DHP 연관 규칙 탐사 알고리즘을 위한 효율적인 해싱 메카니즘

이 형 봉†

요 약

Apriori 알고리즘에 기반한 연관 규칙 탐사 알고리즘들은 후보 빈발 항목 집합의 계수 관리를 위한 자료구조로 해시 트리를 사용하고, 많은 시간이 그 해시 트리를 검색하기 위해 소요된다. *DHP* 연관 규칙 탐사 알고리즘은 해시 트리에 대한 검색 시간을 절약하기 위해 검색 대상인 후보 빈발 항목 집합의 개수를 최대한 줄이고자 노력한다. 이를 위해 사전에 예비 후보 빈발 항목 집합에 대한 간편 계수를 실시한다. 이 때, 예비 계수에 필요한 계산 부담을 줄이기 위해 아주 간단한 직접 해시 테이블 사용을 권고한다. 이 논문에서는 *DHP* 연관 규칙 탐사 알고리즘의 단계 2에서 사전 전지를 위해 사용되는 직접 해시 테이블 H_2와 모든 단계에서 후보 빈발 항목 집합의 계수를 위해 사용되는 해시 트리 C_k에 적용될 수 있는 효율적인 해싱 메카니즘을 제안하고 검증한다. 검증 결과 일반적인 단순 제산(mod) 연산 방법을 사용했을 때보다 제안 방법을 적용했을 경우 최대 82.2%, 평균 18.5%의 성능 향상이 얻어지는 것으로 나타났다.

키워드 : DHP, 직접 해시 테이블, 해시 트리

An Efficient Hashing Mechanism of the DHP Algorithm for Mining Association Rules

Hyung Bong Lee†

ABSTRACT

Algorithms for mining association rules based on the *Apriori* algorithm use the hash tree data structure for storing and counting supports of the candidate frequent itemsets and the most part of the execution time is consumed for searching in the hash tree. The *DHP(Direct Hashing and Pruning)* algorithm makes efforts to reduce the number of the candidate frequent itemsets to save searching time in the hash tree. For this purpose, the *DHP* algorithm does preparative simple counting supports of the candidate frequent itemsets. At this time, the *DHP* algorithm uses the direct hash table to reduce the overhead of the preparative counting supports. This paper proposes and evaluates an efficient hashing mechanism for the direct hash table H_2 which is for pruning in phase 2 and the hash tree C_k, which is for counting supports of the candidate frequent itemsets in all phases. The results showed that the performance improvement due to the proposed hashing mechanism was 82.2% on the maximum and 18.5% on the average compared to the conventional method using a simple mod operation.

Key Words : DHP, Direct Hash Table, Hash Tree

1. 서 론

맥주와 기저귀의 상품 진열 문제[1] 해결로 대표되는 연관 규칙 탐사 알고리즘은 마케팅 분야 뿐만 아니라, 과학, 의학, 인구 통계, 금융 등 방대한 데이터가 축적된 곳에서 긴요하게 활용될 수 있는데, 이 때 데이터 양이 방대하므로 그 처리 과정이 효율적이지 못하면 곤란하다.

연관 규칙 탐사 알고리즘은 처리 속도의 향상을 위하여 꾸준히 진화되어 왔다[2, 6]. 전체 항목들의 모든 조합 즉, 모든 부분 집합에 대한 계수공간을 마련한 후, 각 거래에 포함된 부분 집합들을 찾아 계수하는 단순 알고리즘에서 시작하여 탐색을 단계별로 진행하되, 현재까지의 빈도 통계를 적용하여 전 단계의 빈발 항목 집합으로부터 후보 빈발 항목 집합을 도출하여 단계를 결정하는 *AIS(finding All frequent Item-Sets)* 알고리즘[2], 빈도 통계 대신 각 단계별 빈발 항목 집합의 길이를 1, 2, 3, …과 같이 정확히

※ 본 연구는 2005년도 강릉대학교 학술연구조성비에 의해 수행되었음.
† 종신회원 : 강릉대학교 컴퓨터공학과 조교수
논문접수 : 2006년 5월 25일, 심사완료 : 2006년 8월 25일

1씩 늘려가면서 후보 빈발 항목 집합을 유도하여 단계별로 진행하는 *Apriori* 알고리즘[3], 후보 빈발 항목 집합의 수를 줄이기 위해 사전 전지 정보를 활용하는 *DHP* 알고리즘[4]으로 발전했고, 이후 *CHT(Compound Hash Tree)* 알고리즘[5]은 *DHP*알고리즘을 바탕으로 매 단계마다 거쳐야 하는 거래 데이터베이스대한 탐사 횟수를 줄이기 위해 여러 단계를 한꺼번에 처리하는 방안을 제안하였다. *F.P(Frequent Pattern) tree* 알고리즘[6]은 데이터 베이스 탐사 내용을 재구성하여 저장한 빈발 패턴 트리(frequent pattern tree)로부터 빈발 항목 집합을 직접 도출해 냄으로써 기존의 단계별 후보 빈발 항목 집합 생성이라는 절차적 접근이 아닌 새로운 접근 방법을 제안하여 *Apriori* 알고리즘보다 우수함을 보이고 있다.

위의 진화된 연관 규칙 탐사 알고리즘들의 성능적 우열은 데이터의 크기와 구성 형태, 메모리와 디스크 등 사용하는 컴퓨터 시스템의 특성에 따라 달라질 수 있다. 따라서, 이 논문에서는 전통적으로 활용 빈도가 높았던 *Apriori* 식 접근 방법의 하나인 *DHP* 알고리즘의 성능 개선 방안을 모색하여 그 활용도를 높이고자 한다.

Apriori 알고리즘은 후보 빈발 항목 집합을 저장하고 계수하기 위한 구체적인 자료구조로서 해시 트리를 제안하였고, *DHP* 알고리즘은 해시 트리 외에 사전 전지를 위한 직접 해시 테이블을 도입하여 *Apriori* 식 접근 측면의 다른 진화된 알고리즘들의 토대를 이룬다. 이들 알고리즘들의 진화는 주로 C_k 및 D_k (<표 1> 참조)의 축소와 데이터베이스 탐사 횟수 절약에 초점을 두고 있고, 해시 전략 등 세부 구현 방안은 자세히 다루지 않는다. 즉, 해시 트리에 대한 개념은 제안하고 있으나 이의 세부적인 구현 방법론 제시나 분석은 이루어지지 않았다. 뿐만 아니라, *DHP* 알고리즘에서 각 단계의 사전 전지를 위해 도입하는 직접 해시 테이블의 해싱 방안은 일반적인 단순 제산(mod) 연산 개념으로 제시될 뿐, 그 밖에 좀더 개선할 수 있는 구체적인 방안이 있는지에 대한 연구는 되어있지 않다. 이는 연관 규칙 탐사 알고리즘이 데이터베이스 탐사 횟수 등 주로 거시적 관점의 단계적 처리 절차의 효율화를 목표로 진화되어 왔기 때문이다.

이 논문에서는 *DHP* 알고리즘을 구현하는 관점에서, 단계 2 해시 트리 C_2의 사전 전지 목적으로 사용되는 직접 해시 테이블 H_2의 적중률을 높이기 위한 효율적인 해싱 방안과, 모든 단계에서 해시 트리 C_k가 내부 노드 분화를 일으킬 때 요구되는 항목들의 그룹핑을 위한 효율적인 해싱 방안을 도출하여 그 각각을 일반적인 방법과 비교·분석하고 그 효용성을 검증한다. 이를 위해 2장에서 *DHP* 알고리즘의 특성을 살펴보고, 3 장에서 직접 해시 테이블과 해시 트리를 위한 효율적인 해싱 방안을 제안한 후, 4 장에서 제안된 방안들을 구현하여 *DHP* 알고리즘에 적용하고, 그 성능 측정 결과를 분석한다. 그리고 마지막 5 장의 결론으로 이 논문을 맺는다.

