 높인 새로운 빈발 패턴 마이닝이 필요하게 되었다. 본 논문에서는 빅데이터로부터 빈도 아이템 집합들을 마이닝하기 위해 메모리와 디스크를 함께 사용하는 스택 기반의 새로운 접근법인 PPFP 알고리즘을 제안하였다. PPFP는 빈발 패턴 마이닝 접근법 중 가장 인기 있고 효율적인 접근법 중 하나인 FP-growth를 기반으로 하고 있다. PPFP 마이닝 방법은 다음과 같이 두 단계로 진행된다. (1) IFP-tree 구축: FP-tree를 생성한 후, 새로운 인덱스 번호 부여 방법으로 FP-tree의 각 노드에 인덱스 번호를 부여하고, 이 인덱스 번호가 부여된 FP-tree(IFP-tree)를 테이블로 변환하여(IFP-table) 디스크에 저장한다. (2) PPFP 알고리즘을 이용한 빈발 패턴 마이닝: 스택 기반의 PUSH-POP 방식으로 패턴을 확장시켜 나가며 빈발 패턴을 마이닝한다. 이러한 방식을 통해 메모리 기반의 방법에 비해 반복적으로 많은 시간이 소모되는 연산에 매우 적은 양의 메모리를 활용하여 확장성과 함께 시간효율성 또한 향상시킬 수 있었다. 그리고 기존의 연구 방법들과 비교 실험을 통해 새로운 알고리즘의 성능을 증명하였다.

키워드 : 빅데이터 마이닝, 빈발 패턴 마이닝, FP-growth, IFP-tree, PPFP

† 정 회 원 : 동국대학교 전산원 강의전담교수
†† 비 회 원 : 가천대학교 SW중심대학사업단 초빙교수
Manuscript Received : July 29, 2016
Accepted : August 9, 2016
* Corresponding Author : Lee Jung-Hun(leeye123@naver.com)

1. 서 론

디지털 경제의 확산으로 규모를 가늠할 수 없을 정도로 거대한 양의 정보와 데이터가 생산되는 빅데이터 시대가 도래하였다. 이와 더불어 방대한 규모의 데이터에서 의미 있는 정보만을 추출해 내는 기술이 요구되고 있다.

거대한 데이터에서 아이템들의 연관성을 찾는 연관 규칙에 대한 개념은 1993년 Agrawl. et al.[1]에 의해 처음 소개되었다. 연관 규칙 분석에 대한 연구가 활발히 진행되며, 그 중 Apriori[2] 방법은 그 이전의 빈발 패턴 마이닝에 필요한 후보 집합의 규모를 줄여주는 중요한 역할을 하였다. 하지만 여전히 후보패턴들의 거대한 집합을 생성하는데 많은 비용이 소비되게 되었다.

Apriori 기반의 빈발 패턴 마이닝 방법들의 거대한 후보 집합 생성 비용을 줄이기 위해 등장한 방법으로, 최근 가장 활발히 연구되고 있는 FP-growth[4] 알고리즘이 있다.

FP-growth는 최근 가장 인기 있는 빈발 패턴 마이닝 기법 중 하나로, 빠른 시간 내에 효율적으로 빈발 패턴에 대한 분석을 가능하게 해준다. FP-growth는 트랜잭션 데이터베이스를 의미 있는 정보만 압축하여 표현한 FP-tree라는 데이터 구조물을 메모리에 생성하고, FP-growth를 통해 점진적으로 패턴을 추출하며, 두 번의 DB 전체 스캔으로 트리를 구성하기 때문에 데이터베이스의 스캔 횟수를 획기적으로 줄이고, 연관규칙에 사용되는 거대한 후보군(Candidate) 생성에 대한 문제를 해결하였다. FP-growth에 대한 연구도 활발히 진행되어, FP-growth에 사용되는 FP-tree 구조를 개선시키기 위한 COFI-tree[6], CATS-tree[7], CanTree[8], DS-tree[9], IRFP-tree[10] 등 다양한 트리 생성 알고리즘들이 제시되었고 FELINE[6], AFPIM[12] 등 다양한 마이닝 기법들이 연구되었다.

하지만 이러한 알고리즘들을 이용하여 현실의 데이터를 마이닝 하기에는 물리적 메모리 공간이 부족하다. Goerhals[16]와 Vaarandi[17], 그리고 Buehrer[18] 등은 실제 세상의 데이터 집합들로부터 연관 규칙을 마이닝 하기에 최근 CPU에서 사용 가능한 표준 물리적 메모리들로는 충분치 못하다는 연구 결과를 보였다. 이로 인해 단순히 시간효율성만을 강조하던 기존의 FP-growth의 연구에서 메모리 의존도를 낮추고 보조기억장치를 활용하여 확장성을 목표로 한 연구가 필요하게 되었다.

FP-growth의 확장성을 목표로 한 연구로 DRFP-growth[13]와 BFP-growth[14], SQL기반의 FP-growth 방법[15] 등이 있다. DRFP-growth는 메모리에서 처리 가능한 규모인 경우 FP-tree와 FP-growth를 사용하고, 메모리 영역을 벗어난 규모인 경우 보조기억장치로 확장한 DRFP-tree와 DRFP-growth를 적용시켜 메모리 의존성을 낮춘 방법이다. SQL 기반의 FP-growth 방법은 FP-growth 알고리즘을 보조기억장치만으로 처리 가능하도록 RDBMS상에서 SQL 기반으로 FP-growth 알고리즘을 수행하는 방법이다. 이들 모두 메모리 의존성을 줄이는 데는 의의가 있지만, FP-growth에 비해 시간효율성이 현저히 떨어져 그 실용성에 문제가 있다.

본 논문에서는 메모리를 적절히 활용하며 RDBMS에서 스택을 이용하여, 기존의 보조기억장치를 활용해 확장성을 높인 방법들에 비해 확장성은 유지하며 시간효율성을 향상시킨 새로운 빈발 패턴 마이닝 알고리즘을 제시하고, 실험을 통해 그 성능을 증명하였다.

2. 관련 연구

이 장에서는 본 논문의 새로운 알고리즘을 제시하기에 앞서 기존의 빈발 패턴 마이닝 기법 중 현재 가장 활발히 연구되어 있는 FP-growth 알고리즘과, 메모리 기반의 여타 빈발 패턴 마이닝 방법의 한계를 극복하기 위해 연구된 RDBMS 기반의 빈발 패턴 마이닝 방법인 SQL기반의 FP-growth방법에 대해 살펴본다.

2.1 FP-growth[4]

FP-growth[4]는 기존의 Apriori 연관규칙 알고리즘에 비해 빈발패턴을 마이닝하는데 좀 더 효과적인 방법이다. FP-growth는 두 단계로 구성된다. 첫 번째 단계는 FP-tree를 생성하는 것이다. FP-tree는 효율적인 빈발 패턴 마이닝을 위해 트랜잭션 데이터베이스 내의 빈발 아이템 집합들을 좀 더 압축시켜 FP-tree(a frequent pattern tree)라 불리는 새로운 데이터 구조에 저장하는 것이다. 두 번째 단계는 FP-tree를 이용하여 패턴을 마이닝하는 단계이다. 길이가 1인 패턴(최초의 접미 패턴)들로부터 시작하여 그것의 조건부 패턴 베이스(conditional pattern base)만을 고려하여 그것의 조건부 FP-tree(conditional FP-tree)를 반복적으로 생성하며 패턴을 찾아낸다.

1) FP-tree의 생성

FP-Tree는 단 두 번의 데이터베이스 스캔만으로 빈발 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 빈도수의 역순으로 정렬하며 트리를 구성한다.

