FP-tree와 DHP 연관 규칙 탐사 알고리즘의 실험적 성능 비교

(Performance Evaluation of the FP-tree and the DHP Algorithms for Association Rule Mining)

이 형 봉 † 김 진 호 ††

(Hyung Bong Lee) (Jinho Kim)

요 약 *FP-tree*(*Frequent Pattern Tree*) 연관 규칙 탐사 알고리즘은 DB 스캔에 대한 부담을 획기적으로 절감시킴으로써 전체적인 성능을 향상시키고자 제안되었고, 따라서 다른 기법에 기반하는 알고리즘보다 성능이 매우 우수한 것으로 알려져 있다. 그러나, *FP-tree* 알고리즘은 기본적으로 DB에 저장된 거래 내용 중 빈발 항목을 포함하는 모든 거래를 트리에 저장해야 하기 때문에 그만큼 많은 메모리를 필요로 한다. 이 논문에서는 범용 운영체제인 유닉스 시스템 환경에서 *FP-tree* 알고리즘을 구현하여 소요 메모리와 실행시간 등 두 가지 성능 관점에서 해시 트리 및 직접 해시 테이블을 사용하는 *DHP*(*Direct Hashing and Pruning*) 알고리즘과 비교한다. 그 결과로서 알려진 바와는 크게 다르게 시스템 메모리가 충분한 상황에서도 대형 편의점 수준의 규모에 적용 가능한 거래 건수 100K, 전체 항목 개수 1K~7K, 평균 거래 길이 5~10, 평균 빈발 항목 집합 크기 2~12인 데이타에 대해서 *FP-tree* 알고리즘이 *DHP* 알고리즘보다 열등한 경우가 존재함을 보인다.

키워드 : *FP-tree*, *DHP*, 해시 트리, 직접 해시 테이블

Abstract The *FP-tree*(*Frequent Pattern Tree*) mining association rules algorithm was proposed to improve mining performance by reducing DB scan overhead dramatically, and it is recognized that the performance of it is better than that of any other algorithms based on different approaches. But the *FP-tree* algorithm needs a few more memory because it has to store all transactions including frequent itemsets of the DB. This paper implements a *FP-tree* algorithm on a general purpose UNIX system and compares it with the *DHP*(*Direct Hashing and Pruning*) algorithm which uses hash tree and direct hash table from the point of memory usage and execution time. The results show surprisingly that the *FP-tree* algorithm is poor than the DHP algorithm in some cases even if the system memory is sufficient for the *FP-tree*. The characteristics of the test data are as follows. The size of DB is 100K, the number of total items is 1K~7K, avenrage length of transactions is 5~10, avergage size of maximal frequent itemsets is 2~12(these are typical attributes of data for large-scale convenience stores).

Key words : *FP-tree*, *DHP*, hash tree, direct hash table

† 종신회원 : 강릉대학교 컴퓨터공학과 교수
hblee@kangnung.ac.kr
†† 종신회원 : 강원대학교 컴퓨터학부 교수
jhkim@kangwon.ac.kr
논문접수 : 2007년 8월 21일
심사완료 : 2008년 1월 29일

정보과학회논문지: 데이타베이스 제35권 제3호(2008.6)

1. 서 론

일반적으로 알고리즘의 성능은 그 알고리즘이 실행되는 환경에 따라 달라질 수 있다. 이 때, 환경이란 전형적으로 데이타의 크기나 특성 등을 말하는데, 해당 알고리즘은 그러한 환경에 따라 최선(최고)의 성능을 발휘하기도 하고 최악의 성능을 내기도 한다. 최악의 경우는 성능이 최소한 어느 정도 이상이라는 관점에서, 최선의 경우는 성능이 좋다고 해도 어느 정도 이하라는 관점에서 활용될 수 있다. 이와 같이 최선과 최악의 경우는 어떤 특수한 상황을 전제해야 하기 때문에 평상시의 성능

관점에서는 미흡하다. 이를 해결하기 위해 통계적 기법을 활용하는 평균적 성능을 계산하여 사용한다.

연관 규칙 탐사 알고리즘의 경우 성능 평가를 위해 내부 연산과정을 고려한 세밀한 복잡도를 분석한 사례는 아직 없지만, 대체로 DB에서 거래를 읽고, 거래를 정제하여 검색용 메모리에 저장한 후, 빈발 항목 집합을 탐색하는 등의 절차를 거시적 관점에서 살펴본다면 대다수 알고리즘의 복잡도는 유사할 것이다. 이런 사실은 지금까지 이루어진 연관 규칙 탐사 알고리즘에 관한 거의 모든 성능 분석이, 주어진 동일한 시험용 데이타에 대한 개선정도를 상대적으로 비교하는 방법으로 이루어지고 있다는 점을 통해서도 잘 알 수 있다.

이 논문에서도, 최근 성능이 가장 우수한 연관 규칙 탐사 알고리즘의 하나로 널리 알려진 *FP-tree* 알고리즘[1]이 과연 절대적으로 우수한가를 관찰하기 위하여, 복잡도 분석은 하지 않고 소요 메모리 및 실행 시간 등 두 가지 측면에서, 대비되는 다른 대표적인 알고리즘인 *DHP* 알고리즘[2]과의 성능을 비교·분석한다. 이 연구가 이루어지게 된 동기는 [3]의 논문이 제출되는 과정에서, "성능이 가장 우수한 *FP-tree* 알고리즘이 제안된 상황에서 그보다 성능이 저조한 *DHP* 알고리즘에 관한 연구가 의미를 가지겠는가"라는 의문이 제기된 데에서 비롯되었다. 즉, 어떤 알고리즘이 경우에 관계없이 어떤 상황에서도 상대적 성능 우월성을 유지하기가 쉽지 않다는 보편적인 사실에 입각하여 *FP-tree* 알고리즘의 성능을 재조명해 보고자 하는 것이다.

이를 위하여 2장에서 연관 규칙 탐사 알고리즘의 진화 과정을, 3장에서는 *FP-tree* 알고리즘을 간략하게 고찰하고, 4장에서 다양한 유형의 시험용 데이타를 적용한 두 알고리즘의 성능 평가 결과를 비교·분석하고, 마지막 5장에서 결론으로 이 논문을 맺는다.

2. 연관 규칙 탐사 알고리즘의 진화

가장 알기 쉬운 연관 규칙 탐사 알고리즘은 전체 항목 집합에 대한 모든 부분 집합에 대응되는 계수 공간을 마련한 후, DB에 저장된 각각의 거래를 읽어 거기에 포함된 항목들로 구성 가능한 부분 항목 집합에 대한 계수를 실시하는 것이다. 이를 흔히 원시 알고리즘이라 하는데, 이 알고리즘은 다음과 같이 두 가지 큰 문제점을 안고 있다.

첫 번째 문제점은 많은 양의 계수 공간을 필요로 한다는 점이다. 예를 들어 전체 항목 집합의 크기 $|I|$가 5,000개라면 약 $2^{5,000}$ 개의 계수 공간이 필요한데 이를 지원할 수 있는 시스템은 흔하지 않다. 두 번째 문제점은 계수 공간이 마련되었다 하더라도 대응되는 계수 지점을 찾기 위해서는 많은 시간을 필요로 한다는 점이다.

위와 같은 근본적인 문제점을 해결하고, 보다 나은 성능을 얻기 위해 연관 규칙 탐사 알고리즘은 꾸준히 진화해 왔다. *AIS(finding All frequent Item-Sets)* 알고리즘[4]은 탐색을 단계별로 진행하되, 현재까지의 빈도 통계를 적용하여 전 단계의 빈발 항목 집합으로부터 후보 빈발 항목 집합을 도출하여 다음 단계를 결정함으로써 한꺼번에 필요로 하는 계수 공간을 줄였고, *Apriori* 알고리즘[5]은 빈도 통계 대신 각 단계별 빈발 항목 집합의 길이를 1, 2, 3, …과 같이 정확히 1씩 늘려가면서 후보 빈발 항목 집합을 유도하여 규칙적인 단계로 진행하여 실행 시간을 단축시켰다. *DHP(Direct Hashing and Pruning)* 알고리즘[2]은 길이 2인 단계(단계 2)에서 작성된 직접 해시 테이블(direct hash table)의 사전 전지 정보를 활용하여 길이 3의 후보 빈발 항목 집합의 수를 획기적으로 감소시킴으로써 계수 공간 및 검색 시간을 단축하였을 뿐만 아니라 거래 중에서 빈발이 아닌 항목들을 제거해 나감으로써 다음 단계에서 고려해야 하는 DB의 크기를 줄였다.

DB 스캔 횟수 측면에서 원시 알고리즘은 단 1회로 충분하나, *AIS*, *Apriori*, *DHP* 알고리즘은 최대 빈발 항목 집합의 크기만큼의 DB 스캔 횟수가 필요하기 때문에 소요 기간의 많은 부문이 DB 입·출력에 소비된다.