2. *DHP* 알고리즘의 특성

2.1 *DHP* 알고리즘과 주요 자료구조

2.1.1 자주 사용되는 기호

연관 규칙 탐사 알고리즘은 대량의 거래 자료로부터 거래 항목들 사이에 존재하는 동시 거래 관련성을 발견하는 것으로, 지지도(support), 신뢰도(confidence), 항목 집합(itemset), 빈발 항목 집합(frequent itemset or large itemset), 후보 빈발 항목 집합(candidate frequent itemset), 최대 빈발 항목 집합(maximal frequent itemset), 길이가 k인 항목 집합(k-itemset), 전체 항목 집합 등의 용어와 <표 1>의 기호 들을 사용한다[2, 5, 7, 8].

〈표 1〉 연관 규칙 탐사에서 사용되는 기호

기 호	의 미
D_k, t_k, t	단계 k에서의 데이터베이스와 거래, $t_k, t \in D$
L_k, l_k, l	단계 k에서의 {빈발 항목 집합}, $l_k, l \in L_k$
C_k, c_k, c	단계 k에서의 {후보 빈발 항목 집합}, $c_k, c \in C_k$
H_k, h_k	단계 k에서의 직접 해시 테이블과 해싱 함수
I, i_j	전체 항목 집합, $i_j \in I$
Nnnnn	데이터 전체 항목 집합의 크기($\|I\|$)가 *nnnn* 임 예:*N1000*은 총항목개수가 1,000인 경우를 말함
TxIyDz	거래의 평균 크기 x, 빈발 항목 집합의 평균 크기 y, 총 거래 건수 zx1,000개인 거래데이터 예:*T15I4D100*는 거래에 포함된 항목 개수가 평균 15, 빈발 항목집합의 길이가 평균 4, 전체 거래 건수가 100x1000인 거래 데이터를 말함

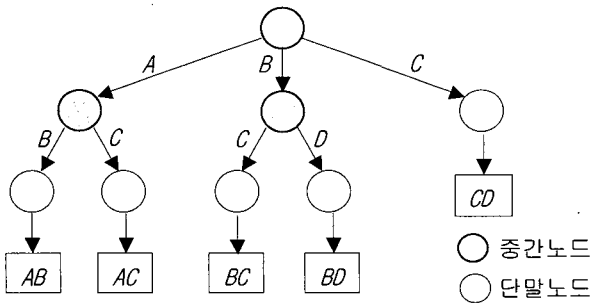
2.1.2 해시 트리

해시 트리(hash tree)는 <표 1>의 C_k를 저장하여 거래 t_k에 포함된 길이 k인 항목 집합의 존재를 탐색하고 계수를 하기 위한 공간으로, (그림 1) 형태의 구조를 갖는다. 이 그림은 후보 빈발 항목 집합 {A, B}, {A, C}, {B, C}, {B, D}, {C, D}가 저장된 모습을 보이고 있는데, 중간 노드는 후보 빈발 항목 집합을 구성하는 항목들의 해시 값에 의해 차례로 결정되고, 단말 노드는 저장 후보 빈발 항목 집합의 개수가 주어진 임계값을 넘으면 다음 번 항목을 해싱하여 아래 단계로 분화한다. (그림 1)에서 단말 노드 분화를 위한 저장 후보 빈발 항목 집합의 개수는 2 이상이고, C 이하에는 오직 한 개의 후보 빈발 항목 집합만이 존재하므로 항목 D에 의한 노드 분화는 이루어지지 않았다.

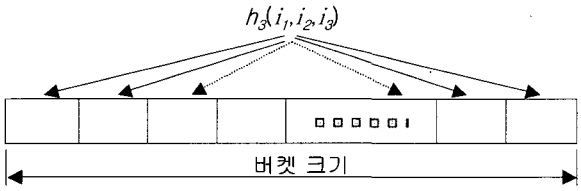
이 논문에서는 C_k와 해시 트리를 동일 용어로 사용한다.

2.1.3 직접 해시 테이블

직접 해시 테이블(direct hash table)은 <표 1>의 H_k를 저장하고 계수하기 위한 공간으로 L_{k-1}로부터 C_k를 생성할 때 참조되어 가능한 한 $|C_k|$(C_k 에 포함된 후보 빈발 항목 집합의 개수)를 줄이기 위해 활용된다. 예를 들어 L_2로부터 후보 빈발 항목 집합 C_3인 {A,B,C}가 유도되었다면 이를 곧바로 C_3에 등록하지 않고 C_2를 계수할 때 미리 생성해 놓은 H_3의 h_f(A, B, C) 위치에 대응되는 계수 값을 참조하여 최소 지지도를 만족하지 못할 경우에는 그대로 버린다. 이 때 H_3는 여러 항목 집합의 지지도가 합쳐진 개략적인 지지도만 유지하므로 계수가 간단하고 신속해야 한다. 즉, 직접 해시 테이블 공간은 전체 예상 후보 빈발 항목 집합 개수에 비해 매우 소량이므로 그 각각의 버켓에는 여러 개의 후보 빈발 항목 집합의 계수 값이 합산된다.



(그림 1) C_2를 위한 해쉬 트리(k=2)의 예



(그림 2) H_3를 위한 직접 해시 테이블(k=3)

(그림 2)에 직접 해시 테이블의 개념적 구조를 보였고, 이 논문에서는 H_k와 직접 해시 테이블을 동일 용어로 사용한다.