아래 FP Tree를 생성하는 간단한 예가 나타나 있다. Table 1은 FP-tree를 생성하기 위한 예제 트랜잭션 데이터베이스이다. 이 트랜잭션 데이터베이스는 다섯 개의 트랜잭션을 가지고 있다. 트리 생성할 때 최소지지도(minimum

Table 1. Simple Transaction Database with Ordered Frequent Items($\xi=2$)

TID	Items	Ordered Frequent Items
T001	T, E, K, S, C	T, E, K, S
T002	T, E, K, J, B	T, E, K, J
T003	T, D, J, F	T, J
T004	E, H, J, P	E, J
T005	T, E, K, R, S	T, E, K, S

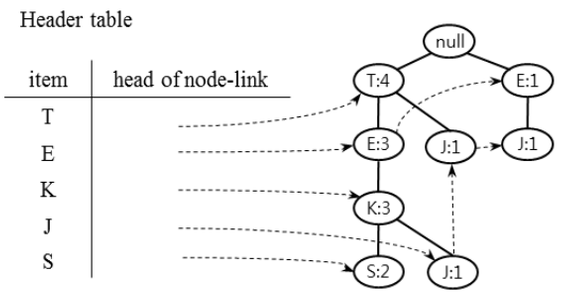


Fig. 1. The FP Tree Built Based on the Data in Table. 1

support threshold) ξ=2로 한다. 각 행은 한 개의 트랜잭션 내에 동시에 발생한 아이템들의 집합으로 구성되어 있고, TID로 구분된다.

첫 번째 DB 스캔을 통해 트랜잭션 데이터베이스 내의 모든 아이템들의 빈도수를 계산하여 빈도수의 역순으로 정렬시켜 빈발 아이템 리스트를 생성한다. Table 1의 전체 트랜잭션 내 빈발 아이템 리스트(frequent items list) L은 <(T:4), (E:4), (K:3), (J:3), (S:2)>가 된다.

두 번째로 우선 트리의 루트 노드를 생성한다. 그 후 데이터베이스를 두 번째로 스캔한다. 데이터베이스를 스캔하며 첫 번째 트랜잭션인 T001의 아이템 <T, E, K, S, C>를 가져와서 빈발 아이템을 L의 순서대로 정렬시키면 아이템 리스트 <T, E, K, S>가 된다. 이 아이템 리스트로 트리의 첫 번째 분기<(T:1), (E:1), (K:1), (S:1)>을 만든다. 그 다음 두 번째 트랜잭션인 T002을 읽어와 정렬된 빈발 아이템 리스트<T, E, K, J>를 생성한다. 여기서 <T, E, K>는 트리에 이미 존재하는 경로인 <T, E, K, S>에 포함되기 때문에 공유된 노드는 빈도수만 1 증가시켜주고 (J:1)만을 새로운 노드로 생성하여 (K:2)의 자식 노드로 링크시킨다. 이와 같은 방법으로 전체 트랜잭션의 빈발 아이템 리스트를 만들어 가며 트리를 확장시키면 Fig. 1의 FP-tree가 생성된다. 트리의 효과적인 탐색을 위하여 Fig. 1의 Header테이블을 생성하고 헤더 테이블이 트리 내 각 아이템들이 처음 발생한 지점을 링크시킨다. 모든 트랜잭션 데이터베이스의 스캔이 끝난 후 트리의 동일한 아이템 노드들을 링크로 연결시켜 헤더테이블의 아이템 링크를 따라가면 트리 내 모든 동일 아이템들이 탐색될 수 있다.

트리 생성 알고리즘은 Pseudo Code 1에 기술되어 있다.

2) FP-growth 알고리즘을 이용한 빈발 패턴 추출

주어진 FP-tree를 이용하면 분할-정복 방식으로 빈발 패턴을 추출할 수 있다. 이 단계는 길이가 1인 패턴(헤더 테이블에 존재하는 단일 아이템에 해당하는 패턴)으로부터 시작하여 그것의 조건부 패턴 베이스(conditional pattern base, 해당 패턴을 하위 패턴으로 포함하여 함께 발생하는 아이템들의 집합으로 구성된 부분 데이터베이스)를 생성하여 그 조건부 FP-tree(conditional FP-tree)를 생성하여 재귀적으로 패턴을 구해내는 방법이다.

FP-growth 알고리즘을 적용하여 Fig. 1에 나타나 있는 FP-tree의 빈발 패턴을 추출하는 방법은 다음과 같다. 우선 Fig. 1의 Header table 가장 하단의 아이템인 (S:2)부터 시작한다. 처음으로 빈발 패턴 (S:2)가 추출되고, 헤더 테이블의 링크를 따라가 보면 (S:2)노드와 연결된 경로 <T:4, E:3, K:3, S:2> 를 찾을 수 있다. 이 경로가 <S>노드와 함께 발생한 빈도수는 해당 경로에서 <S>가 발생한 빈도수인 2번이 된다. <S>가 발생한 상위 경로를 <S>가 발생한 빈도수로 수정하면 <T:2, E:2, K:2>가 되고, 이것을 조건부 패턴 베이스(conditional pattern base)라 한다. 이를 이용하여 조건부 트리를 구성하면 Fig. 2에 나타나 있는 <S>의 조건부 FP-tree가 된다. 새로 만들어진 <S>의 FP-tree 헤어 테이블의 최 하단 아이템인 <K:2>부터 다시 마이닝을 시작하면,

Pseudo Code 1. FP-tree Construction

Algorithm 1 FP-tree construction
Input: A transaction database DB, a minimum support threshold ξ
Output: Its frequent pattern tree, FP-tree
Method: The FP-tree is constructed in the following steps.

```
1.  Scan the transaction database DB once. Collect the set of frequent items F and their supports.
2.  Sort  F in support descending order as L, the list of frequent items.
3.  Create  a root of an FP-tree, T, and lable it as 'null'.
4.  For each transaction Trans in DB do
5.      Select and sort the frequent items in Trans according to the order of L.
6.      Let  the sorted frequent item list in Trans be [p|P], where p is the first item and P is the last.
7.      call  insert_tree([p|P], t)
8.  end for

insert_tree([p|P], t)
1.  if  T has a child n such that N.item-name = p.item-name then
2.          Increment n's count with 1
3.  else
4.          Create a new node n and let its count be 1, its patent link be linked to t, and its node-link be linked
        to the nodes with the same item-name via the node-link structure.
5.  end if
6.  if P ≠ Ø then
7.          Call insert-tree(P, N)
8.  end if
```

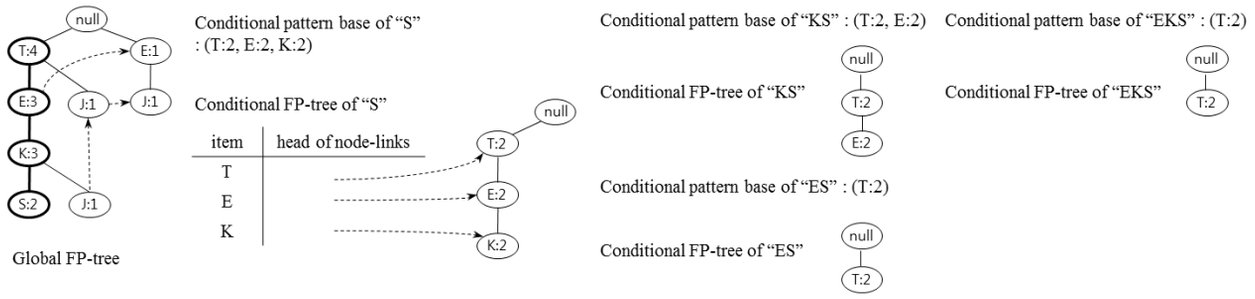


Fig. 2. Example of Mining Process Using FP-tree

Table 2. Mining of All-patterns by Creating Conditional (Sub)-Pattern Bases

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns
S	{(T:2, E:2, K:2)}	{(T:2, E:2, K:2)} | S	S:2, KS:2, ES:2, TS:2, EKS:2, TKS:2, TEKS:2, TES:2
J	{(T:1, E:1, K:1), (T:1), (E:1)}	{(T:2, E:2)} | J	J:3, EJ:2, TJ:2
K	{(T:3, E:3)}	{(T:3, E:3)} | K	K:3, EK:3, TK:3, TEK:3
E	{(T:3)}	{(T:3)} | E	E:4, TE:3
T	Ø	Ø	T:4

(KS:2) 패턴이 추출되고, <T:2, E:2> 조건부 패턴 베이스가 생성된다. 다시 이를 이용해 <EKS:2>이 조건부 패턴 베이스를 생성하면 (EKS:2) 패턴이 추출되고 <T:2>라는 조건부 패턴 베이스가 생성된다. 이처럼 재귀적으로 조건부 패턴 베이스를 찾아내 조건부 FP-tree를 생성해 나가며 빈발 패턴을 추출하면 Table 2의 네 번째 열에 해당하는 빈발 패턴이 추출된다. FP-growth의 알고리즘은 Pseudo Code 2에 나타나 있다.