FP-tree 알고리즘[1]은 지금까지의 알고리즘들이 필요로 했던 여러 번의 DB 스캔 횟수를 단 2회로 단축시킴으로써 DB 입·출력 부담을 획기적으로 줄였다. 이 알고리즘은 단계 1에서 빈발 항목이 아닌 항목들을 제거하고, 단계 2에서 빈발 항목들로만 구성된 거래들을 모두 *FP-tree* 구조에 저장한 다음, 단계 3에서 *FP-tree*로부터 빈발 항목 집합을 도출해 낸다.

즉, *FP-tree* 알고리즘은 DB의 거래 내용을 필터링한 모든 거래 내용을 메모리에 구축한 후 그 곳으로부터 빈발 항목 집합을 추출한다. 따라서, *FP-tree* 알고리즘은 다른 알고리즘보다 많은 양의 메모리를 필요로 하지만, 전체적인 성능은 다른 방법론에 기반한 어떤 알고리즘보다 우수한 것으로 알려져 왔다[1,6-9]. 그러나, 이 논문의 실험결과 메모리가 충분한 상황에서도 *FP-tree* 알고리즘이 *DHP* 알고리즘보다 열등한 경우가 다수 존재함이 밝혀졌다. 이와 같이 *FP-tree* 알고리즘의 우수성이 무조건적으로 당연시 되었던 이유는 *FP-tree* 알고리즘을 최초로 제안한 [1]에서 *DHP*알고리즘을 *Apriori* 알고리즘의 한 부류로 치부하고 세밀한 비교대상에서 제외했기 때문인 것으로 판단된다. 알고리즘의 특성 측면에서 *DHP*가 *Apriori* 유형에 속한다는 점은 사실이지만 성능 측면에서는 분명히 차원이 다르다[2, 10-13].

이 논문에서는 *FP-tree*와 *DHP* 알고리즘의 근원적

인 성능을 비교·분석하기 위해 *FP-tree*의 변형[6-9 등]과 *DHP*의 변형[10-13 등]들은 고려하지 않고 각각의 기본형인 [1]과 [2]의 알고리즘을 사용하는데, 그 이유는 *FP-tree*의 변형들은 메모리 부족 등 주로 특수한 경우를 효과적으로 처리하기 위한 제안들이고, *DHP*의 경우는 이 논문의 논점상 성능이 최하인 기본형을 사용해도 무방하기 때문이다. [1]의 알고리즘은 [14]의 구현 내용을 사용하되 여기서는 주요 자료구조만 간략하게 짚어본다. 그리고 [2]의 알고리즘은 [13]에서 구현된 내용을 발췌하여 그대로 사용한다.

3. *FP-tree* 알고리즘 고찰

[1]에 제시된 *FP-tree* 알고리즘은 *FP-tree*를 생성하는 과정과 *FP-tree*로부터 빈발 항목 집합을 추출하는 과정 등 크게 두 단계로 이루어져 있고, 소요 메모리에 가장 큰 영향을 미치는 요소는 *FP-tree* 및 패턴 연산을 위한 자료구조이다.

3.1 *FP-tree*의 생성

*FP-tree*는 DB를 두 번 스캔해서 완성되는데, 첫 번째 스캔에서 빈발 항목을 선별하고, 두 번째 스캔에 서 각 거래로부터 빈발 항목이 아닌 것들은 제외하고 나머지 빈발 항목들로만 구성된 정제된 거래만을 *FP-tree*에 삽입한다. 이는 빈발 항목 집합에 기여할 수 없는 항목들을 미리 전지하고 거래에 포함된 패턴을 중복시킴으로써 *FP-tree* 자체의 크기를 줄임과 동시에 검색 시간을 단축하기 위함이다. *FP-tree*의 개념을 그림 1에 보였다. 그림 1에서 트리의 각 노드는 root로부터 출발하는 트리 고유의 링크(실선)와 head table로부터 출발하는 해당 항목 노드 링크(점선) 등 두 개의 링크를 유지하고, 각 노드는 item-name, count 등 두 가지 요소로 구성 된다.

그림 1은 지지도 3을 기준으로 그림 2의 거래 DB로부터 생성된 *FP-tree*의 모습이다.

3.2 *FP-tree*의 탐색(빈발 항목 집합 추출)

그림 1로부터 빈발 항목 집합을 탐색하는 방법은 사뭇 직관적이라고 말할 수 있다. 예를 들어 항목 *p*가 포함된 빈발 항목 집합을 찾는다면 우선 두 빈발 항목 집합 {*fcamp*:2}, {*cbp*:1}은 쉽게 찾아 낼 수 있다(여기서 ":2", ":1"은 지지도를 표시). 나머지 빈발 항목 집합들은 항목 *p*가 포함되어 있다는 전제 하에 두 항목 집합 {*fcam*:2}과 {*cb*:1}를 독립된 거래로 보고 두 거래에서 지지도를 만족하는 또 다른 부분 항목 집합이 존재하는지를 탐색하면 되는데, 이는 {*fcam*:2}과 {*cb*:1}를 대상으로 하부 *FP-tree*를 구성한 다음 위 과정을 재귀적으로 반복하여 얻어진 빈발 항목 집합들에 항목 *p*를 추가하면 얻을 수 있다.

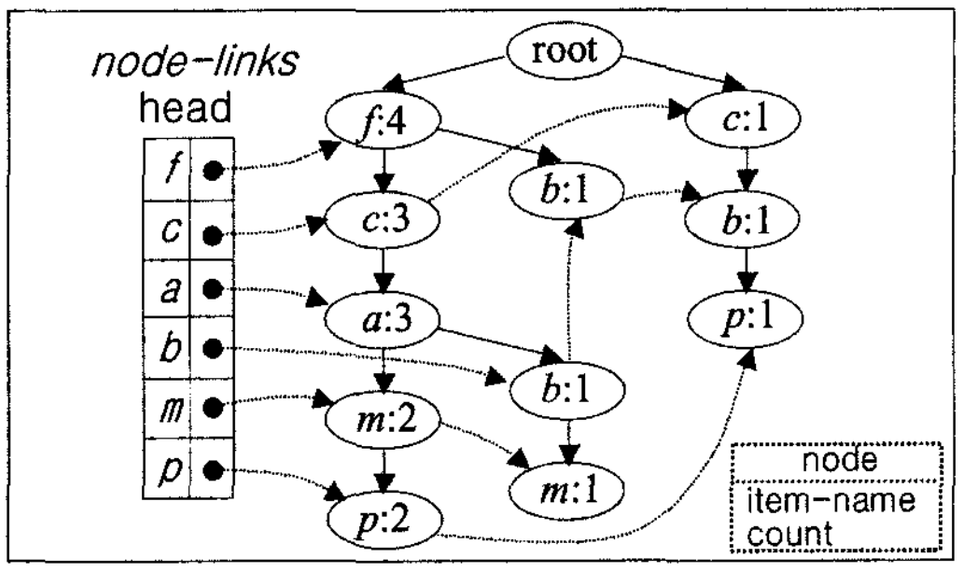


그림 1 *FP-tree* 자료구조

TID	Items in transaction	Ordered frequent items
100	*f, a, c, d, g, i, m, p*	*f, c, a, m, p*
200	*a, b, c, f, l, m, o*	*f, c, a, b, m*
300	*b, f, h, j, o*	*f, b*
400	*b, c, k, s, p*	*c, b, p*
500	*a, f, c, e, l, p, m, n*	*f, c, a, m, p*

그림 2 그림 1에 사용된 거래 DB의 예

[1]에서 {*fcam*:2}와 {*cb*:1}를 "*p* 조건부 패턴 기반(*p*'s conditional pattern-base)"이라 명명했고, 이들을 대상으로 생성된 트리를 "*p* 조건부 *FP-tree*(*p*'s conditional *FP-tree*)"라고 명명했으며, *p*를 "선행 경로(prefix path)"라고 정의했다. 선행 경로는 재귀 호출이 깊어질수록 그 길이도 함께 증가한다. 즉, *FP-tree* 알고리즘은 최초에 생성된 통합 트리 외에, 탐색 과정에서 선행 경로를 위한 조건부 *FP-tree*의 재귀적 생성과 소멸을 반복해야 한다.

3.3 *FP-tree* 알고리즘의 구현

여기서는 [14]에서 구현된 내용 중, *FP-tree* 알고리즘에 가장 큰 영향을 미치는 *FP-tree*와 패턴 및 패턴 집합의 자료구조만을 간략하게 제시한다.