2.1.4 *DHP* 알고리즘

(그림 3)에 *DHP* 연관 규칙 탐사 알고리즘을 보였다. 이 그림에서 라인 6의 H_2는 라인 11과 라인 38에서 C_2를 생성할 때 참조되어 $|C_2|$를 줄이는데 매우 큰 영향을 미친다[4]. 이 후부터 단계 k가 진행되는 도중 라인 17과 라인 45에서 C_{k+1}를 위한 H_{k+1}가 생성되고 (Part 2), 더 이상 H_{k+1}의 의미가 없을 때는 이 과정이 생략된다(Part 3)[4].

```
1  s = a minimum support;                    /* - Part 1 - */
2  set all buckets of H2 to zero;            /* hash table */
3  forall transactions t∈D do begin
4    insert & count 1-items occurrences in hash tree;
5    forall 2-subsets x of t do
6      H2[h2(x)]++;
7  end
8  L1 = {c|c.count≥s, c is s leaf node of hash tree};
9  k = 2;  Dk = D;                           /* - Part 2 - */
10 while(|{x|Hk[x] ≥s}| ≥LARGE) {
11   gen_candidate(Lk-1, Hk, Ck); /* make hash table */
12   set all the buckets of Hk+1 to zero;
13   Dk+1 = ∅;
14   forall transactions t∈D do begin
15     count_support(t, Ck, k, t');  /* t'⊆t */
16     if (|t'| > k) then do begin
17       make_hasht(t', Hk, k, Hk+1, t'');
18       if (|t''| > k) then Dk+1 = Dk+1 ∪{t''};
19     end
20   end
21   Lk = {c∈Ck|count≥s};
22   k++;
23 }
```

```
24 gen_candidate(Lk-1, Hk, Ck);              /* - Part 3 - */
25 while(|Ck| > 0) {
26   Dk+1 = ∅;
27   forall transactions t∈D do begin
28     count_support(t, Ck, k, t');  /* t'⊆t */
29     if (|t'| > k) then Dk+1 = Dk+1 ∪{t'};
30   end
31   Lk = {c∈Ck|count≥s};
32   if ((|Dk| = 0) then break;
33   Ck+1 = apriori_gen(Lk); k++;
34 }
```

```
35 Procedure gen_candidate(Lk-1, Hk, Ck)
36   Ck = ∅;
37   forall c = cp[1]·... ·cp[k-2]·cp[k-1]·cq[k-1],
38             cp, cq ∈ Lk-1, |cp ∩cq| = k-2  do
39     if (Hk[hk(c)] ≥s) then
40       Ck = Ck ∪{c}; /* insert c into hash tree*/
41 End Procedure
```

```
42 Procedure make_hasht(t', Hk, k, Hk+1, t'')
43   forall (k+1)-subsets x (=t'i1·...·tik+1) of t' do
44     if (for all k-subsets y of x, Hk[hk(y)] ≥s)
45       then do begin
46         Hk+1[hk+1(x)]++;
47         for (j = 1; j ≤k+1; j++) a[ij]++;
48       end
49   for (i=0, j=0; i < |t'|; i++)
50     if (a[i] > 0) then do begin
51       t''j = t'i; j++;
52     end
53 End Procedure
```

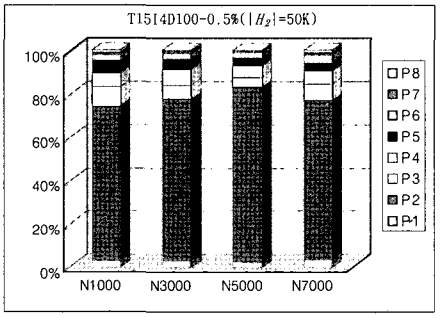
(그림 3) *DHP* 연관 규칙 탐사 알고리즘

2.2 *DHP* 알고리즘의 특성 고찰

이 논문과 관련이 깊은 *DHP* 알고리즘의 특성으로 단계별 소요 시간 추이, H_k 및 C_k 정책이 전체 소요 시간에 미치는 영향을 들 수 있는데, 여기서는 (그림 3)의 알고리즘을 실행시킨 결과로써 그 특성들을 조명한다.

2.2.1 단계별 소요시간 추이

DHP 알고리즘을 포함한 대부분의 연관 규칙 탐사 알고리즘들의 공통된 특징 중 하나는 전체 실행시간 대비 단계 2의 실행시간 비중이 70~80%로 아주 높다는 점인데 이는 후보 빈발 항목 집합 $|C_2|$가 매우 크기 때문이다. (그림 4)에 몇 가지 전형적인 연관 규칙 탐사 시험용 데이터[7]에 대한 *DHP* 알고리즘의 단계별 소요시간 비율을 보였다.

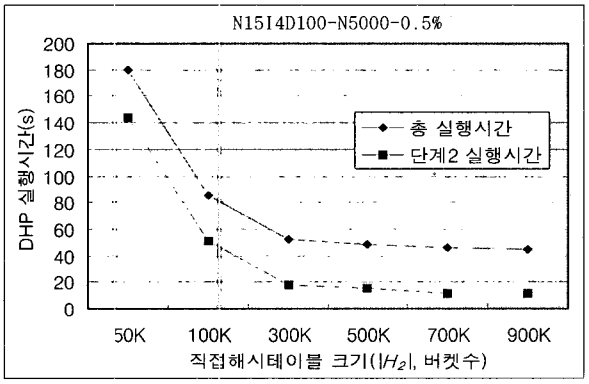


(그림 4) *DHP* 알고리즘의 단계별 소요시간 특성

2.2.2 $H_k(H_2)$가 소요시간(단계 2)에 미치는 영향

(그림 4)의 특성에 착안하여 $|C_2|$를 줄이기 위해H_2에 의한 C_2의 전지율을 극도로 향상((그림 3)의 라인 6, 38, 39)시키기 위해 [9]는 H_2에 완전 해싱 정책 도입을 제안하였고, [10]은 H_2에 완전 해싱을 도입하되 그 장점을 최대한 활용하기 위해 단계 1과 단계 2를 한꺼번에 처리하는 새로운 알고리즘 *PHP (Perfect Hashing & Pruning)*를 제안하였다. 이들 두 제안은 모두 VLM (Very Large Memory) 시스템[11]을 전제로 하고 있다. 이와 같이 H_2에 완전 해싱 정책을 도입할 경우 ($\binom{|I|}{2}$)개의 계수 공간이 필요하고, 이 경우 C_2는 불필요하고 곧바로 L_2 도출이 가능하다. *N1000*인 경우 약 2MB(sizeof (int)*1,000*999/2)가, *N7000*인 경우 약 100MB (sizeof (int)*7,000* 6,999/2)의 메모리가 필요하다. 이 정도의 메모리 소요는 현대의 VLM 환경에서 큰 부담이 되지 않지만, *N10000*, *N20000*, … 와 같이 $|I|$의 크기가 계속 증가하거나, 다중 프로그래밍의 정도가 높아지는 환경에서는 메모리 사용량에 대한 부담이 심각해 질 수 있기 때문에 $|H_k|$에 대한 상한을 두는 것이 현실적이다.

H_2를 위한 해싱 함수로 단순 제산 연산을 사용했을 때 $|H_2|$가 단계 2 실행 시간에 미치는 영향을 (그림 5)에 보였다.



(그림 5) $|H_k|$가 소요시간에 미치는 영향

2.2.3 C_k가 소요시간에 미치는 영향

소요시간에 영향을 미치는 해시 트리 C_k 정책 요소로 크게 해시 트리 자체를 위한 메모리 사용량과 항목들의 그룹핑을 위한 해싱 함수를 들 수 있다.

• 해시 트리의 메모리 사용량과 검색 성능

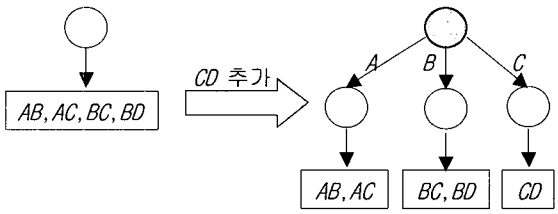
(그림 1)의 해시 트리를 운용하기 위해서는 아래의 서로 상충되는 두 가지 목표에 대한 상호 조율이 필요하다.

- 해시 트리 전체를 위한 메모리 사용량
- 해시 트리의 시간적 검색 성능

만약 해시 트리의 내부 노드 차수(degree of internal node)를 낮춘다면 내부 노드의 개수가 감소하여 메모리 사용량이 줄지만, 외부 노드에서의 후보 빈발 항목 집합의 개수가 늘어나 검색 시간이 늘어난다. 반대로 내부 노드의 차수를 높이면 내부 노드의 개수가 증가하여 메모리 사용량은 늘지만, 외부 노드에서의 후보 빈발 항목 집합의 개수가 줄어서 검색 시간이 단축된다.