FP-growth는 Apriori 기반의 방법들에서 비용이 많이 드는 후보 집합의 생성을 피하기 위해 만든 접근법으로, 그 이전의 연구에 비해 매우 빠른 속도로 빈발 패턴을 마이닝할 수 있다. FP-growth에 있어서 FP-tree는 트랜잭션 데이터베이스의 빈번하지 않은 아이템들을 우선 가지치기한 후 데이터를 압축시켜 주기 때문에 FP-growth를 진행할 영역을 줄여주고, 단 두 번의 스캔으로 트리를 구성하여 반복적인 데이터베이스 스캔 비용을 줄여준다. 이러한 특성으로 인해 필드에서 가장 인기 있는 빈발 패턴 마이닝 방법 중 하나이다. 하지만 트리의 특성상 메모리에 의존적이고, 반복적으로 생성되는 조건부 트리를 메모리에 유지시켜야 하기 때문에 현실 세계의 거대한 양의 트랜잭션 데이터베이스를 마이닝 하기에는 물리적 메모리 공간의 한계로 인한 어려움이 있다.

2.2 SQL 기반의 빈발 패턴 마이닝[15]

빈발 패턴 마이닝 문제가 제기된 이래로, 대부분의 빈발 패턴 마이닝 알고리즘들은 메모리 의존적으로 마이닝 속도와 정확성을 높이는 것을 목표로 하였다. 하지만 빅데이터 시대가 도래하며, 현실적으로 거대한 양의 마이닝을 처리하기에 인 메모리(in-memory) 기반의 알고리즘들은 한계가 있다. SQL 기반의 빈발 패턴 마이닝은 빈발 패턴 마이닝의 시간 효율성과 정확성을 향상시키기 위한 이전의 연구들과는 차별되게 확장성(scalability)을 향상시키기 위한 연구를 진행하였다. [15]의 연구는 기존의 빈발 패턴 마이닝에서 크게

Pseudo Code 2. FP-growth

Algorithm 2 FP-growth(*Tree*, α) // Mining frequent patterns with FP-tree
Input: An FP-tree *Tree*, a set of prefix items α and a minimum support threshold ξ.
Output: The complete set of frequent patterns

```
1.  if Tree contains a single path P then
2.      for each combination β of the nodes on the path P do
3.          Generate pattern β∪α with support = minimum support of the nodes in β
4.      end for
5.  else
6.      for each item a_i in the header table of Tree do
7.          Generate pattern β := a_i∪α with support := a_i.support
8.          Construct β's conditional pattern base and then β's conditional FP-tree Tree_β
9.          if  Tree_β ≠ Ø then
10.             Call FP-growth(Tree_β, β)
11.         end if
12.     end for
13.  end if
```

Pseudo Code 3. Algorithm for Constructing Table FP

```
Algorithm 3 construction table FP
Input: a transferred transaction table T'
Output: a table FP
 1.  for items with the first identical tid in the table T'
 2.        insert into the table FP
 3.  for items with the other identical tids in the table T'
 4.        insertFP(items);

insertFP(items)
 1.  count := 1;
 2.  curpath := null;
 3.  if FP has an item f = i1(the first item in the items) and f.path = null
 4.      for   each item ik in the items
 5.          insert ik into the table FP;
 6.          curpath += ik;
 7.  else
 8.      for   each item ik in the items
 9.          if   FP has not an item f = ik
10.              insert ik into the table FP;
11.          else
12.              if f.path != ik.path
13.                  insert ik into the table FP;
14.              else
15.                  curcount = ik.count+ 1;
16.                  update the table FP;
17.      curpath += ik;
```

이슈가 되던 FP-growth를 RDBMS에 통합시키는 방법을 제시하였다. RDBMS는 메모리에 비해 물리적 용량이 클뿐더러 확장성이 높기 때문에 기존에 메모리상에서 처리되지 못한 큰 데이터에 대한 마이닝을 가능하게 해 준다. SQL을 이용한 빈발 패턴 마이닝의 방법은 FP-growth와 마찬가지로 FP-tree를 테이블화 시키는 것과, 만들어진 *table FP*를 이용해 빈발 패턴을 찾아내는 두 단계로 진행된다.

1) *FP 테이블*의 생성

FP-tree에 대응되는 *FP 테이블*를 구성하기에 앞서 트랜잭션 데이터베이스를 *table FP*를 구성하기에 적합한 형태로 변형시킨다. 우선 트랜잭션 데이터베이스의 각 트랜잭션 식별자(TID)와 트랜잭션 내 아이템(item)을 속성으로 하는 *table T*를 생성한다. 한 트랜잭션 내의 아이템들마다 하나의 열이 생성된다. 이처럼 생성된 *table T*에서 TID 순서대로 트랜잭션 별 최소 지지도를 만족시키는 아이템만 빈도수의 역순으로 정렬시키면 *table T'*가 생성된다.

Table 3은 Table 1의 간단한 트랜잭션 데이터베이스를 SQL 기반의 FP-growth 방법을 이용하여 빈발 패턴을 추출해내는 예이다. Table 3에서와 같이 우선 Table 1의 각 트랜잭션의 아이템마다 하나의 행으로 테이블에 입력하여 Table 3-(a)와 같이 *table T*를 생성한다.

*table FP*는 아이템(item), 빈도 수(count), 경로(path) 세 개의 속성으로 구성된다. 우선 *table T'* 내에 처음 나타나는 TID의 아이템들을 빈도수는 1, 경로는 null로 하여 *table FP*에 저장한다. 그 후 *table T'*의 각 트랜잭션 별 아이템들을 묶어 insertFP(items) 프로시저를 실행시킨다. insertFP() 프로시저에서는 입력된 아이템들이 *table FP* 내에 존재하는지 확인한다. 입력된 아이템이 테이블에 존재하지 않거나, 존재하지만 경로가 다른 경우 *table FP*에 새로운 행으로 저장한다. 만약 이미 테이블 내에 존재하고, 경로도 동일한 경우는 count만 1 증가시켜 준다. Table. 3-(c)는 이와 같은 방법으로 Table. 3-(b)의 T' 테이블을 이용하여 만든 *table FP*이다.

*table T'*로부터 *table FP*를 생성하기 위한 알고리즘은 Pseudo Code 3과 같다.

2) *table FP*로부터 빈발 패턴 마이닝

*table FP*로부터 빈발 패턴을 마이닝하기 위해, 우선 모든 빈발 아이템들의 집합인 *table F*를 생성한다. 이 빈발 아이템들을 이용하여, 우선 각 빈발 아이템 a_i에 대해 조건부 빈발 패턴 베이스 테이블인 *PB-*a_i 테이블을 생성한다. *PB-*a_i 테이블은 트랜잭션 식별자(TID), 아이템(item), 빈도 수(count) 세 개의 속성으로 구성된다. *PB-*a_i 테이블을 생성하기 위해 먼저 *table FP* 테이블에서 아이템이 a_i인 행(TID, count, path)들을 검색한다. 검색 결과의 각 행에 대해 우선 TID를 1로 하고, path를 각각의 아이템으로 분류하여 *PB-*a_i테이블에 저장한다. 이렇게 만들어진 *PB-*a_i 테이블이 만약 단일경로로 구성되어 있다면 그 조합을 패턴으로 찾아내고, 아니라면 해당 *PB-*a_i를 이용하여 table FP를 만들 때 사용했던 알고리즘인 Pseudo Code 3을 이용하여 조건부 *FP-*a_i 트리인 *ConFP-*a_i를 생성한다. FP테이블을 이용하여 패턴을 찾아내는 FP-growth 알고리즘은 Pseudo Code 4에 나타나 있다.