• *FP-tree* 자료구조의 구현

그림 1의 *FP-tree*를 C 언어로 구현한 자료구조 및 관련 코드를 그림 3에 보였다. 여기서 sibling link는 부모가 동일한 모든 노드들을 연결하고, child link는 자식 노드들 중 첫 노드를 연결한다. 또한, 노드의 parent link는 자신의 부모 노드를 지시하므로, 부모가 동일한 모든 형제 노드들의 parent link 값은 모두 동일하다. 그리고, 그림 1에서의 노드 링크를 혼돈을 피하기 위해 여기서는 item head link라는 용어를 사용하였다.

• 패턴 및 패턴 집합 자료구조의 구현

이 논문 및 [1]에서의 패턴은 [2]에서 정의한 빈발 항목 집합(large itemset)에 해당되고, 따라서 패턴 집합은 빈발 항목 집합의 집합을 말한다. *FP-tree* 알고리즘을 구현하기 위해 도입한 패턴과 패턴 집합의 자료구조를

```
typedef struct  node {
    u_short      count, item_name;
    u_short      flag;    /* tag root etc,.... */
    struct node  *clink;  /* child link */
    struct node  *plink;  /* parent link */
    struct node  *slink;  /* sibling link */
    struct node  *ilink;  /* item head link */
} Node;
```

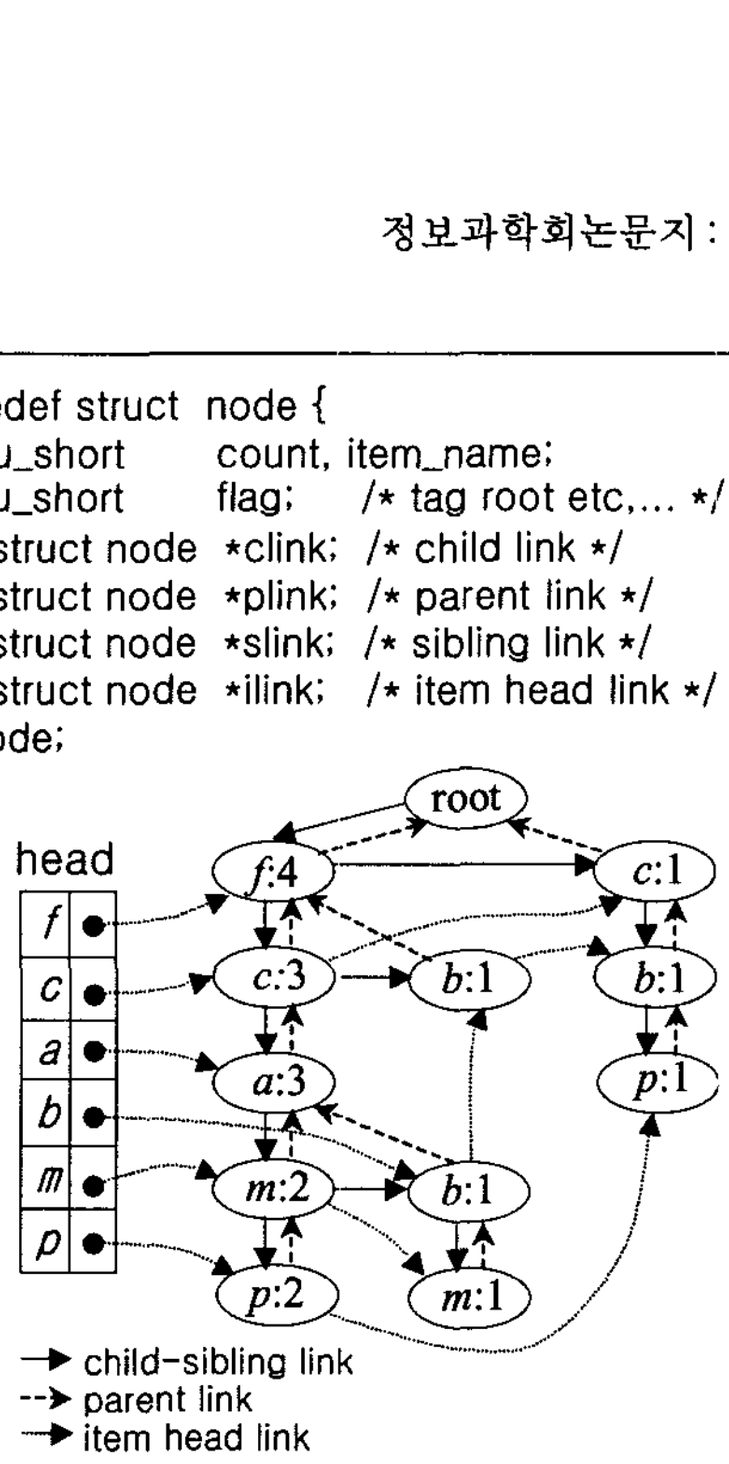


그림 3 *FP-tree*의 구현 자료구조

```
typedef struct  pattern {
    struct pattern  *next;  /* pattern link list */
    u_short      item_len;  /* pattern length */
    u_short      item_name[1];   /* item list */
} Pattern;

typedef struct  patternset {
    short        nset;       /* # of  patterns */
    struct pattern  *first;  /* front pattern list */
    struct pattern  *last;   /* rear pattern list */
} PatternSet;
```

pattern set array

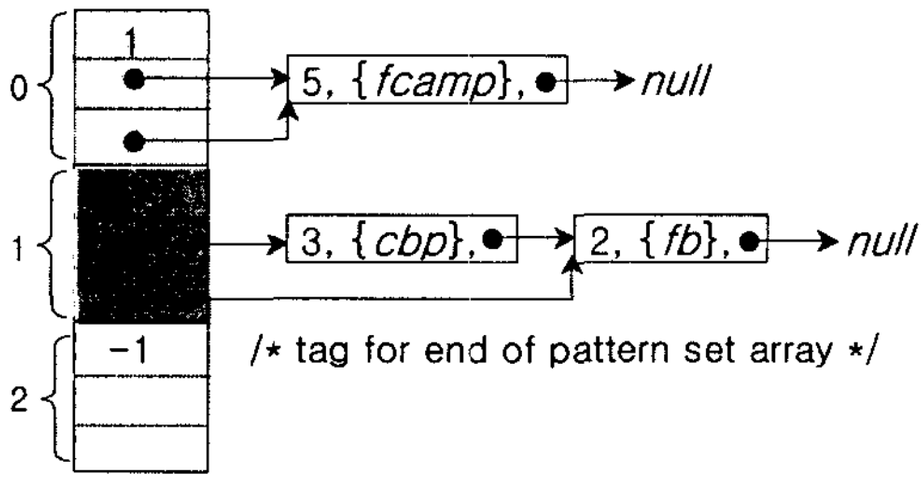


그림 4 패턴과 패턴 집합의 구현 자료구조

그림 4에 보였는데, 알고리즘의 구현 결과 패턴 집합이 단독으로 참조되는 경우 보다는 여러 개의 패턴 집합들이 그룹으로 참조되는 경우가 많고, 패턴과 패턴 집합의 연산은 패턴 및 패턴 집합의 수를 역동적으로 변화시키기 때문에 이런 연산 특성에 유연하게 대응하기 위해 모든 패턴 집합은 패턴 집합의 배열로 할당하여 사용하도록 하였다.

그림 4에 보인 패턴 집합 배열의 예는 2 개의 패턴 집합을 포함하고 있는데, 첫 번째 패턴 집합은 길이 5인 패턴 {*fcamp*} 하나를, 두 번째 패턴 집합은 길이 3인 패턴 {*cbp*}과 길이 2인 패턴 {*fb*} 등 2 개의 패턴을 저장하고 있는 모습이다.

4. *FP-tree* 알고리즘 성능의 비교 · 분석

4.1 성능 분석 환경

구현된 *FP-tree* 알고리즘의 성능 분석이 이루어진 시스템 환경은 표 1과 같고, [15]에서 제공된 시험용 데이타 생성 프로그램을 활용하여 표 2의 데이타들을 성능 분석에 사용하였다. 이 때, 데이타는 {거래 길이, 항목1, 항목 2, … 항목 n} 형태의 레코드로 구성된 파일로 구축하였다.

표 1 성능 분석 시스템 환경

항 목	사 양
모 델	COMPAQ WS au600
C P U	Alpha Ev67 667MHz
메모리	1GB
운영체제	Digital Tru64UNIX 4.0F

표 2 시험 데이타의 유형

분류기준	분류내용
데이타 특성	T5I2, T10I2, T10I4, T15I40, T20I2, T20I4, T20I6
전체 항목 개수(|I|)	N1000, N2000, N3000, N5000, N7000, N9000
DB 크기(|D|)	D100

표기 예
- T1014 : 거래의 평균 길이가 10, 최대빈발항목집합의 평균 크기가 4
- N5000 : 전체 항목 개수가 5000
- D100 : 전체 거래 건수가 100K

4.2 *FP-tree*와 *DHF* 알고리즘의 성능 특성 비교

4.2.1 성능 비교 · 분석 방법

DHP 알고리즘의 소요 메모리 및 실행 시간은 후보 빈발 항목 집합을 사전에 전지하기 위한 단계 2에서의 직접 해시 테이블의 버켓 크기를 어느정도 할당하느냐에 따라 큰 영향을 받는다. 이 논문에서는 *FP-tree* 알고리즘과의 성능 비교를 위하여 버켓 크기 100K, 300K, 500K, 700K, 900K 각각에 대하여 표 2의 시험용 데이타에 대한 *DHP* 알고리즘의 메모리 및 실행 시간 성능을 측정한 후, 이중 그림 8, 그림 9, 그림 10의 *FP-tree* 알고리즘 성능과의 비교 대상으로 사용할 버켓의 크기를 아래의 방법으로 선택하였다.