〈표 2〉 해시 트리에서 해시그룹 크기와 내부 노드수

전체 항목수	해시그룹 크기	내부노드 수	
		$C_2(k=2)$	$C_3(k=3)$
N1000	30	1,100	36,000
	10	10,000	1,000,000
N7000	30	54,000	12,000,000
	10	490,000	343,000,000



(그림 6) 해시 트리의 노드 분화 과정

해시 트리에서 내부 노드의 차수는 전체 항목들의 해시 값의 충돌 정도 즉, 동일한 해시 값을 갖는 항목들의 집합(해시 그룹)의 평균 크기에 따라 결정된다. <표 2>에 해시 그룹 크기에 따른 내부 노드 최대 필요 개수의 예를 보였다.

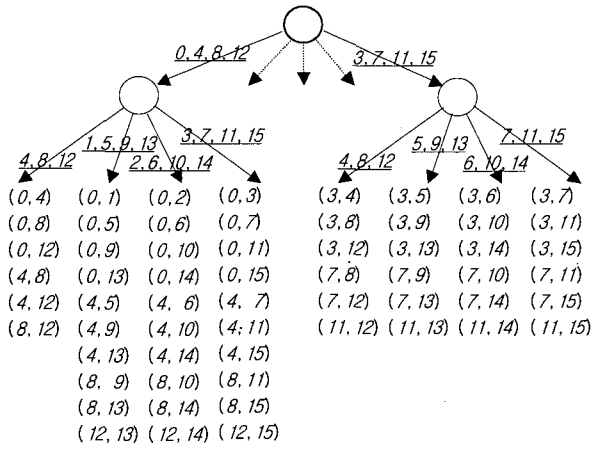
내부 노드의 수를 줄이기 위해 *DHP* 알고리즘은 해시 트리의 모든 외부 노드의 깊이를 처음부터 후보 빈발 항목 집합의 길이만큼 만들지 않고, 외부 노드인 뿌리 노드로부터 시작해서, 해당 노드에 후보 빈발 항목 집합을 수집하다가 그 수가 일정 한계를 넘으면 (그림 6)과 같이 해당 노드를 내부 노드로 전환하고 다음 단계의 외부 노드로 분화 시킨다((그림 3)의 라인 39).

• 해시 트리의 항목 그룹핑을 위한 해싱 함수

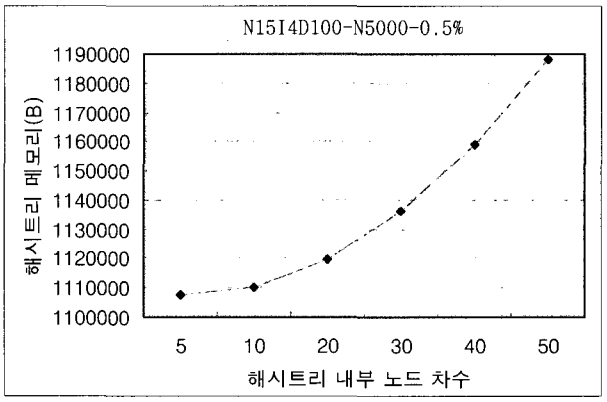
(그림 6)에서 내부 노드를 분화 시킬 때 내부 노드 수를 줄이기 위해 해싱 함수를 사용하여 동일 해시 값을 갖는 항목 그룹을 만든다. 예를 들어, 단계 2 에서 A~P(0~15)의 16개 항목에 대한 조합 가능한 빈발 항목 집합은 <표 3>과 같고, 이를 바탕으로 제산 연산을 사용하는 단순 등간격 그룹핑에 의해 구성한 해시 트리 C_2 의 모습은 (그림 7)과 같다. (그림 8), (그림 9)에는 단순 등간격 그룹핑을 사용했을 때 C_2의 내부 노드 차수가 내부 노드의 개수와 소요시간에 미치는 영향을 보였다.

〈표 3〉 빈발 항목 집합 구성의 특성

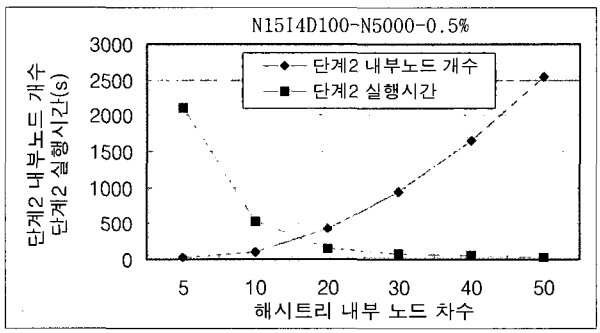
첫항 \ 이항	*A*	*B*	*C*	▫▫▫▫	*N*	*O*	*P*
A		*AB*	*AC*	▫▫▫▫	*AN*	*AO*	*AP*
B			*BC*	▫▫▫▫	*BN*	*BO*	*BP*
C					*CN*	*CO*	*CP*
▫▫						▫▫	▫▫
O							*OP*
P							



(그림 7) 등간격 그룹핑에 의한 C_2의 구성($|I|=16$)



(그림 8) C_2의 내부 노드 차수와 소요 메모리량의 관계



(그림 9) $|C_2|$가 단계 2 소요시간에 미치는 영향

3. H_2와 C_k를 위한 효율적인 해싱 방안

여기서는 2 장에서 언급한 H_2및 C_k정책 각각에 대한 기존의 일반적인 방법을 개선할 수 있는 효율적인 해싱 방안을 제안한다.

3.1 H_2 해싱 방안

3.1.1 일반적인 방법

(그림 3)의 *DHP* 알고리즘은 라인 6에서 직접 해시 테이블 H_2를 만들고, 이를 라인 11, 38, 39의 해시 트리 C_2를 생성할 때 활용한다. 이 때 길이 2의 예비 후보 빈발 항목 집합 $\{i_1, i_2\}$에 대응되는 H_2의 버켓을 결정하기 위한 해싱 함수((그림 3)의 라인 6에서 $H_2()$)로 아래 식 ①의 단순 제산 해싱(simple mod hashing) 함수 $dh_{smh}(i_1, i_2)$를 사용하는 것이 일반적이다.

$$dh_{smh}(i_1, i_2) = (i_1 \times |I| + i_2) \bmod |H_2| \cdots \cdots 식\ ① $$

그런데 식 ①의 해싱 함수 $dh_{smh}(i_1, i_2)$를 사용할 경우 <표 3>의 빈발 항목 집합 구성 특성에 따라 해싱 공간의 분포가 고르지 못하다. 즉, (그림 10)에서 보는 바와 같이 $|H_2|= |I|-1(|H_2|=7)$인 경우를 제외하고는 균일한 해싱 공간을 만족하는 $|H_2|$를 얻기가 어렵고, 주어진 가용 메모리 한계가 $|H_2|=|I|-1$를 만족하리라고 기대하기는 더욱 어렵다. 뿐만 아니라 $|H_2|=|I|+1(|H_2|=9)$를 만족하는 $|H_2|$의 설정은 최악의 경우를 낳는다.