현재 가장 널이 사용되는 빈발 패턴 마이닝 방법 중 하나

Pseudo Code 4. Algorithm for Finding Frequent Patterns from Table FP

```
Algorithm 4 FindFP(table FP, table F) // Mining frequent patterns with table FP
Input: A table FP and table F(set of all frequent items)
Output: A table Pattern // Which consists of complete set of frequent patterns
 1.    if items in the table FP in a single path
 2.        combine all the items and suffix, insert into Pattern
 3.    else
 4.        for each item ai in the table F
 5.            construct table ConFP-ai
 6.            if ConFP-ai≠ Ø
 7.                call FindFP(ConFP-ai);
 8.            end if
 9.        end for
10.    end if
```

인 FP-growth를 RDBMS를 기반으로 통합한 이 연구는 기존의 시간효율성만 고려한 방법들과 차별성을 보인다. 트랜잭션 데이터베이스를 SQL을 이용해 테이블상에 트리 구조를 평면적으로 구현하고, 그렇게 만들어진 FP 테이블을 SQL을 이용하여 빈발 패턴을 마이닝하여 빈발 패턴 마이닝의 공간적인 제약을 줄여주었다.

앞서 살펴본 바와 같이 FP-growth는 시간 효율성 측면에서 매우 뛰어난 성능을 보인다. 뿐만 아니라 기존의 Apriori 방식의 방대한 양의 후보 집합을 생성하는 방법들에 비해 확장성 측면에서도 좋은 성능을 보여준다. 하지만 FP-growth 과정에서 빈발 패턴을 마이닝하기 위해 메모리상에 조건부 패턴 트리를 누적하여 상주시켜야 하기 때문에 빅데이터의 경우 물리적 메모리의 한계로 인해 마이닝이 어려울 수 있다. 이러한 물리적 메모리 공간의 한계를 극복하기 위해 SQL기반의 빈발 패턴 마이닝 방법이 제시되었으나, 확장성 측면에서는 만족할 만한 성능을 보이지만 기존의 메모리 기반의 방법들에 월등히 많은 시간이 소비되기 때문에 실효성이 없어 보인다. 따라서 본 논문에서는 메모리와 RDBMS를 적절히 활용하여 SQL기반의 방법에 비해 시간효율성을 높이고 FP-growth 방법에 비해 확장성을 높인 새로운 빈발 패턴 마이닝 방법을 제시한다.

3. PPFP (Push and Pop Frequent Pattern mining method)

현실적인 빅데이터를 마이닝하기 위해 우리는 PPFP라는 스택을 기반으로 한 새로운 빈발 패턴 마이닝 알고리즘을 제시하였다.

PPFP 알고리즘은 기존의 FP-tree에 스택 기반의 마이닝에 필요한 속성을 추가하여 확장시킨 *IFP-tree*라 불리는 변형된 FP-tree를 이용하여 트랜잭션 데이터베이스를 압축시키고, 생성된 *IFP-tree*를 테이블화 시킨 *IFP-table*을 생성한 후에 스택을 기반으로 SQL을 이용해 빈발 패턴을 확장시켜 나가며 마이닝하는 새로운 빈발 패턴 마이닝 방법이다.

3.1 IFP-tree 테이블의 생성

PPFP 빈발 패턴 마이닝을 위해 트랜잭션 데이터베이스를 압축시켜 필요한 정보만을 가지는 압축된 데이터 구조가 필요하다. 이를 위해 우리는 현재 가장 인기 있는 압축된 데이터 구조 중 하나인 FP-tree를 이용하였다. FP-tree는 압축된 데이터 정보를 가지고 있는 정교한 트리 구조일 뿐만 아니라, 트리 구성 시 최소지지도를 만족시키지 못하는 아이템들을 가지치기하여 불필요한 검색 영역을 줄일 수 있다.

FP-growth는 bottom-up 방식으로 특정 아이템과 연결된 모든 상위 경로를 수집하여 조건부 패턴 베이스를 생성하여 조건부 FP-tree를 구축해 나가며 빈발 패턴을 마이닝한다. 하지만 우리가 제시하는 스택 기반의 PPFP 마이닝 방법은 top-down 방식으로 특정 패턴의 하위 경로들을 수집하여 조건부 패턴 베이스를 생성한 후 스택에 PUSH-POP 방식을 통해 패턴을 확장시켜 나간다. 이처럼 PPFP 알고리즘을 이용하여 마이닝을 하기 위해서는, 특정 패턴의 하위 경로를 테이블화 되어 있는 트리 구조에서 SQL을 이용해 검색해 올 수 있어야 한다. 하지만 단순히 FP-tree를 테이블화 시킨 것만으로는 테이블 상에서 특정 패턴의 하위 집합의 범위를 구하기 어렵다. 이를 위해 우리는 새로운 인덱스 번호 부여 방법을 사용하여 FP-tree의 각 노드에 인덱스 번호를 부여함으로써 FP-tree를 디스크상에 저장한 후에도 간단히 특정 패턴의 하위 경로 집합을 구할 수 있도록 하였다.

PPFP 마이닝 방법을 위해 FP-tree의 특정 노드, 혹은 특정 패턴의 하위 경로를 모두 구할 수 있도록 인덱스 번호가 부여되어야 한다. 이를 위해 우리는 FP-tree의 각 노드에 현재 인덱스 번호를 의미하는 *C-Index* (Current Index Number)와 현재 노드를 포함한 하위 집합 노드의 시작 인덱스를 의미하는 *L-Index* (Lowest Index Number)를 부여한다. Fig. 3은 Fig. 1의 예제 트리에 인덱스를 부여하는 방법을 순차적으로 보여준다. Fig. 3에서 인덱스 부여가 끝난 최종 트리를 보면, <T> 노드의 인덱스 번호는 *C-Index:6*, *L-Index:1*이다. 이 인덱스 번호는, <T>노드를 포함한 <T>노드의 하위 집합은 *C-Index:*1-6인 노드들을 포함한다는 것을 의미한다. 마찬가지로 <E>노드를 포함한 <E>노드의 하위 집합은 두

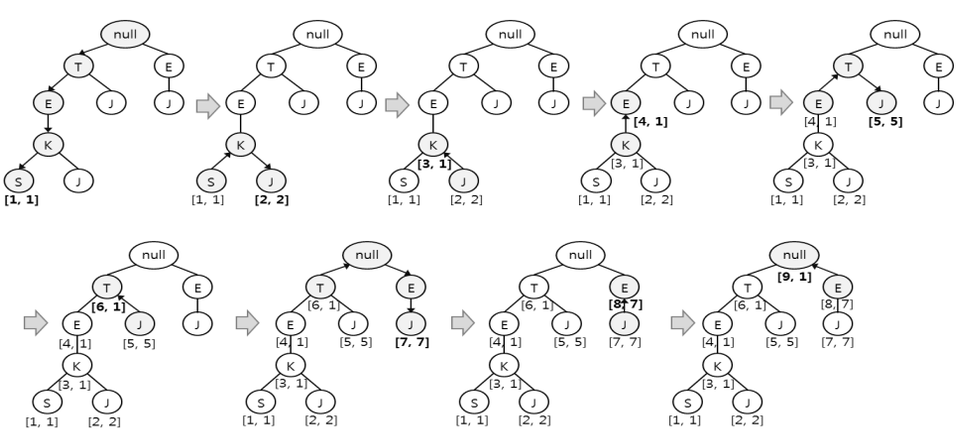


Fig. 3. The Process of Assign Index Number for Each Node

Pseudo Code 5. Algorithm for Assign Index Number for FP-tree

```
Algorithm 5 generation IFP-tree // Assign index number for each node in FP-tree.
Input: FP-tree
Output: IFP-tree //Indexed FP-tree
 1.   c_node = null; // from root node
 2.   c_index = 0; // current node;
 3.   call searchNextNode(c_node);

searchNextNode(c_node)
 1.   if there is a c_node.child_node which didn't be assigned a c_index number
 2.        c_node = the node with the most  frequent item among c_node's child nodes
 3.   else
 4.        c_node.c_index = ++c_index;
 5.        if c_node.child_node ≠ Ø
 6.             c_node.l_index = c_index;
 7.        else   // every c_node.child_node's c_index were assigned
 8.             c_node.l_index = the lowest c_index number among the c_node's sub-tree nodes;
 9.        c_node=c_node.patrent_node;
10.      call searchNextNode(c_node);
```

개의 <E> 노드의 인덱스 번호 범위의 합인 *C-Index*:1-4, 7-8의 노드들를 포함한다. 이처럼 인덱스 번호 만으로 특정 노드의 하위 집합을 식별할 수 있도록 인덱스번호를 부여한다. Pseudo Code 5는 FP-tree에 인덱스 번호를 부여하는 알고리즘이다.