- 그림 10의 *FP-tree* 알고리즘의 실행 시간 보다 우수하게 되는 첫 버켓 크기를 선택. 예를 들어 지지도

0.25%일 때, T10I4, N5000 데이타에 대한 *DHP* 알고리즘의 실행 시간은 버켓 크기가 100K이면 86.7sec, 300K이면 53.4sec, 500K이면 48.7sec, 700K이면 46.6sec, 900K이면 46.6sec인데, *FP-tree* 알고리즘의 실행 시간인 49.5sec보다 성능이 최초로 우수해지는 500K를 사용했을 때의 메모리 및 실행시간을 비교 대상 성능으로 선정.

- 버켓 크기와 관계 없이 *DHP* 알고리즘의 실행 시간이 그림 10의 *FP-tree* 알고리즘보다 열등할 경우에는 성능 향상이 뚜렷하게 이루어지는 마지막 버켓 크기를 선택. 예를 들어 지지도 0.25%일 때, T20I6, N5000 데이타에 대한 *DHP* 알고리즘의 실행 시간은 버켓 크기가 100K이면 8332.5sec, 300K이면 1596.1sec, 500K이면 1224.6 sec, 700K이면 1120.7sec, 900K이면 1073.8sec인데, 이들 모두가 *FP-tree* 알고리즘의 실행 시간인 312.8sec보다 열등하므로, 그 중 자체 성능이 뚜렷하게 우수한 900K를 사용했을 때의 메모리 및 실행 시간을 비교 성능 대상으로 선정.

4.2.2 DB 속성에 따른 요구 메모리 변화 특성

DB 크기는 탐사 대상이 되는 전체 거래 즉, 알고리즘에 입력되는 데이타의 크기를 말하는 것으로, 기본적으로 거래 건수와 데이타 특성 등 두 가지 요인에 따라 결정된다. 거래 건수와 DB 크기는 단순 선형 비례 관계임이 자명하고, 거래의 평균 길이와 최대 빈발 항목 집합의 크기가 클수록 DB의 크기를 증가시킬 것이다. 그림 5에 이 논문에서 사용된 데이타의 속성에 따른 DB 크기의 변화를 보였다.

그림 6, 그림 7에는 DB 크기에 따라 *FP-tree*와 *DHP* 알고리즘이 요구하는 메모리의 증가 추이를 각각 보였는데, 이 그림으로부터 *FP-tree*의 경우 데이타 속성에 따른 메모리 요구량이 DB 크기의 증가 추세보다 훨씬 많은 양을 요구하고, *DHP*는 어느 경우 에나 메모리 요구량이 입력 데이타인 DB의 크기에 따라 급격한 변화를 보이지 않는다는 사실을 볼 수 있다. 즉, DB의 크기가 거래의 길이 등에 따라 완만하게 증가할 때 *FP-tree*의 경우 이를 처리하기 위한 알고리즘의 메모리 량은 그 보다 훨씬 가파른 증가율을 보인다는 것이다. 예를 들어, 그림 6에서 T5I2와 T20I6의 DB 크기는 각각 3.2MB와 9.2MB로 약 3배로 증가하였으나, 알고리즘의 요구 메모리는 각각 19.1MB와 174.1MB로 약 9배 가까이 증가하였다. 따라서 거래의 길이나 빈발 항목 집합의 길이가 메모리가 수용할 수 있는 한계를 벗어나면 [1]에서 언급한 바와 같이 B+-tree 구조를 바탕으로 하는 disk-resident FP-tree 개념의 도입이 불가피하고 이는 결국 또 다른 DB 스캔 부담을 유발한다.

이와 같은 현상은 그림 8에 보인 바와 같이 다양한 데이타 유형에서 동일하게 나타난다. 그러나, 그림 9에

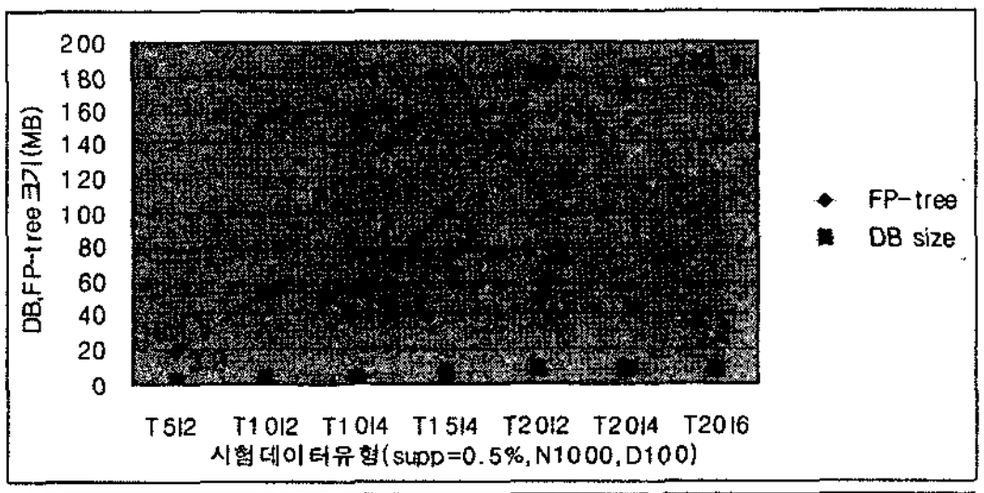
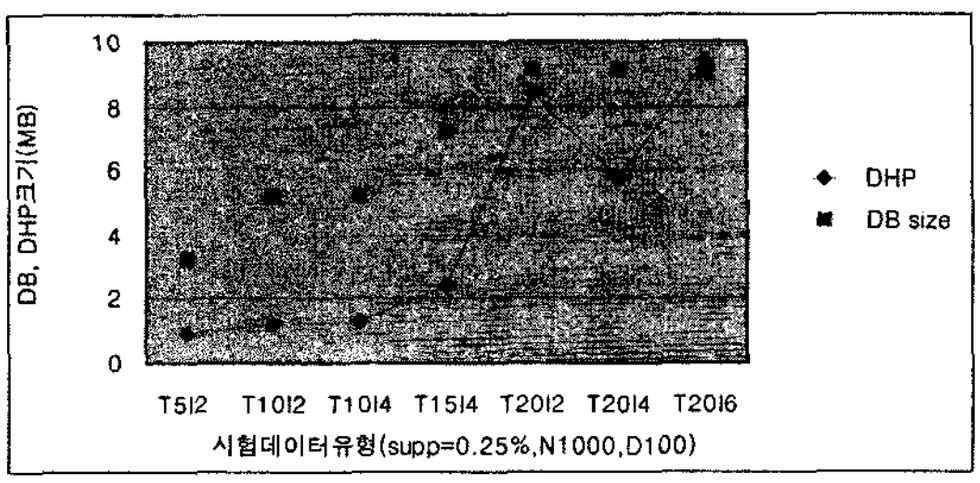