$|H_2|=7$

해시키	x	0,1	0,2	0,3	0,4	0,5	0,6
	0,7	x	x	1,2	1,3	1,4	1,5
	1,6	1,7	x	x	x	2,3	2,4
	2,5	2,6	2,7	x	x	x	x
	3,4	3,5	3,6	3,7	x	x	x
	x	x	4,5	4,6	4,7	x	x
	x	x	x	x	5,6	5,7	x
	x	x	x	x	x	x	6,7
	x	x	x	x	x	x	x
대응버켓	0	1	2	3	4	5	6
히트개수	4	4	4	4	4	4	4

$|H_2|=9$

해시키	x	0,1	0,2	0,3	0,4	0,5	0,6	0,7	x
	x	1,2	1,3	1,4	1,5	1,6	1,7	x	x
	x	2,3	2,4	2,5	2,6	2,7	x	x	x
	x	3,4	3,5	3,6	3,7	x	x	x	x
	x	4,5	4,6	4,7	x	x	x	x	x
	x	5,6	5,7	x	x	x	x	x	x
	x	6,7	x	x	x	x	x	x	x
대응버켓	0	1	2	3	4	5	6	7	8
히트개수	0	7	6	5	4	3	2	1	0

$|H_2|=13$

해시키	x	0,1	0,2	0,3	0,4	0,5	0,6	0,7	x	x	1,2	1,3	1,4
	1,5	1,6	1,7	x	x	x	2,3	2,4	2,5	2,6	2,7	x	x
	x	x	3,4	3,5	3,6	3,7	x	x	x	x	x	4,5	4,6
	4,7	x	x	x	x	x	x	5,6	5,7	x	x	x	x
	x	x	x	6,7	x	x	x	x	x	x	x	x	
대응버켓	0	1	2	3	4	5	6	7	8	9	10	11	12
히트개수	2	2	3	3	2	2	2	3	2	1	2	2	2

(그림 10) $dh_{smh}(i_1, i_2)$에 의한 해싱 공간 분포($|I|=8$)

3.1.2 사상 완전 해싱 방법

사상 완전 해싱(mapped perfect hashing)은 (그림 10)에서 'x'로 표시된 사용되지 않는 해시 키가 차지하는 해싱 공간을 제거하여 $|H_2|$에 관계 없이 모든 유효 해시 키에 대한 해싱 공간을 완전히 균일한분포로 만든다. 이를 위한 한 가지 방안으로 이 논문에서는

모든 유효 키를 일렬로 배열하여 그 각각에 0, 1, 2, …, ((|I|-1)×|I|/2)-1의 완전 순서 번호를 부여한 다음, 그 대응 순서 번호에 단순 제산 연산을 적용하는 해싱 함수 $dh_{mph}(i_1, i_2)$를 제안한다.

모든 유효 해시 키에 대해 완전 순서 번호를 대응시키기 위해서 우선 모든 조합 가능한 두 항목의 조합 (k, j) 대한 k(행) 우선 2차원 배열인 (그림 11)을 고려한다. 이 그림으로부터, 사용되지 않는 항목 조합을 제외시켰을 때 항목 조합 (k, j)에 대한 완전 순서번호는 명백하게 아래 식 ②의 $f_{seqall}(k, j)$로 얻을 수 있다.

$$f_{seqall}(k, j) = k\times(e+1)+j \quad \cdots\cdots \text{식 ②}$$

단, $e=|I|-1$.

유효 항목 조합이 고려된 항목 조합 (k, j), $(k<j)$에 대한 완전 순서 번호를 도출하기 위해서는 식 ②의 $f_{seqall}(k, j)$로부터 0~k행에 포함된 사용되지 않은 모든 항목 조합들의 수 $T_r(k)$만큼 감하면 된다. 즉, 유효 항목 조합이 고려된 두 항목의 조합 (k, j)에 대한 완전 순서 번호는 아래의 식 ③ $f_{seq}(k, j)$로 얻을 수 있다.

$$f_{seq}(k, j) = f_{seqall}(k, j) - T_r(k) \quad \cdots\cdots \text{식 ③}$$

단, $T_r(k)$는 0~k행에 포함된 사용되지 않은 모든 항목 조합의 수.

그런데, 명백하게 i 행에 포함된 사용되지 않은 항목 조합의 수 $L_r(i)$는 i+1 이다. 따라서 0 행부터 k 행까지 포함된 사용되지 않은 항목들의 총 개수 $T_r(k)$는 아래의 식 ④로 얻을 수 있다.

$$Tr(k) = \sum_{i=0}^{k} L_r(i) = \sum_{i=0}^{k} (i+1)$$

$$= \frac{k\times(k+1)}{2}+(k+1) = \frac{(k+1)\times(k+2)}{2} \quad \cdots\cdots \text{식 ④}$$

식 ②, ③, ④로부터 $f_{seq}(k, j)$ 즉, 유효 항목 조합 (k, j)에 대한 완전 순서 번호는 아래와 같이 정리하여 얻을 수 있다.

$$f_{seq}(k, j) = f_{seqall}(k, j) - T_r(k)$$

$$= k\times(e+1)+j - \frac{(k+1)\times(k+2)}{2}$$

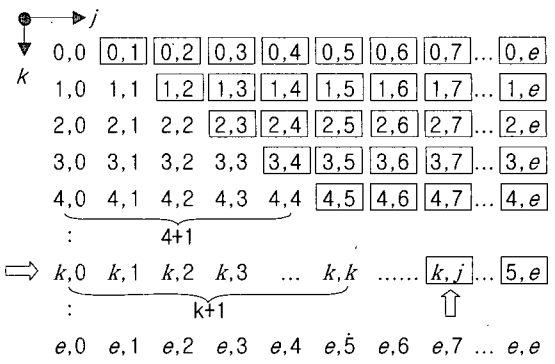
$$= k\times|I|+j - \frac{(k+1)\times(k+2)}{2} \quad \cdots\cdots \text{식 ⑤}$$

마지막으로, 후보 빈발 항목 집합 $\{i_1, i_2\}$에 대한 대응 버켓을 결정하는 사상 완전 해싱 함수 $dh_{mph}(i_1, i_2)$는 식 ⑤에 단순 제산 연산을 적용하여 아래의 식 ⑥으로 구할 수 있다.

$$dh_{mph}(i_1, i_2) = f_{seq}(i_1, i_2) \bmod |H_2|$$

$$= \left(i_1\times|I|+i_2 - \frac{(i_1+1)\times(i_1+2)}{2}\right) \bmod |H_2| \quad \cdots\cdots \text{식 ⑥}$$

(그림 12)에 보인 식 ⑥의 $dh_{mph}(i_1, i_2)$에 의한 해싱 공간 분포가 (그림 10)의 식 ①에 의한 해싱 공간 분포에 비하여 훨씬 균일함을 알 수 있다.



(그림 11) 두 항목 조합 (k, j)의 배열(e=|I|-1)

|H_2|=9 ()는 사상 완전순서번호

해시키									
	0,1 (0)	0,2 (1)	0,3 (2)	0,4 (3)	0,5 (4)	0,6 (5)	0,7 (6)	1,2 (7)	1,3 (8)
	1,4 (9)	1,5 (10)	1,6 (11)	1,7 (12)	2,3 (13)	2,4 (14)	2,5 (15)	2,6 (16)	2,7 (17)
	3,4 (18)	3,5 (19)	3,6 (20)	3,7 (21)	4,5 (22)	4,6 (23)	4,7 (24)	5,6 (25)	5,7 (26)
	6,7 (27)								
대응 버켓	0	1	2	3	4	5	6	7	8
히트 개수	4	3	3	3	3	3	3	3	3

|H_2|=13

해시키													
	0,1 (0)	0,2 (1)	0,3 (2)	0,4 (3)	0,5 (4)	0,6 (5)	0,7 (6)	1,2 (7)	1,3 (8)	1,4 (9)	1,5 (10)	1,6 (11)	1,7 (12)
	2,3 (13)	2,4 (14)	2,5 (15)	2,6 (16)	2,7 (17)	3,4 (18)	3,5 (19)	3,6 (20)	3,7 (21)	4,5 (22)	4,6 (23)	4,7 (24)	5,6 (25)
	5,7 (26)	6,7 (27)											
대응 버켓	0	1	2	3	4	5	6	7	8	9	10	11	12
히트 개수	3	3	2	2	2	2	2	2	2	2	2	2	2