Fig. 4는 Table 1의 트랜잭션 데이터베이스를 이용하여 구성한 FP-tree에서 Pseudo Code 5의 알고리즘으로 각 노드에 인덱스 번호를 부여하여 *IFP-tree*를 생성하고 이를 테이블화 시킨 것이다. FP-tree에서 인덱스 부여 절차가 끝나면 이를 *IFP-tree*라 하고, 이를 테이블에 저장하여 Fig. 4-(b)와 같이 *IFP-table*을 생성한다. PPFP 마이닝 단계에서는 Fig. 4-(b)를 이용하여 특정 패턴의 *L-Index*부터 *C-Index*까지 범위에 포함하는 노드들을 검색하여 조건부 패턴 베이스를 생성할 수 있게 된다.

3.2 PPFP 알고리즘

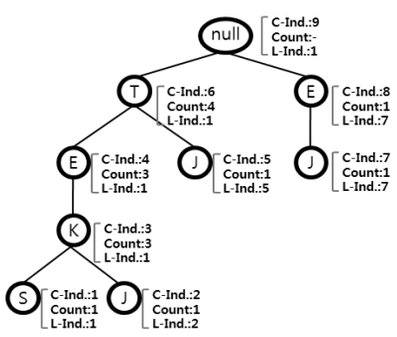
이 장에서는 앞서 구축한 IFP-table을 이용하여 반복적인 조건부 FP-tree의 생성 없이 빈발 패턴을 마이닝하는 방법에 대해 알아본다. PPFP 알고리즘은 본 논문에서 제시하는 빈발 패턴 마이닝 기법이다. FP-growth가 반복적인 조건부 FP-tree를 생성해 나가며 빈발 패턴을 마이닝하는데 비해 PPFP 알고리즘은 단 하나의 스택을 이용하여 푸쉬 팝(Push-Pop) 방식으로 순차적으로 빈발 패턴을 마이닝하는 방법이다.

Fig. 4는 Fig. 3에서 생성된 IFP 테이블을 이용해 PPFP 알고리즘을 적용시킨 예이다.

PPFP 마이닝을 위해 우선 하나의 조건부 FP-stack인 (Conditional FP-stack) *CFP-stack*을 생성한다. Fig. 3의 헤더 테이블 내에 존재하는 빈발 아이템 리스트를 *CFP-stack*에 입력한다. 그 후 푸쉬 팝(Push-Pop) 방식의 빈발 패턴 마이닝을 반복적으로 실행한다. 푸쉬 팝 방식의 첫 번째로 우선 *CFP-stack*의 가장 마지막에 입력된 아이템인 <S:2>를 POP한다. POP된 <S:2>를 빈발 패턴 테이블(frequent pattern table)에 입력한다. 그 후 <S:2>가 포함된 하위 집합들의 아이템들을 가져온다. <S:2>는 하위 집합이 없기 때문에 다시 *CFP-stack*의 마지막 아이템 <J:3>을 호출한다.

<J:3>을 빈발 패턴 테이블에 입력한다. 그리고 <J:3>이 포함된 모든 하위 경로의 아이템들을 가져온다. <S:2>와 마찬가지로 <J:3> 역시 하위 경로가 없기 때문에 CFP-stack의 다음 아이템인 <K:3>을 POP한다.

<K:3>을 빈발 패턴 테이블에 입력한다. 그리고 *IFP-table* 검색을 통해 <K:3>의 하위 빈발 아이템들을 가져와



(a) IFP-tree

C-Index	item	count	L-Index
1	S	2	1
2	J	1	2
3	K	3	1
4	E	3	1
5	J	1	5
6	T	4	1
7	J	1	7
8	E	1	7
9	null	-	1

(b) IFP-tree table

frequent item	count
T	4
E	4
K	3
J	3
S	2

(c) Header table

Fig. 4. Construction of IFP-table

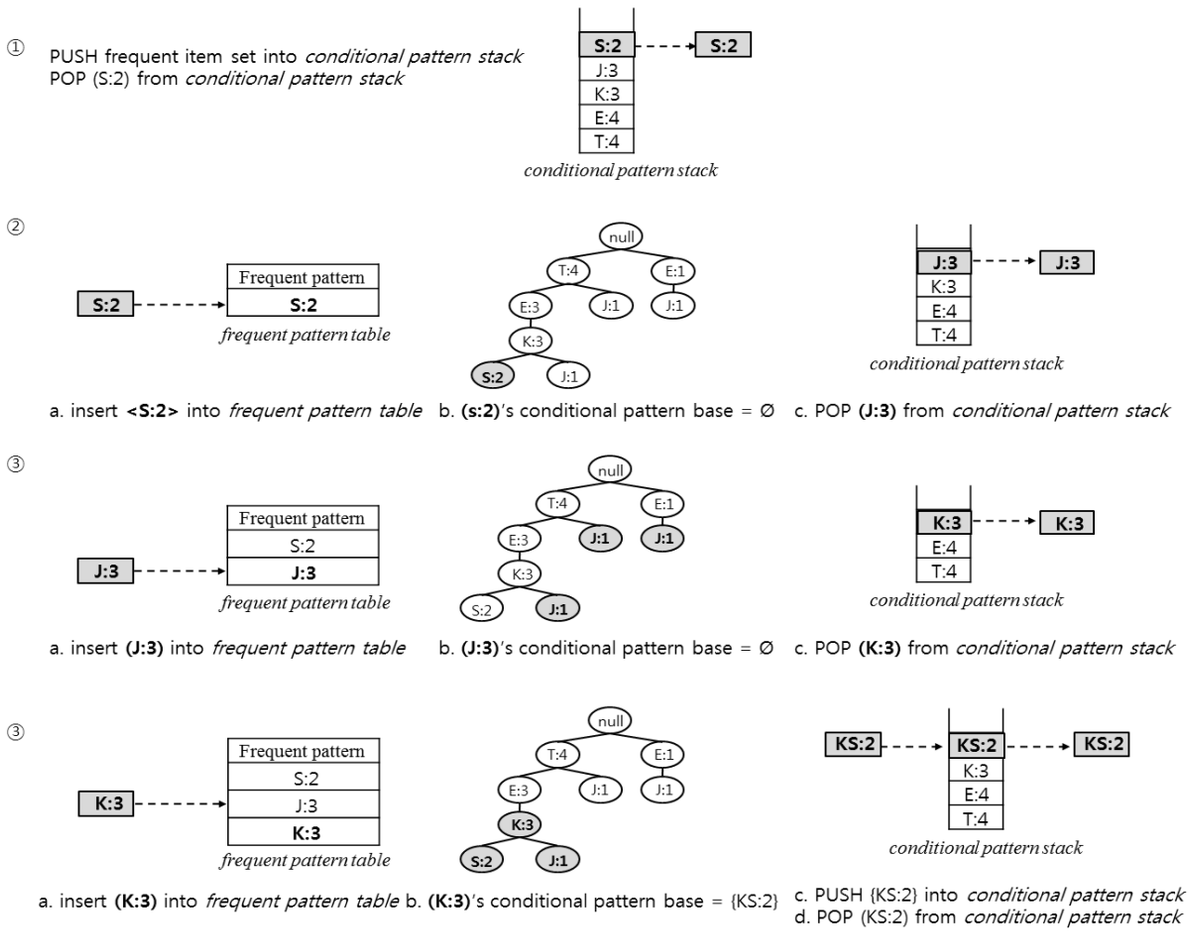


Fig. 5. Simple PPFP Example with Table.1

최소지지도 이상의 아이템만을 선택하여 조건부 패턴 베이스(Conditional pattern base)를 생성한다. <K:3>의 하위 경로의 아이템은 {J:2, S:1}이 있다. 하지만 {S:1}은 최소지지도를 만족시키지 못하기 때문에 {J:2}만이 conditional pattern base가 된다. conditional pattern base가 있기 때문에 앞선 경로를 포함하여 {KJ:2}를 다시 CFP-stack에 입력한다.