그림 6 DB 크기와 *FP-tree* 소요 메모리량의 관계



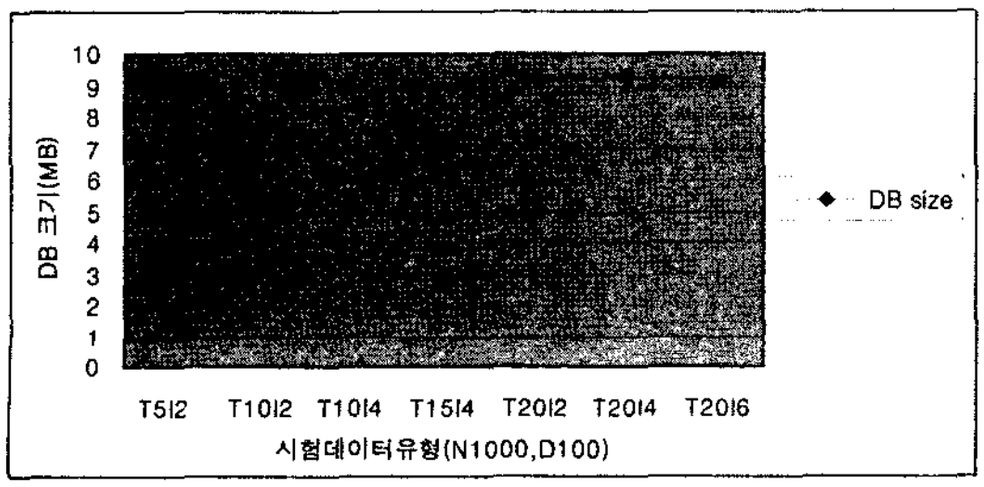
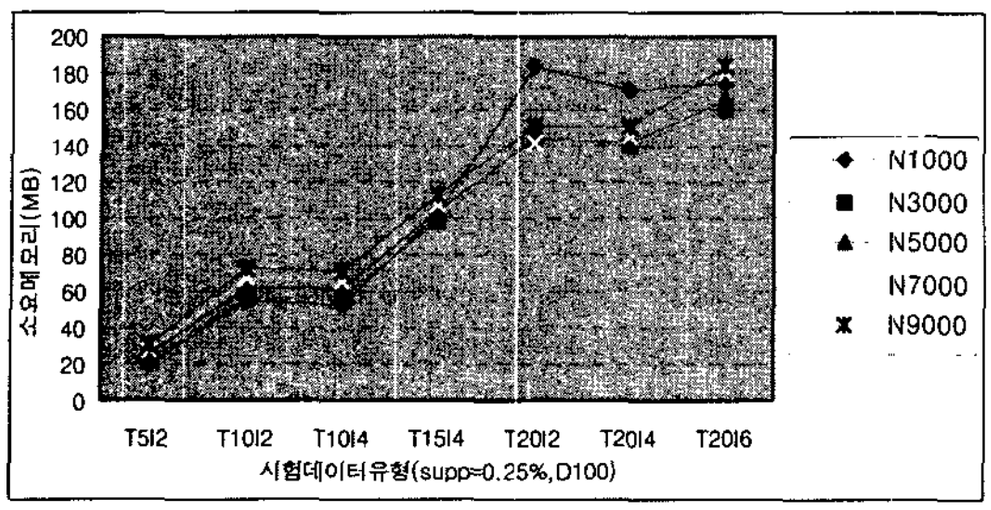
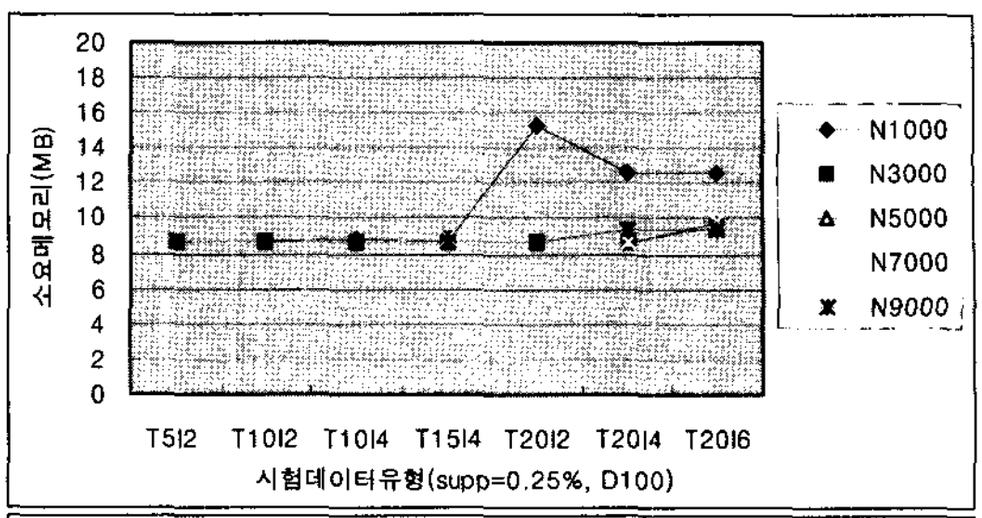
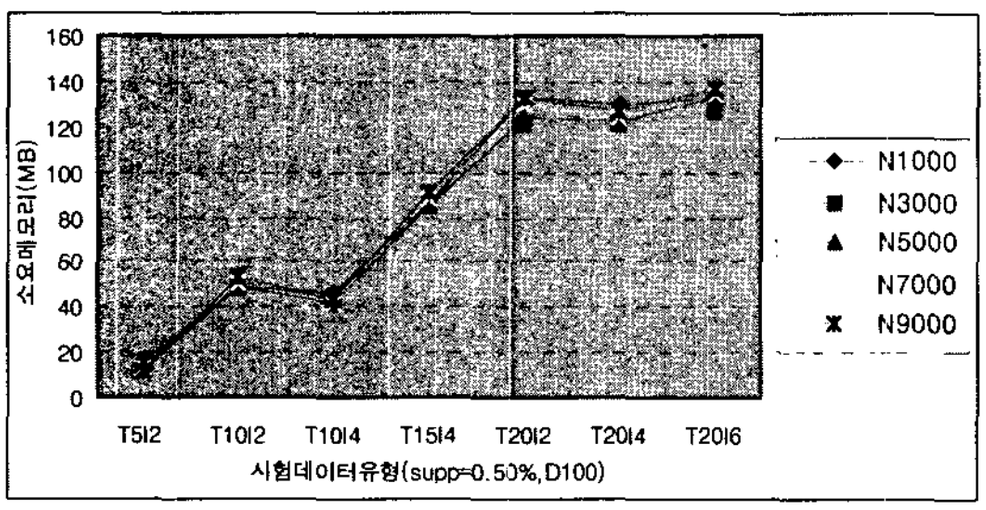