(그림 12) $dh_{mph}(i_1, i_2)$에 의한 해싱 공간 분포(|H|=8)

3.2 C_k ($k\geq 2$)의 해싱 방안

3.2.1 일반적인 방법

일반적으로 해시 트리의 내부 노드를 분기시킬 때((그림 3)의 라인 39), 항목들의 그룹핑을 위한 해싱 함수로 제산 연산을 사용하는 아래 식 ⑦의 단순 등간격 그룹핑(regular interval grouping) 방법 $ch_{rig}(i)$를 사용한다.

$$ch_{rig}(i) = i \bmod |C_{kb}| \quad \cdots\cdots \text{식 ⑦}$$

단, $|C_{kb}|$는 해시 트리 C_k의 내부 노드 차수(해시 버켓 크기)

그러나 이 방법은 (그림 7)($|I|$=16,$|C_{kb}|$=4)에서 보는 바와 같이 해시 그룹의 크기가 왼쪽에서 오른쪽으로 갈수록 점점 적어져서 균형을 이루지 못하므로 해시 트리 검색 성능의 저하 원인이 된다.

3.2.2 대칭 등간격 그룹핑 방법

(그림 7)의 치우침 현상을 완화시키기 위한 방안으로 이 논문에서는 원하는 항목 그룹 수($|C_{kb}|$, 내부 노드 차수)의 2배 만큼의 단순 등간격 항목 그룹을 생성한 후, 좌·우 양쪽 대칭되는 한 쌍씩을 묶어 최종 항목 그룹으로 생성하는 대칭 등간격 그룹핑(symmetric interval grouping) 방법을 제안한다. 예를 들어 (그림 7)의 단순 등간격 그룹핑을 대칭 등간격 그룹핑으로 설정하기 위해서 우선 (그림 13)의 (a)와 같이 단순 등간격 그룹핑 방법($ch_{rig}(i)$)으로 원하는 항목 그룹 수 4($|C_{kb}|$=4)의 2 배인 8개의 항목 그룹을 만든다. 그런 다음, (그림 13)의 (b)와 같이 좌·우 양쪽의 대칭되는 항목 그룹을 한 쌍씩 차례로 묶어 각각을 최종 항목 그룹으로 통합한다. (그림 13)의 대칭 등간격 그룹핑은 아래 식 ⑧의 해싱 함수 $ch_{sig}(i)$로 얻을 수 있다.

$$ch_{sig}(i) = \begin{cases} i \bmod (|C_{kb}|\times 2) \\ \quad 단, (i \bmod (|C_{kb}|\times 2)) < |C_{kb}| \\ |C_{kb}|-(i \bmod |C_{kb}|)-1 \end{cases}$$

단, $(i \bmod (|C_{kb}|\times 2)) \geq |C_{kb}|$ ……식 ⑧

식 ⑧의 대칭 등간격 그룹핑 해싱 함수 $ch_{sig}(i)$를 적용하여 (그림 7)의 C_2를 재구성한 모습을 (그림 14)에 보였는데, 이 그림으로부터 해시 트리가 균형을 이루고 있음을 알 수 있다.

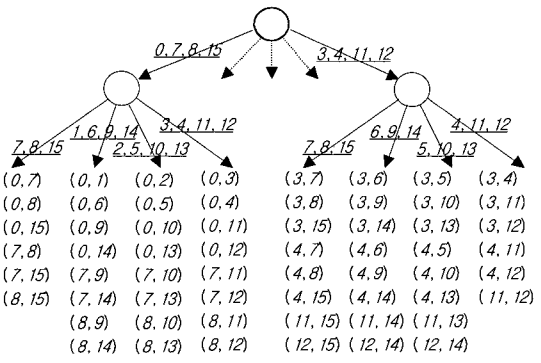
항목 그룹	0,8	1,9	2,10	3,11	4,12	5,13	6,14	7,15
버켓 번호	0	1	2	3	4	5	6	7

(a) $ch_{rig}(i) = i \bmod 8$에 의한 항목 그룹핑

항목 그룹	0,8, 7,15	1,9, 6,14	2,10, 5,13	3,11, 4,12
버켓 번호	0	1	2	3

(b) 좌·우 양쪽 항목 그룹을 하나의 그룹으로 통합

(그림 13) 대칭 등간격 그룹핑 개념



(그림 14) $ch_{sig}(i)$(대칭 등간격 그룹핑)에 의한 (그림 7)의 재구성

4. 제안된 H_2 및 C_k의 해싱 방법의 구현 및 성능 평가

여기서는 3 장에서 제안된 단계 2 직접 해시 테이블 H_2의 해싱 방법인 식 ⑥의 $dh_{mph}(i_1, i_2)$와 해시 트리 C_k의 항목 그룹핑 방법인 식 ⑧의 $ch_{sig}(i)$ 해싱 함수를 구현하여 적용한 *DHP* 알고리즘의 성능을 분석하고 검증한다.

4.1 해싱 함수 $dh_{mph}()$와 $ch_{sig}()$의 구현

식 ⑥의 $dh_{mph}(i_1, i_2)$에서 $\frac{(i_1+1)\times(i_1+2)}{2}$항은 항목 i_1에만 종속되어 있으므로 모든 항목에 대한 대응 값을 미리 계산하여 배열((그림 15)에서 mph_fctr[])에 저장한 다음 해싱 함수에서 참조하도록 한다.'식 ⑧의 $ch_{sig}(i)$는 수식 전체가 항목 i에만 종속 되지만 두 가지식으로 분리되어 있으므로 이 또한 미리 계산하여 배열((그림 15)에서 sig_group[])에 저장하여 사용하도록 한다.

<부록 1>에 $dh_{smh}(i_1, i_2)$, $dh_{mph}(i_1, i_2)$, $ch_{rig}(i)$, $ch_{sig}(i)$를 구현하는 C 코드를 보였다. 이들 중, $dh_{smh}(i_1, i_2)$와 $dh_{mph}(i_1, i_2)$는 (그림 3)의 line 6, 38에서, $ch_{rig}(i)$와 $ch_{sig}(i)$는 (그림 3)의 line 39에서 각각 사용된다.

4.2 시험 환경

• 성능 측정 시스템

성능 측정 시스템으로 <표 4> 사양의 알파 프로세서 기반의 유닉스 시스템을 사용하였다.

〈표 4〉 성능 측정 시스템의 환경

항 목	사 양
모 델	COMPAQ WS au600
C P U	Alpha Ev67 667MHz
메 모 리	1GB
운 영 체 제	Digital Tru64UNIX 4.0F

• 성능 측정 방법

연관 규칙 탐사 시험용 데이터 생성 프로그램[7]을 사용하여 <표 5>와 같이T5I2D100~T20I6D100 의 특성 각각에 대하여 전체 항목 개수 $|I|$가 N1000~N9000인 데이터(<표 4> 참조)를 생성하고, 이들 모두에 대하여 <표 6>의 네 가지 형태의 *DHP* 알고리즘을 각각 적용하여 1/10초 단위의 실행 시간(system mode+user mode)을 측정하였다.