Fig. 5는 Fig. 4의 *IFP-table*을 이용하여 PUSH-POP 방식으로 빈발 패턴을 마이닝하는 단계를 도식화하여 보여준다. 여기서 *IFP-table*은 설명의 편의를 위해 FP-tree로 대체하여 나타내었다.

Table 4는 Fig. 4의 *IFP-table*을 이용하여 PPFP방식으로 마이닝할 경우 각 단계별로 POP되는 패턴과 POP되는 패턴으로 파생되어 생성되는 조건부 패턴 베이스(conditional pattern base), 그리고 이를 *CFP-stack*에 입력시킨 후 *CFP-stack*의 변화에 대해 테이블로 표현한 것이다. 이 예제를 이용하여 최종적으로 나타나는 빈발 패턴들은 전체 POP된 패턴들과 같다.

예제를 통해 살펴본 바와 같이, FP-growth가 FP-tree를 이용해 조건부 패턴 베이스를 생성한 후 이를 이용하여 조건부 FP-tree를 메모리상에 쌓아 나가며 빈발 패턴을 구하는 반면, PPFP는 IFP-table에서 SQL을 통해 조건부 패턴 베이스를 구해 스택에 PUSH한 후 하나씩 POP해 나가며 빈발 패턴을 구하고 있다. *IFP-table*에만 존재하는 새로운 인덱스 방식으로 트리를 테이블화 하였기에 테이블에 존재하는 트리 정보로 하위 테이블을 구해내는 것이 가능해 졌다.

*IFP-table*에서 조건부 패턴 베이스를 구하는 방식은 다음과 같다.

Query 1은 *IFP-table*로부터 특정 경로 $[a_1, \cdots, a_i]$를 포함하는 하위 집합을 찾아내어 조건부 패턴 베이스를 만드는 SQL문이다. 조건부 패턴 베이스를 만드는 쿼리는 두 단계로 나누어 구성된다. 첫 번째 쿼리는 특정 경로 $[a_1, \cdots a_{i-1}]$가 포함된 노드 a_i들을 찾아내기 위한 것이다. 아이템 a_i를 기준으로 상위 경로를 찾아내는 것이기 때문에, 이 아이템 a_i를 기준아이템이라 한하고 쿼리를 통해 발견된 노드들을 기준 노드라 한다. Fig. 4-(a) 트리에서 예를 들어 보면 경로가 [T, J]에서 상위 경로 [T]가 포함된 <J> 기준 노드들은 인덱스 번호 {2, 5} 두 개가 존재한다. 두 번째 쿼리는 찾아낸 기준 노드들의 하위 집합을 구해 조건부 패턴 베이스를 만드는 쿼리이다. 찾아낸 기준 노드들의 *l-index*부터 *c-index - 1* 사이의 노드들을 아이템 별로 정렬한 후 빈도수를 계산하여 최소지지도를 만족하는 아이템만을 결과로 보여준다.

Query 1. SQL Query to Generate [a1, …, ai]'s Conditional Pattern Base

```
Query 1  SQL query to finding conditional pattern base
Input: IFP-tree table IFPTable
Output: a path [a1,…,ai]'s conditional pattern base
 1.  with tempTable as /* select ai's rows(criterion nodes) which occurred with a1,…,ai-1 items from IFPTable*/
 2.  (
 3.      select IFP1.item, IFP1.count, IFP1.c-index, IFP1.l-index
 4.      from IFPTable IFP1, IFPTable IFP2 /*   IFP1 is for candidate criterion nodes, IFP2 is for upper path nodes */
 5.      where IFP1.item = ai /* candidate nodes */
 6.            and (IFP1.c-index BETWEEN IFP2.l-index and (IFP2.c-index -1)) /* parent nodes which include the candidate
               criterion nodes */
 7.          and IFP2.item IN [a1,…,ai-1] /*upper path nodes*/
 8.      group by IFP1.c-index, IFP1.count, IFP1.item, IFP1.l-index
 9.      having count(IFP1.c-index) = (i - 1) /* the number of upper path items */
10.  )
11.  select   sum(IFP.count), IFP.item /* [a1, …, ai]'s conditional pattern base */
12.  from   IFPTable IFP, tempTable TEMP
13.  where   IFP.c-index BETWEEN TEMP.l-index and (TEMP.c-index -1)
14.  group   by IFP.item
15.  having   sum(IFP.count) >= MIN_SUPPORT;
```

Pseudo Code 6. PPFP: The Algorithm for Finding Frequent Patterns from IFP-table

```
Algorithm PPFP
Input: FP-tree table constructed based on Algorithm 1 using FP-tree, minimum support   threshold ξ,
Output: frequent pattern table
Method: Call stack-growth(), which is implemented as follows.

Procedure PPFP()
{
 (1) make conditional pattern stack,
 (2) insert {frequent item set} into conditional pattern stack
 (3) α = POP one pattern from conditional pattern stack
 (4) IF α ≠ Ø
 (5) THEN call pushAndPop(α)
}

Procedure   pushAndPop(α)
{
  (1) Insert α into frequent pattern table
  (2) β = collect α's conditional pattern base /* by Query 1. */
  (3) IF β ≠ Ø
  (4) THEN PUSH β into conditional pattern stack
  (5) α = POP one item   from conditional pattern stack
  (6) IF α ≠ Ø
  (7) THEN call pushAndPop (α)
}
```

4. 실 험

본 논문에서 제시하는 PPFP는 기존의 FP-growth방식에 비해 메모리효율성을 높여 확장성을 보장하고, 또한 기존에 보조기억장치를 활용하는 SQL기반의 FP-growth방식에 비해 시간효율성을 높여주기 위한 빈발 패턴 마이닝 알고리즘이다. 이를 증명하기 위해 다음과 같은 실험을 구상하였다.

실험은 크게 두 가지로 진행하였다. 우선 확장성 측면에서 PPFP의 성능 비교를 위해 PPFP 알고리즘과 FP-growth를 사용하여 마이닝할 때의 메모리 소비량에 대한 비교 실험을 진행하였다. 그리고 시간효율성 측면의 성능 비교를 위해 보조기억장치를 활용하는 PPFP와 SQL기반의 FP-growth 방법으로 마이닝했을 때 처리 시간에 대해 비교하고, 메모리기반의 FP-growth와의 처리 시간에 대한 비교 실험 또한 함께 진행하였다.

실험을 위해 사용된 실험 환경은 다음과 같다. 모든 실험은 i5-3210M 2.5GHz CPU와 DDR3 2G 메모리, HDD가 장착되고 윈도우7 32bit professional K 운영체제를 사용하는 컴퓨터에서 진행되었다. 데이터베이스는 오라클 11g Express Edition으로, 데이터베이스는 별도의 튜닝 없이 일반 SQL과 Table index만을 사용하여 실험하였다. 또한 데이터베이스의 경우 다중 접속을 이용한 다중 처리가 가능하지만, 실험을 위하여 한 번에 하나의 트랜잭션만 처리하도록 제한하였다.