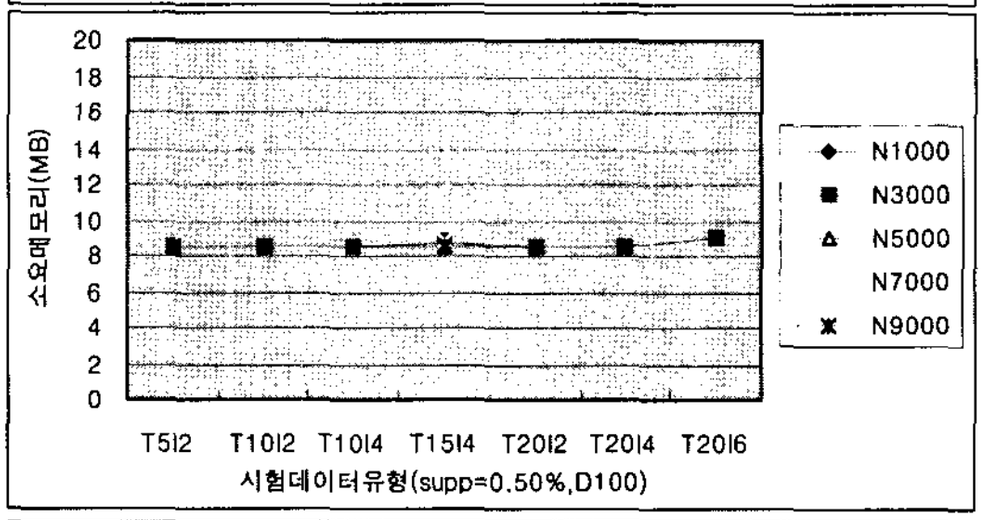
그림 5 데이타의 속성과 DB 크기의 관계

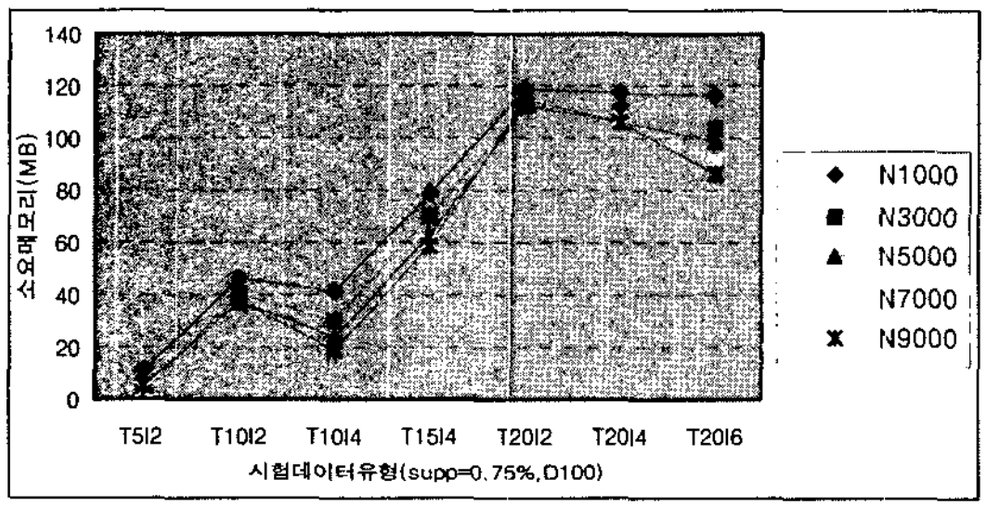
그림 7 DB 크기와 *DHP* 소요 메모리 량의 관계











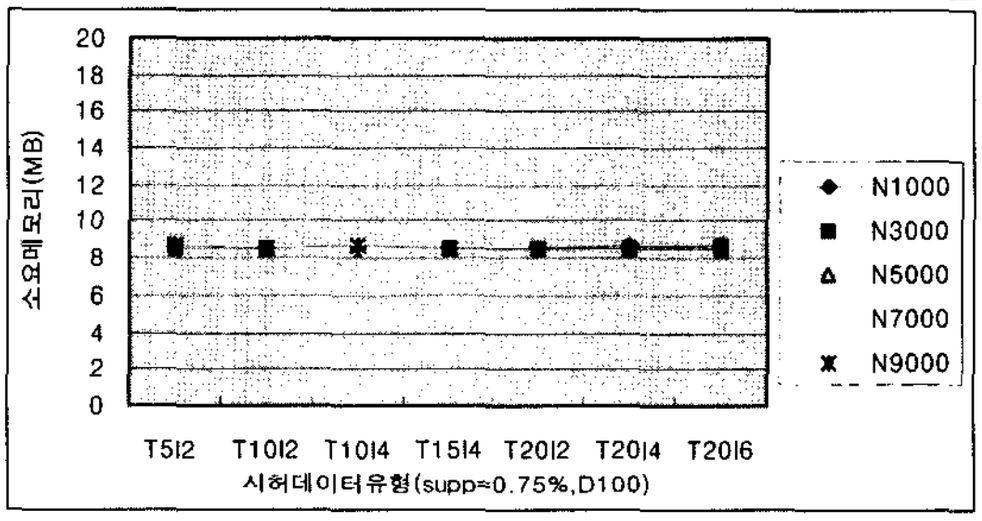


그림 8 *FP-tree* 알고리즘의 메모리 성능 특성

그림 9 *DHP* 알고리즘의 메모리 성능 특성

보인 *DHP*의 경우에는 유형에 관계 없이 입력 데이타의 크기에 따른 메모리 요구량의 변화가 거의 존재하지 않는다.

4.2.3 DB 속성에 따른 소요 시간 변화 특성

그림 10에 그림 8, 그림 9에 사용된 것과 동일한 데이타에 대한 *FP-tree*와 *DHP* 알고리즘의 실행 시간 성능의 측정 결과를 보였다. 이 그림으로부터 유추되는 사실은 *FP-tree*의 경우 거래의 평균 길이나 빈발 항목 집합의 평균 길이에 따른 메모리 소요량과 실행 시간은 민감한 비례 관계에 있으나, 기준 지지도에 따른 상관 관계는 상당히 희박한 비례관계에 있는 점이다. 즉, 그림 8에서 지지도가 0.25%, 0.50%, 0.75%로 높아지면서 메모리 소요량은 점차 줄어들고 있으나, 그림 10의 실행 시간에서의 변화는 미미한데, 이는 소요 메모리 크기가 감소한다고 해서 실행 시간이 반드시 그만큼 감소하지는 않음을 의미한다. 이를 반대로 풀이하면 요구 메모리 양이 증가하더라도 실행 시간은 크게 증가하지 않는다고 말할 수 있다. 이들 두 가지 관점을 조합하면 *FP-tree* 알고리즘은 실행시간 측면에서 대체로 메모리 요구량이 큰 경우 즉, 거래나 빈발 항목 집합의 길이가 길거나 탐색 대상이 되는 빈발 항목 집합이 큰 경우에 적합하고, 그렇지 않은 경우에는 오히려 효과적이지 못할 수도 있음을 의미한다.

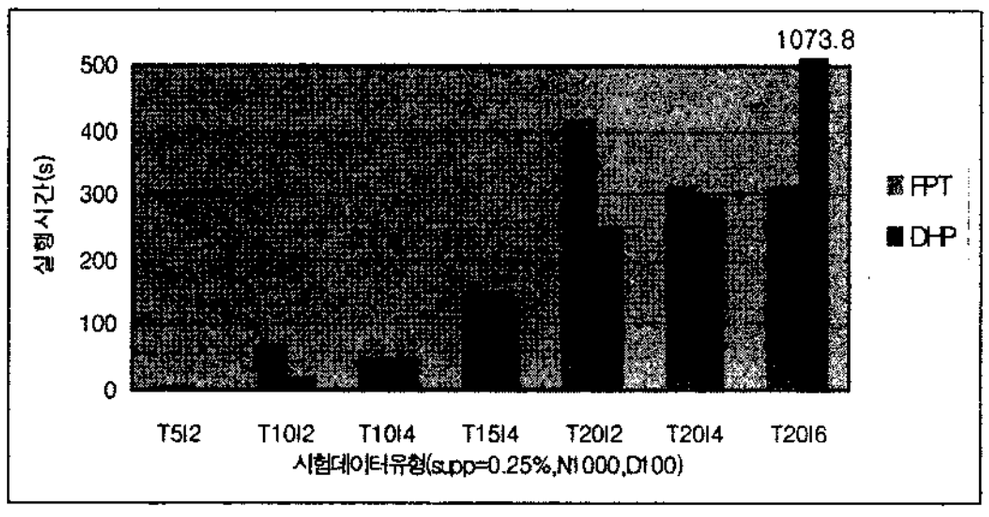
그러나 *DHP*의 경우, 소요 시간이 거래 건수나 데이타의 속성에 따른 DB의 크기에 대체로 비례하는 알고리즘의 보편적 특성을 지니고 있음을 알 수 있다.

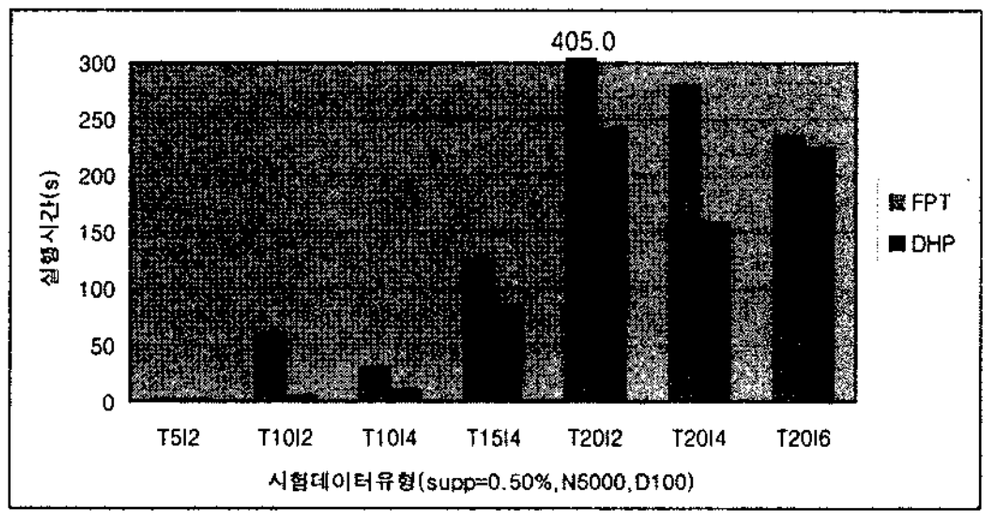
4.3 *FP-tree*와 *DHF* 알고리즘 성능의 비교 종합

• 소요 시간 측면

그림 10에 보인 *FP-tree*(*FPT*)와 *DHP* 알고리즘의 실행 시간 성능 비교에서 살펴본 바와 같이 지지도 0.25%에서 T20I6의 경우를 제외하고는 모두 *DHP* 알고리즘이 우수하다. 특히, 지지도가 0.25%, 0.50%, 0.75%로 커지면서 *DHP* 알고리즘의 상대적 우수성은 더욱 커지고 있다. 그 이유는 초기에 생성되는 *FP-tree*의 방대함에 비하여 거기에 포함된 빈발 항목 집합의 개수는 상대적으로 적기 때문이다. 즉, *FP-tree*의 적중률 혹은 충실성이 낮을 때 초기 *FP-tree*를 생성하기 위한 비용은 상대적으로 높아질 수 밖에 없다는 것이다.