〈표 5〉 시험 데이터의 유형

분 류 기 준	분 류 내 용
데이터 특성	T5I2D100, T10I2D100, T10I4D100, T15I4D100, T20I2D100, T20I4D100, T20I6D100
전체 항목 개수($\|I\|$)	N1000, N2000, N3000, N5000, N7000, N9000

〈표 6〉 변화된 *DHP* 알고리즘의 유형

알고리즘 유형(기호)		제안된 해싱 방법 적용 내용
①	$dh_{smh}()+ch_{rig}()$	H_2와 C_k 모두 일반적인 방법
②	$dh_{mph}()+ch_{rig}()$	H_2 에만 제안 방법 적용
③	$dh_{smh}()+ch_{sig}()$	C_k 에만 제안 방법 적용
④	$dh_{mph}()+ch_{sig}()$	H_2, C_k 모두 제안 방법 적용

4.3 $dh_{mph}(i_1, i_2)$와 $ch_{sig}(i)$의 성능 분석

(그림 15)에 전형적인 시험용 데이터 유형인 T15I4D100과 T20I6D100을 대상으로 <부록 1>과 같이 구현한 단계 2 직접 해시 테이블 H_2와 해시 트리 C_k를 위해 제안된 해싱 방법 $dh_{mph}(i_1, i_2)$와 $ch_{sig}(i)$ 각각을 다양한 조합으로 적용한 *DHP* 알고리즘을 사용하여 얻은 성능 측정 결과를 보였다.

4.3.1 $dh_{mph}(i_1, i_2)$와 $dh_{smh}(i_1, i_2)$의 비교

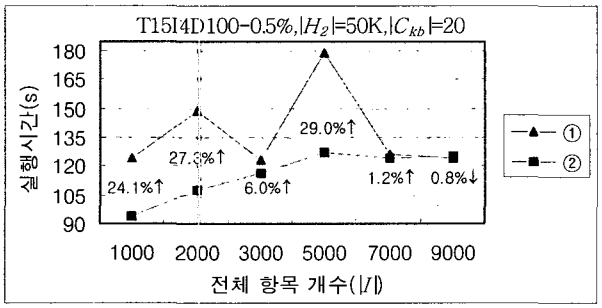
(그림 15)의 (a)에 H_2를 위한 해싱 방법의 성능비교 결과를 보였다. 이 그림으로부터 아래 사실들을 추론할 수 있다.

- 일반적인 방법 ①($dh_{smh}(i_1, i_2)$, 단순 mod 해싱)은 데이터의 구성 상태에 따라 성능의 편차가 매우 심하다.
- 제안된 방법 ②($dh_{mph}(i_1, i_2)$, 사상 완전 해싱)는 데이터의 구성 상태에 관계없이 늘 안정적인 성능을 발휘한다.
- 대부분의 경우 방법 ②가 방법 ①보다 우수하지만 데이터 구성이 특이한 경우 측정 오차 범위 내에서 방법 ①이 우수할 가능성이 있다.

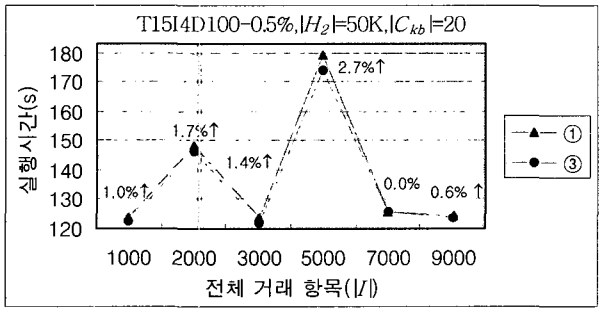
4.3.2 $ch_{sig}(i)$와 $ch_{rig}(i)$의 비교

(그림 15)의 (b)에 C_k를 위한 항목 그룹핑 방법의 성능 비교 결과를 보였다. 이 그림으로부터 아래 사실들을 추론할 수 있다.

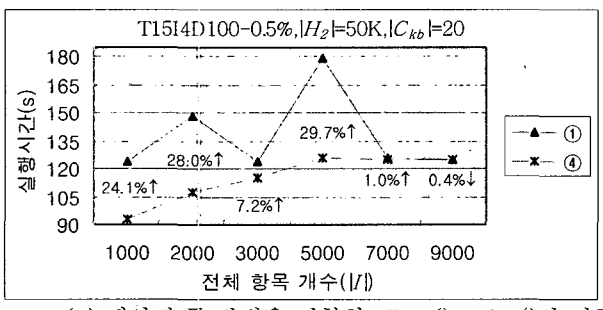
- 대부분의 경우 제안된 방법 ②($ch_{sig}(i)$, 대칭 등간격 그룹핑)가 일반적인 방법 ①($ch_{rig}(i)$, 단순 등간격 그룹핑)보다 크지는 않지만 안정적인 우수함을 보인다.



(a) 직접 해시 테이블의 dh_{mph}(사상 완전 해싱)와 dh_{smh}(단순 mod 해싱)의 성능 비교



(b) 해시 트리의 ch_{sig}(대칭 등간격 그룹핑)와 ch_{rig}(단순 등간격 그룹핑)의 성능 비교



(c) 제안된 두 방법을 결합한 $dh_{mph}()+ch_{sig}()$와 기존의 단순 방법을 결합한 $dh_{smh}()+ch_{rig}()$의 성능 비교

(그림 15) 단계 2직접 해시 테이블(H_2)과 해시 트리(C_k)를 위해 제안된 해싱 방법들의 성능 분석

• 데이터의 구성 상태에 따라 방법 ①이 측정 오차 범위 내에서 우수할 가능성이 있다.

4.3.3 $dh_{mph}(i_1, i_2)$와 $ch_{sig}(i)$를 결합한 성능 비교

(그림 15)의 (c)에 H_2 와 C_k를 위한 해싱 방법 들을 결합하여 적용한 방법 ④의 성능 비교 결과를 보였다. 이 그림의 전체적인 모습은 예측된 대로, 성능 지배력이 큰 H_2 해싱 방법론 비교의 경우인 (b)와 비슷하고, 거기에 C_k의 구룹핑 방법에 의한 성능 향상이 반영되어 있음을 볼 수 있다.

4.4 종합 성능 분석

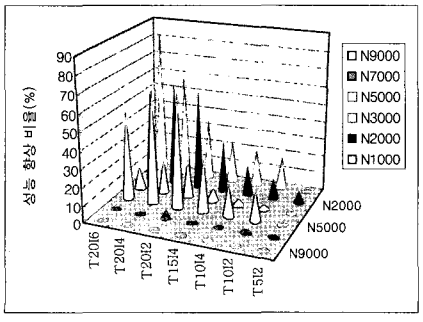
<표 7>에는 <표 4>에서 분류한 모든 유형의 시험 데이터에 대한 방법 ① 대비 방법 ④의 성능 향상 비율 수치를 나타냈고, (그림 16)에는 그 추세를 그림으로 보였다. 이 그림으로 아래 사실들을 추론할 수 있다.

• 제안된 해싱 방법들은 거의 모든 영역에서 성능 향상 효과가 있고, 특히 빈발 항목 집합을 구성하는 항목들이 조밀하고 빈발 항목 집합의 길이가 긴 경우(N1000-T20I6)에 효과가 크다.
• 빈발 항목 집합을 구성하는 항목들이 희박하고 빈발 항목 집합의 길이가 짧은 경우(N9000-T5I2)에는 성능 향상이 없거나 저하된다. 그러나 성능 저하가 있더라도 그 비율은 극히 미미하다.
• <표 4>의 모든 시험 데이터 유형에 대한 평균 성능 향상 비율은 18.5%이다.