실험에 사용된 데이터 집합(Data set)은 FP-Growth의 논문실험에서 사용된 IBM Almaden Quest research group의 generator를 이용하여 생성된 것이다. 이는 일반적으로 데이

터 마이닝 논문의 실험을 위해 가장 널리 사용되는 데이터 집합으로 TxxIyyDzzzK 속성을 설정하며 실험에 적합한 데이터 집합을 생성할 수 있다. TxxIyyDzzzK 속성에서 xx는 각 트랜잭션에서 발생되는 평균 아이템 수를, yy는 각 아이템들의 평균 지지도 값을, 마지막으로 zzz는 총 데이터 집합의 트랜잭션 수를 의미하며 1K는 1000개의 트랜잭션을 말한다. 실험에 사용된 데이터 집합은 Table 2와 같다.

첫 번째 실험은 PPFP의 확장성에 대한 성능 검증을 위하여 FP-growth와 PPFP의 메모리 사용량 비교 분석 실험이다. 실험 데이터는 T40I10D100K 데이터 집합을 사용하여 최소지지도를 0.8%를 기준으로 0.1%씩 증가시켜가며 마이닝하는데 소비되는 메모리량(Mbyte)을 측정하였다. Fig. 6의 그래프를 보면 PPFP에 비해 FP-growth를 이용하여 마이닝하는데 훨씬 더 많은 메모리 공간이 필요하다는 것을 확인할 수 있다. FP-growth의 경우 최소지지도가 작아짐에 따라 메모리 소비량이 급격히 증가하며, T40I10D100K 데이터 집합에서 최소지지도 0.5%의 마이닝 시 메모리가 600Mbyte 이상 소비되었다. 본 실험에 사용된 컴퓨터에서 총 메모리 중 가용 가능 여유 메모리가 500Mbyte 이하로 떨어진 경우 프로그램이 멈추는 경우도 발생하였다.

두 번째 실험은 PPFP 알고리즘의 시간효율성과 관련된 실험이다. 실험을 통해 PPFP 알고리즘과 SQL기반의 FP-growth방법의 마이닝 처리 시간을 비교하여 보조기억장치를 활용하여 빈발 패턴을 마이닝하는 방법에서 PPFP알고리즘의 시간효율성을 비교 분석하였다. 실험에는 T25I10D10K 데이터 집합을 사용하여 최소지지도를 1%부터 10%까지 1%씩 증가시켜가며 실험하였다. 실험 결과는 Fig. 7과 같다. Fig. 7에서와 같이 PPFP방식은 SQL기반의 방식에 비해 매우 좋은 시간효율성을 보이고 있다. 최소지지도가 1%인 경우 PPFP에 비해 SQL기반의 방식이 100배 이상의 시간이 소비되고, 10%의 경우 200배 이상의 차이를 보였다.

다음은 PPFP 알고리즘과 메모리 기반의 마이닝 방법인 FP-growth의 시간효율성 비교 실험이다. 실험을 위해 T10I4D100K, T25I10D10K, T25I20D100K의 세 데이터 집합을 사용하였고, 최소지지도를 0.5%부터 1%까지 0.1%씩 증가시켜가며 PPFP와 FP-growth 알고리즘을 이용하여 마이닝하는데 소비되는 시간을 기록하였다. Fig. 8과 Fig. 9는 각각 PPFP와 FP-growth 알고리즘을 이용하여 마이닝하는데 소비된 처리 시간에 대한 그래프이다.

실험 결과에서 보이는 바와 같이, PPFP 알고리즘이 기존의 보조기억장치를 활용한 방법들에 비해 빠른 성능을 보이지만 메모리 기반의 FP-Growth에 비해서는 많은 시간이 소비되는 것을 볼 수 있다. 하지만 PPFP 알고리즘의 경우 최소지지도가 증가함에 따라 처리시간이 급격히 줄어드는 것을 볼 수 있다. 반면 FP-Growth의 경우 최소지지도의 증가가 처리 시간에 큰 영향을 미치지 않는다. Fig. 8의 그래프에서 보면, T10I4D100K의 처리시간이 T25I10D10K에 대한 처리시간보다 오히려 오래 걸리는 것을 볼 수 있다. 이러한 특징은 발생되는 빈발 패턴의 수 보다는 FP-tree의 크기가 FP-growth의 처리 시간에 영향을 미치기 때문이다.

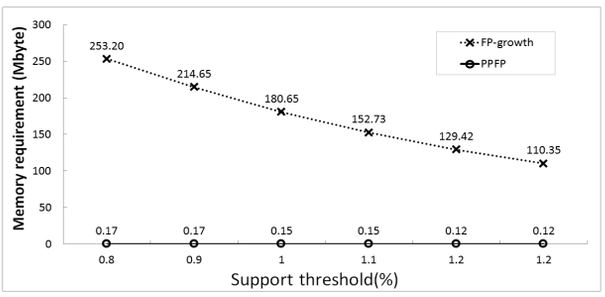


Fig. 6. Memory Requirement of PPFP on T40I10D100K Dataset

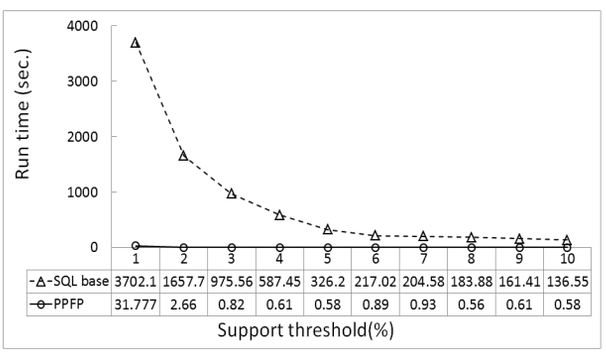


Fig. 7. Run Time of PPFP on Various Support Thresholds from T25I10D10K

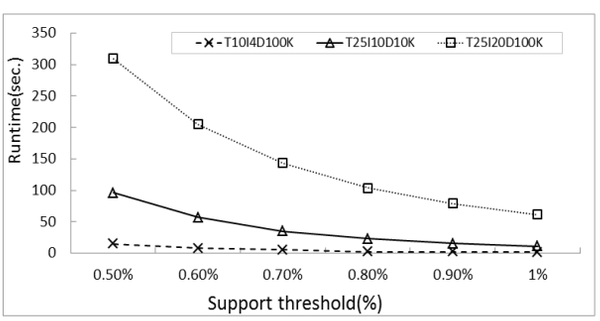


Fig. 8. Runtime of PPFP-growth on Various Sizes of Datasets

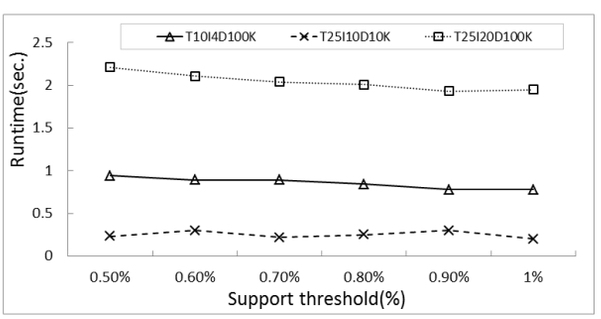


Fig. 9. Runtime of FP-growth on Various Size of Datasets

Fig. 10과 Fig. 11은 Fig. 8과 Fig. 9에 사용된 데이터 집합과 최소지지도를 이용해 마이닝 과정에서 생성되는 FP-tree의 노드의 수와 마이닝의 결과로 발생된 빈도 패턴의 개수에 대한 그래프이다.