표 3에 그림 10에서 사용된 데이타를 포함하는 표 2의 모든 유형의 시험용 데이타에 대한 두 알고리즘의





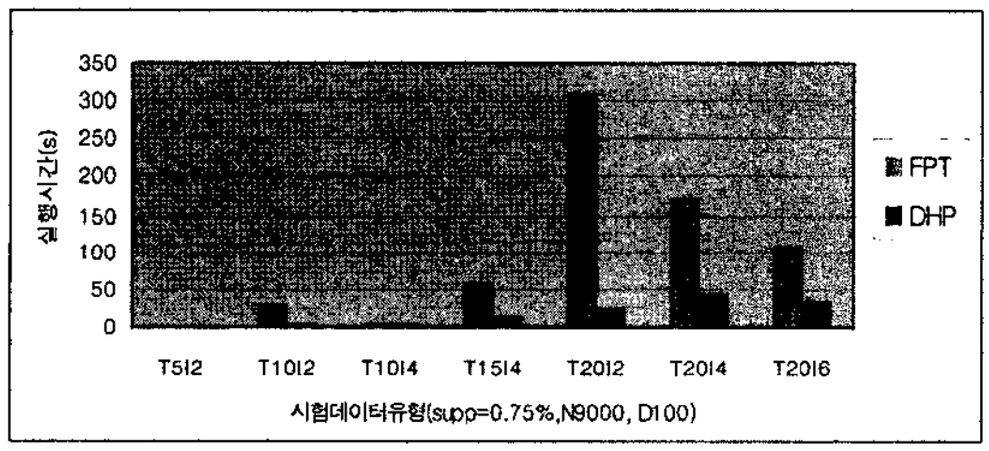


그림 10 *FP-tree*와 *DHP* 알고리즘의 실행시간 성능 특성 비교

실행 시간 성능을 보였다. 이 표로부터 *DHP* 알고리즘의 경우 전체 항목 수가 커질수록 *FP-tree* 알고리즘에 대한 상대적 성능 우수성이 낮아지거나 열등하게 되는데, 이는 직접 해시 테이블에 의한 전지 효율이 저하되기 때문이다.

• 소요 메모리 측면

표 3에서 실시한 실행시간 측정과 동시에 측정된 소요 메모리를 표 4에 보였다. 이들 두 표로부터 *DHP* 알고리즘이 대체적으로 실행시간 성능도 우수하면서도 메모리 사용량이 적어서 양쪽 성능 모두 우수함을 알 수 있다. 특히, 메모리 사용량이 10배이상 차이나는 경우가 다수 있고, 심지어는 100배까지 차이가 나는 경우도 관찰되었다.

5. 결 론

[1]에서 *FP-tree* 알고리즘이 제안되면서 이 알고리즘이 *Apriori*를 포함한 다른 어떤 알고리즘보다 우수하다고 인식된 듯하다. 일반적으로 알고리즘의 대체적인 성능은 최악, 최선, 평균 등 경우에 따른 복잡도로 평가되고 각각의 경우에 따라 우열이 달라질 수도 있다. 여기서 경우란 대부분 입력 데이타의 상태(특성)나 크기를 말하는 것인데, 연관 규칙 탐사 알고리즘에서는 표 2와 같이 거래(T) 및 빈발 항목 집합(I)의 평균 길이, 전체 항목의 개수(N), 거래의 총 개수(D)를 데이타의 주요 특성 인자로 사용한다.

연관 규칙 탐사 알고리즘도 경우에 따라 성능이 달라질 수 있기 때문에 어떤 상황에서도 경우에 관계없이 절대적으로 우수한 알고리즘은 있을 수 없을 것이다. 이 논문은 [3]의 제출 및 기각 과정에서 제기된 "*FP-tree* 알고리즘이 존재하는 상황에서 *DHP* 알고리즘에 대한 연구가 의미를 가질 수 있는가"라는 의문의 해소를 위해 작성되었는데, 이 연구 과정에서 두 알고리즘의 성능 특성에 대한 세밀한 비교·분석이 이루어졌고, 그 결과

표 3 데이타 유형별 *FP-tree*와 *DHP* 알고리즘의 실행 시간 비교

0.25 %	PTF DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		16.4	▽	10.3	▽	8.6	▽	7.8	▽	7.2	▽
		11.8		2.4		2.9		3.3		4.0	
T10I2		143.9	▽	84.2	▽	71.0	▽	63.7	▽	59.7	▽
		16.3		13.2		18.9		18.2		20.7	
T10I4		110.5	▽	59.7	▽	49.5	▽	43.5	▲	40.7	▲
		53.6		49.4		48.7		46.4		49.7	
T15I4		358.4	▽	185.9	▽	152.2	▽	132.6	▽	123.4	▲
		245.7		149.9		141.4		124.0		132.8	
T20I2		921.5	▽	492.9	▽	416.7	▽	367.2	▽	347.4	▽
		612.6		233.2		244.1		101.3		114.1	
T20I4		817.5	▽	382.5	▽	311.0	▽	270.4	▲	271.1	▲
		538.3		298.0		295.7		290.1		320.0	
T20I6		765.3	▲	375.0	▲	312.8	▲	280.4	▲	265.8	▲
		1090.1		1038.8		1073.8		1102.5		1189.8	

0.50 %	FPT DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		13.7	▽	4.8	▽	3.1	▽	2.2	▽	1.8	▲
		1.2		1.5		1.7		2.0		2.3	
T10I2		139.9	▽	76.4	▽	61.3	▽	51.9	▽	47.7	▽
		5.5		5.9		6.9		7.4		8.7	
T10I4		103.7	▽	44.6	▽	29.6	▽	21.5	▽	17.9	▽
		9.2		9.4		10.6		10.2		11.5	
T15I4		347.9	▽	166.2	▽	124.9	▽	101.7	▽	89.6	▽
		69.7		66.9		83.7		70.9		73.7	
T20I2		847.7	▽	480.0	▽	405.0	▽	355.3	▽	335.7	▽
		257.4		110.8		240.5		78.2		81.4	
T20I4		738.5	▽	357.4	▽	280.0	▽	234.3	▽	211.9	▽
		325.6		253.2		157.5		147.8		155.0	
T20I6		704.2	▽	316.2	▽	234.9	▽	189.9	▲	165.2	▲
		472.1		272.5		225.0		228.1		244.8	

0.75 %	FPT DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		8.8	▽	1.5	▽	0.8	▽	0.5	▽	0.5	▲
		0.9		1.1		1.3		1.5		1.8	
T10I2		135.5	▽	64.1	▽	47.8	▽	31.6	▽	31.4	▽
		3.6		4.0		4.6		5.1		6.2	
T10I4		97.4	▽	30.2	▽	15.3	▽	7.0	▽	4.7	▽
		3.3		3.3		3.5		3.7		4.6	
T15I4		341.5	▽	145.4	▽	98.0	▽	71.9	▽	59.3	▽
		21.0		17.1		19.2		17.7		20.1	
T20I2		831.2	▽	468.5	▽	388.1	▽	333.7	▽	312.4	▽
		92.3		33.3		45.2		35.4		38.5	
T20I4		727.5	▽	333.8	▽	247.2	▽	198.9	▽	172.2	▽
		118.4		80.2		99.3		81.5		81.5	
T20I6		690.2	▽	282.5	▽	193.6	▽	138.4	▽	110.0	▽
		115.1		75.2		94.5		67.6		65.1	

단위 : sec, ▲ : FPT 우수, ▽ : DHP 우수

연관 규칙 탐사 알고리즘도 경우에 따라 성능이 달라질 수 있기 때문에 어떤 상황에서도 경우에 관계없이 절대적으로 우수한 알고리즘은 있을 수 없을 것이다. 이 논문은 [3]의 제출 및 기각 과정에서 제기된 "*FP-tree* 알고리즘이 존재하는 상황에서 *DHP* 알고리즘에 대한 연구가 의미를 가질 수 있는가"라는 의문의 해소를 위해 작성되었는데, 이 연구 과정에서 두 알고리즘의 성능 특성에 대한 세밀한 비교·분석이 이루어졌고, 그 결과

표 4 데이타 유형별 *FP-tree*와 *DHP* 알고리즘의 메모리 사용량 비교

0.25 %	FPT / DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		19.1	▽	21.8	▽	24.7	▽	28.1	▽	31.0	▽
		0.9		1.0		1.1		1.1		1.2	
T10I2		**54.7**	▽	**56.9**	▽	**61.2**	▽	**67.2**	▽	**71.3**	▽
		1.2		**1.2**		**1.3**		**1.3**		**1.5**	
T10I4		53.4	▽	56.1	▽	62.3	▽	64.8	▽	70.1	▽
		1.3		2.9		4.3		7.8		6.0	
T15I4		**101.4**	▽	**98.2**	▽	**103.0**	▽	**107.2**	▽	**112.9**	▽
		2.4		**3.5**		**4.8**		**7.8**		**6.6**	
T20I2		183.3	▽	146.1	▽	143.3	▽	141.1	▽	150.5	▽
		8.4		4.3		4.2		3.5		3.5	
T20I4		**170.4**	▽	**146.3**	▽	**139.8**	▽	**142.0**	▽	**146.3**	▽
		5.7		**4.8**		**8.5**		**8.5**		**6.6**	
T20I6		174.1	▽	159.1	▽	165.9	▽	179.3	▽	183.5	▽
		9.4		8.8		9.1		8.5		9.4	