<표 7> <표 4> 시험 데이터에 대한 방법 ④의 성능

DB \ |N|	1000	2000	3000	5000	7000	9000
T5I2	0.0	7.1 ▲	0.0	0.0	0.0 ▲	0.0
T10I2	18.2 ▲	11.4 ▲	2.6 ▲	15.9 ▲	2.1 ▲	1.0 ▽
T10I4	20.3 ▲	16.9 ▲	3.7 ▲	16.9 ▲	0.7 ▲	0.7 ▲
T15I4	24.1 ▲	28.0 ▲	7.2 ▲	29.7 ▲	1.0 ▲	0.4 ▽
T20I2	34.6 ▲	56.1 ▲	21.4 ▲	63.8 ▲	5.0 ▲	0.2 ▽
T20I4	57.6 ▲	58.2 ▲	18.5 ▲	64.3 ▲	1.7 ▽	3.2 ▽
T20I6	82.2 ▲	51.4 ▲	12.6 ▲	64.3 ▲	1.0 ▲	0.5 ▽

D100-0.5%, $|H_2|$=50K, $|C_{k0}|$=20, ▲:향상, ▽:저하, 단위:%



(그림 16) <표 4> 시험 데이터에 대한 방법 ④의 성능 추이

5. 결 론

연관 규칙 탐사 알고리즘은 주로, 후보 빈발 항목 집합의 수를 줄여서 전체적인 검색 및 계수를 위한 계산량을 줄이거나, DB에 대한 스캔 횟수를 감소시켜 IO 시간을 단축하는 등의 거시적 측면에서 발전되어 왔다. 이에 반하여 이 논문에서는 *DHP* 알고리즘을 대상으로 H_2와 C_k의 구체적인 구현 측면에서 효율적인 해싱 방법 $dh_{sig}(i_1, i_2)$와 $ch_{mph}(i)$을 제시하여 그 성능을 측정하고 평가하였다. 그 결과 일반적인 방법 대비 최대 82.2%, 평균 18.5% 성능 향상이라는 매우 의미 있는 결과를 얻었다. 극히 일부 데이터 유형에 대해서는 성능이 저하되는 경우가 있었으나 그 폭이 3.2% 이하로 미미하므로 전체적으로 안정적인 성능을 발휘하는 제안된 방법을 사용하지 않을 이유가 되지는 못할 것이다.

현재 효율적인 해싱 방법론에 관한 몇 가지 연구가 더 진행되고 있는데, 이 연구가 완료되면 *DHP* 알고리즘이 일부 적용 환경에서 탁월한 성능을 발휘할 것으로 기대되므로, 이와 더불어 *DHP*, *F.P tree* 등 다양한 알고리즘들이 적용 환경에 따라 어떤 성능 특성을 보이는지에 관한 연구를 진행할 예정이다.

참 고 문 헌

[1] 박종수, 유원경, 홍기영, "연관 규칙 탐사와 그 응용", 정보과학회지, 제16권, 제 9호, pp.37~44, 1998.

[2] 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, 1993.

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

[4] 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.

[5] 이재문, 박종수, "복합 해쉬 트리를 이용한 효율적인 연관 규칙 탐사 알고리즘", 정보과학회 논문지(B) 제 26권, 제 3호, pp.343~352, 1999.

[6] Jiawei Han, Jian Pei, and Yiwen Yin, "*Mining frequent patterns without candidate generation*", Proceedings of 2000 ACM SIGMOD Int. Conf. Management of Data(SIGMOD'00), Dallas, TX, pp.1-12

[7] R. Agrawal and et al, "*Synthetic Data Generation Code for Associations and Sequential Patterns*", http://www.almaden.ibm.com/cs/quest, 1999.

[8] A. Savasere, E. Omiecinski and S. Navathe, "*An Efficient Algorithm for Mining Association Rules in Large Databases*", Proceedings of the 21th VLDB Conference, pp.432~444, 1995.

[9] 이재문, "대용량 주기억장치 시스템에서 효율적인 연관 규칙

탐사 알고리즘", 정보처리학회 논문지D 제9-D권, 제4호, pp.579-586, 2002.

[10] 이형봉, "완전 해싱을 위한 *DHP* 연관 규칙 탐사 알고리즘의 개선 방안", 정보과학회 논문지 : 데이터베이스, 제31권, 제2호, pp.91~98, 2004.

[11] Zarka Cvetanovic, Darrel D. Donaldson, Jane, "*AlphaServer 4100 Performance Characterization*", Digital Technical Journal, Vol.8, No 4., pp.3-20, 1996, http://www.hpl.hp.com/hpjournal/dtj/vol8num4/vol8num4art1.pdf.

이 형 봉

e-mail : hblee@kangnung.ac.kr

1984년 서울대학교 계산통계학과(이학사)

1986년 서울대학교 대학원 계산통계학(전산과학)과(이학석사)

2002년 강원대학교 대학원 컴퓨터과학과(이학박사)

1986년~1994년 LG전자 컴퓨터 연구소 선임연구원

1994년~1999년 한국디지탈이큅먼트㈜ 책임컨설턴트

1997년~1999년 전자계산조직응용, 전자계산기, 정보통신기술사

1999년~2004년 호남대학교 정보통신공학부 조교수

2004년~현재 강릉대학교 컴퓨터공학과 조교수

관심분야 : 프로그램 언어 및 보안, 운영체제, 지식 탐사 알고리즘, 센서 네트워크 등

〈부록 1〉 $dh_{smh}(i_1, i_2)$, $dh_{mph}(i_1, i_2)$, $ch_{rig}(i)$, $ch_{sig}(i)$의 구현 코드

```
#define NITEM       7000  /* |I|, configurable */
#define N_H2BUCKET 50000 /* |H2|, tunable */
#define N_CKBUCKET 20    /* |Ckb|, tunable */
   /* simple mod hashing for direct hash table H2 */
#define DH_SMH(i1,i2) \
            (i1*NITEM +i2) % N_H2BUCKET
   /* mapped perfect hashing for direct hash table H2 */
#define DH_MPH(i1,i2) \
            (i1*NITEM +i2 -mph_fctr[i1]) % N_H2BUCKET
   /* regular interval grouping for hash tree Ck */
#define CH_RIG(i)  rig_group[i]
   /* symmetric interval grouping for hash tree Ck */
#define CH_SIG(i)  sig_group[i]
int  mph_fctr[NITEM],rig_group[NITEM],sig_group[NITEM];
void dh_mph_init()  /* initialization for DH_MPH() */
{
     int   i;
     for (i = 0; i < NITEM; i++)
        mph_fctr[i] = (i+1)*(i+2)/2;
}
void ch_rig_init() /* initialization for CH_RIG() */
{
     int   i;
     for (i = 0; i < NITEM; i++)
        rig_group[i] = i % N_CKBUCKET;
}
void ch_sig_init() /* initialization for CH_SIG() */
{
     int   i;
     for (i = 0; i < NITEM; i++) {
        if ((i % (N_CKBUCKET*2)) < N_CKBUCKET)
           sig_group[i] = i % (N_CKBUCKET*2);
        else
           sig_group[i] = N_CKBUCKET-(i%N_CKBUCKET)-1;
     }
}
```