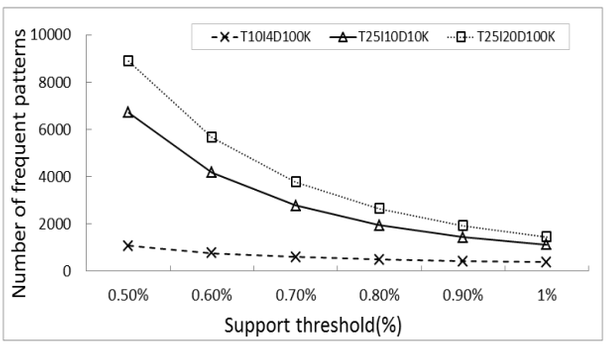


Fig. 10. The Number of Frequent Patterns at Various Sizes of Datasets

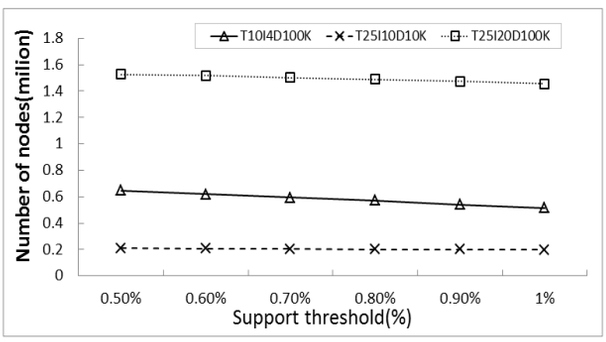


Fig. 11. The Number of Nodes in FP-tree at Various Sizes of Datasets

PPFP는 FP-tree에 인덱스를 추가한 *IFP-tree*를 사용하여 마이닝하기 때문에 FP-tree와 *IFP-tree*의 노드의 개수는 같다. 또한 마이닝 후 발생된 빈도 패턴의 개수도 동일하다. 따라서 PPFP와 FP-growth에 사용되는 FP-tree와 *IFP-tree*의 노드 개수는 같으며, 마이닝 결과로 발생되는 빈발 패턴의 수 역시 동일하다. 이 실험 결과 그래프를 자세히 살펴보면 앞선 PPFP와 FP-growth 알고리즘의 처리 시간 그래프와 유사한 점을 발견할 수 있다. Fig. 8의 PPFP의 처리 시간 그래프와 Fig. 10의 발생된 빈도 패턴의 수 그래프는 비슷한 곡선을 보이고 있다. 또한 Fig. 9의 FP-growth의 처리 시간 그래프는 Fig. 11의 FP-tree의 노드의 수 그래프와 유사성을 보인다. Fig. 10의 빈발 패턴의 수 그래프는 Fig. 8의 PPFP 처리시간 그래프와 같이 최소지지도가 증가할수록 급격히 줄어드는 것을 볼 수 있다. 반면 Fig. 11의 FP-tree 노드의 수 그래프는 Fig. 9의 FP-growth 처리 시간 그래프와 같이 최소지지도의 증가에 크게 영향을 받지 않는 것을 볼 수 있다. 이를 통해 PPFP 알고리즘의 경우 규모가 큰 데이터 집합에서 아이템 발생 빈도가 적으며 실시간을 요하지 않고 트리의 크기가 크고 발생되는 패턴의 빈도가 높은 데이터 처리에 적합함을 유추할 수 있다. 마찬가지로 FP-growth의 경우 빠른 처리시간을 요하고, 전체적인 아이템 빈도가 높으며, 트리의 크기가 작고, 발생 패턴의 수가 많은 데이터의 처리에 유리하다.

5. 결론 및 향후 연구과제

PPFP 알고리즘은 메모리와 보조기억장치를 적절히 활용하여 기존에 메모리 의존적인 알고리즘들에 비해 확장성을 높이고, 또한 보조기억장치만을 활용한 방법들에 비해 시간효율성을 높인 새로운 빈도 패턴 마이닝 방법이다. 실험을 통해 기존의 메모리 의존성이 높은 FP-growth에 비해 PPFP 알고리즘이 확장성이 높다는 것과, 기존의 보조기억장치를 활용한 SQL 기반의 FP-growth 방식에 비해 시간효율성이 좋다는 것 역시 확인할 수 있었다.

하지만, 기존의 보조기억장치를 활용한 방법들에 비해 시간적인 비용을 줄이는데 일조하였으나, 여전히 FP-growth에 비해 시간효율성이 떨어지는 것을 볼 수 있었다.

이에 대한 성능 개선을 위해 현재 데이터베이스의 이점인 다중트랜잭션 환경을 이용하고, 멀티코어 환경에서 HDD 대신 SSD를 사용하여 성능을 향상시키는 실험을 진행하고 있다. 이러한 실험을 통해 FP-growth에 비해 떨어지는 시간 효율성을 개선시킬 수 있을 것이라 여겨진다.

References

[1] R. Agrawal, T. Imieliski, and A. Swami, “Mining association rules between sets of items in large databases,” in *Proc. ACM SIGMOD Int. Conf. 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] A. Amir, R. Feldman, and R. Kashi, “A new and versatile method for association generation,” *Inf. Syst.*, Vol.22, No.6/7, pp.333-347, Sep.-Nov., 1997.

[4] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp.1-12, 2000.

[5] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Trans. Knowl. Data Eng.*, Vol.12, No.3, pp.372-390, May, 2000.

[6] M. El-Hajj and O. R. Zaiane, “COFI approach for mining frequent item-sets revisited,” in *Proc. ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery*, New York, pp.70-75, 2004.

[7] W. Cheung and O. R. Zaiane, “Incremental mining of frequent patterns without candidate generation or support constraint,” in *Proc. IEEE Int. Conf. Database Eng. Appl., Los Alamitos*, CA, pp.111-116, 2003.

[8] C. K.-S. Leung, Q. I. Khan, and T. Hoque, “Cantree: A tree structure for efficient incremental mining of frequent patterns,” in *Proc. IEEE Int. Conf. Data Mining, Los Alamitos*, CA, pp.274-308, 2005.

[9] 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.

[10] J. H. Lee, "IRFP-tree: Intersection Rule Based FP-tree," *KIPS Transaction on Software and Data Engineering*, Vol. 5, Issue 3, pp.155-164, 2016.
[11] J.-L. Koh and S.-F. Shieh, "An efficient approach for maintaining association rules based on adjusting FP-tree structures," in *Proc. DASFAA*, Springer-Verlag, Berlin Heidelberg New York, pp.417-424, 2004.
[12] G. Liu, H. Lu, J. X. Yu, W. Wang, and X. Xiao, "AFOPT: An efficient implementation of pattern growth approach," in *Proc. FIMI*, 2003.
[13] M. Adan and R. Alhajj, "DRFP-tree: Disc-resident frequent pattern tree," *Appl. Intell.*, Vol.30, No.2, pp.207-216, 2009.
[14] M. Adan 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*, Vol.41, Issue 1, pp.154-172, 2011.
[15] X. Shang, K.-U. Sattler, and I. Geist, "SQL Based Frequent Pattern Mining with FP-growth," in *INAP/WLP*, pp.32-46, 2005.
[16] B. Goethals, "Memory issues in frequent itemset mining," in *Proc. ACM SAC*, pp.530-534, 2004.
[17] R. Vaarandi, "A breadth-first algorithm for mining frequent patterns from event logs," in *Proc. IEEE INTELLCOMM*, pp.293-308, 2004.
[18] 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 동국대학교산업기술 연구원
2014년~2015년 동국대학교 IT 융합교육센터
2015년~현 재 동국대학교 전산원 강의전담교수
관심분야 : SW 품질평가, 빅데이터, 데이터마이닝

민 연 아

e-mail : ltjgus0524@empas.com
1996년 동국대학교 컴퓨터공학과(학사)
2002년 동국대학교 컴퓨터공학과(석사)
2013년 동국대학교 컴퓨터공학과(박사)
2010년~2013년 M Communication 대표
2014년~2014년 국제 프로솔 컨폼 연구원
2014년~2016년 글로벌데이터연구소 연구소장
2010년~2016년 한양여자대학교 정보경영과 겸임교수
2016년~현 재 가천대학교 SW중심대학사업단 초빙교수
관심분야 : 하이브리드 디스크 시스템, 메타데이터 관리 시스템, 상호작용기반의 협동학습 시스템, 데이터마이닝