0.50 %	FPT / DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		15.8	▽	12.8	▽	12.6	▽	12.9	▽	13.0	▽
		0.9		0.9		1.0		1.0		6.0	
T10I2		**49.5**	▽	**49.2**	▽	**48.9**	▽	**50.5**	▽	**52.9**	▽
		1.0		**1.0**		**1.**		**1.1**		**1.2**	
T10I4		46.5	▽	44.4	▽	41.3	▽	41.3	▽	43.8	▽
		1.0		1.0		1.1		1.1		1.2	
T15I4		**86.0**	▽	**84.6**	▽	**85.5**	▽	**87.7**	▽	**90.5**	▽
		1.3		**1.5**		**1.6**		**1.5**		**1.6**	
T20I2		132.0	▽	121.4	▽	124.2	▽	129.5	▽	132.4	▽
		2.0		1.6		2.7		1.5		1.6	
T20I4		**129.3**	▽	**121.2**	▽	**122.8**	▽	**124.6**	▽	**127.5**	▽
		2.2		**2.2**		**3.2**		**3.2**		**3.2**	
T20I6		32.0	▽	126.7	▽	132.5	▽	134.6	▽	136.4	▽
		2.2		3.2		6.5		8.2		8.3	

0.75 %	FPT / DHP	N1000		N3000		N5000		N7000		N9000	
T5I2		11.4	▽	5.4	▽	4.5	▽	4.3	▽	4.8	▽
		0.9		0.9		1.0		1.0		1.1	
T10I2		**46.8**	▽	**40.2**	▽	**38.1**	▽	**37.0**	▽	**36.9**	▽
		0.9		**1.0**		**1.0**		**1.1**		**1.1**	
T10I4		41.6	▽	29.5	▽	23.6	▽	18.8	▽	18.8	▽
		0.9		0.9		1.0		1.0		1.1	
T15I4		**78.9**	▽	**69.9**	▽	**63.0**	▽	**59.6**	▽	**58.8**	▽
		96.8		**1.1**		**1.1**		**1.1**		**1.2**	
T20I2		118.3	▽	112.8	▽	112.7	▽	115.2	▽	114.8	▽
		132.0		1.1		1.2		1.2		1.2	
T20I4		**117.9**	▽	**110.6**	▽	**106.2**	▽	**107.7**	▽	**106.7**	▽
		1.3		**1.3**		**1.5**		**1.5**		**1.5**	
T20I6		116.3	▽	104.0	▽	99.3	▽	91.0	▽	86.7	▽
		1.5		1.32		1.6		1.3		1.32	

단위 : MB, ▲ : FPT 우수, ▽ : DHP 우수

FP-tree 알고리즘은 DB 크기에 따른 요구 메모리 증가율이 높고, 빈발 항목 집합에 대한 충실도에 관계없이 기본적으로 소요되는 메모리가 많은 반면, *DHP* 알고리즘은 DB 크기에 관계없이 요구되는 메모리 양이 일정한 한계를 벗어나지 않는 것으로 나타났다. 또한 *FP-tree* 알고리즘은 DB 크기에 따른 소요 시간의 비례 관계가 *DHP* 알고리즘 보다 희박한 것으로 관찰되어 두 알고리즘 사이에 절대적인 우열 관계는 존재할 수 없다는 결론으로 귀결되었다.

그림 10, 표 3, 표 4에서 보는 바와 같이 표 1의 UNIX 운영체제를 탑재한 범용 시스템에서 표 2의 특성을 가지는 데이타에 대한 *FP-tree*와 *DHP* 알고리즘의 성능비교 결과, 소요 메모리와 실행시간 양쪽 모두 *DHP* 알고리즘의 성능이 상당히 우수하게 측정되었다. 이는 예상했던 수준을 크게 벗어나는 것으로 매우 흥미로운 결과가 아닐 수 없다.

앞으로, 거래의 평균 길이나 거래의 건수가 매우 큰 경우 등 좀더 극단적인 상황하에서의 성능도 비교해 볼 필요성이 있겠으나, 이번 결과만으로도 *FP-tree* 알고리즘이 결코 어떤 경우에도 *DHP*보다 우수한 절대적인 알고리즘은 아니라는 점을 말하기에 충분하리라고 본다. 따라서, *DHP* 등 다른 고전적인 알고리즘들에 대한 연구가 소홀히 다뤄져서는 아니될 것이다.

참 고 문 헌

[1] J. Han, J. Pei, and Y. Yin, *"Mining frequent patterns without candidate generation,"* Proceedings ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'00), pp. 1-12, May 2000.

[2] J. S. Park, M.-S. Chen and P. S. Yu, *"An Effective Hash-Based Algorithm for Mining Association Rules,"* Proceedings of ACM SIGMOD, pp. 175-186, 1995.

[3] 이형봉, 김진호, "DHP 연관 규칙 탐사 알고리즘에서 직접 해시 테이블과 해시 트리를 위한 효율적인 해싱 방안", 정보과학회 제출논문(DB06-01-19-01-2), 2006년 1월.

[4] R. Agrawal, T. Imielinski and A. Swami, *"Mining Association Rules between Sets of Items in Large Databases,"* Proceedings of ACM SIGMOD on Management of Data, pp. 207-216, May 1993.

[5] R. Agrawal and R. Srikant, *"Fast Algorithms for Mining Association Rules,"* Proceedings of the 20th International Conference on Very Large Databases, pp. 487-499, September 1994.

[6] A. Pietracaprina, D. Zandolin, *"Mining Frequent Itemsets Using Patrica Tries,"* Proceedings of IEEE ICDM Workshop Frequent Itemset Mining Implementations, CEUR Workshop Proceedings, Vol.80, November 2003.

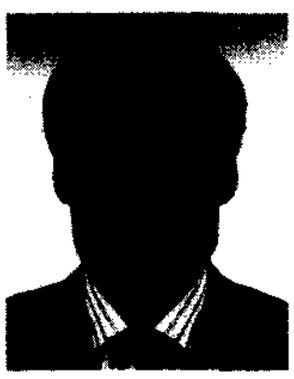
[7] G. Grahne, J. Zhu, *"High Performance Mining Frequent of Maximal Frequent Itemsets,"* Proceedings of SIAM Workshop High Performance Data Mining: Pervasive and Data Stream Mining, May 2003.

[8] G. Grahne, J. Zhu, *"Fast Algorithms for Frequent Itemset Mining Using FP-trees,"* IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.10, pp. 1347-1362, October 2005.

[9] M. Adnan, R. Alhajj, K. Barker, *"Alternative Method for Incrementally Constructing th FP-Tree,"* Proceedings of 3rd Int'l IEEE Conference Intelligent Systems, pp. 494-499, September 2006.

[10] 이재문, 박종수, "복합 해쉬 트리를 이용한 효율적인 연관 규칙 탐사 알고리즘", 정보과학회논문지(B) 제 26권, 제 3호, pp. 343-352, 1999년 3월.

[11] D.L. Yang, C. T. Pan, Y. C. Chung, *"An Efficient Hash-Based Method for Discovering the Maximal Frequent Set,"* Proceedings of 25th Annual Int'l Computer Software and Applications Conference (COMPSAC'0), pp. 511-516, October 2001.
[12] 이재문, "대용량 주기억장치 시스템에서 효율적인 연관 규칙 탐사 알고리즘", 정보처리학회논문지D 제9-D권, 제4호, pp. 579-586, 2002년 8월.
[13] 이형봉, "완전 해싱을 위한 DHP 연관 규칙 탐사 알고리즘의 개선 방안", 정보과학회논문지:데이타베이스, 제31권, 제2호, pp. 91-98, 2004년 4월.
[14] 이형봉, "FP-tree 연관 규칙 탐사 알고리즘의 구현 및 성능 특성", 정보처리학회 추계학술발표대회논문집, 제13권, 제2호, pp. 337-340, 2006년 11월.
[15] R. Agrawal and et al, *"Synthetic Data Generation Code for Associations and Sequential Patterns,"* http://www.almaden.ibm.com/cs/quest



이 형 봉

1984년 2월 서울대학교 계산통계학과 학사. 1986년 2월 서울대학교 계산통계학(전산과학)과 석사. 2002년 2월 강원대학교 컴퓨터과학과 박사. 1986년 3월~1993년 12월 LG전자 컴퓨터연구소 선임연구원. 1994년 2월~1999년 2월 한국디지탈㈜ 책임컨설턴트. 1999년 3월~2004년 2월 호남대학교 조교수. 2004년 3월~현재 강릉대학교 부교수. 관심분야는 임베디드 시스템, 센서네트워크, 데이타마이닝 알고리즘

김 진 호

정보과학회논문지 : 데이타베이스
제 35 권 제 2 호 참조