한 번의 데이터베이스 탐색에 의한 빈발항목집합 탐색

채 덕 진[†] · 김 용[††] · 이 용 미[†††] · 황 부 현[††††] · 류 근 호[†††††]

요 약

본 논문에서는 한 번의 데이터베이스 스캔으로 빈발항목집합들을 생성할 수 있는 효율적인 알고리즘을 제안한다. 제안하는 알고리즘은 빈발항목과 그 빈발항목을 포함하고 있는 트랜잭션과의 관계를 나타내는 이분할 그래프(bipartite graph)를 생성한다. 그리고 생성된 이분할 그래프를 이용하여 후보 항목집합들을 생성하지 않고 빈발 항목집합들을 추출할 수 있다. 이분할 그래프는 빈발항목들을 추출하기위해 대용량의 트랜잭션 데이터베이스를 스캔할 때 생성된다. 이분할 그래프는 빈발항목들과 그들이 속한 트랜잭션들 간의 관계를 엣지(edge)로 연결한 그래프이다. 즉, 본 논문에서의 이분할 그래프는 대용량의 데이터베이스에서 쉽게 발견할 수 없는 빈발항목과 트랜잭션의 관계를 검색하기 쉽게 색인(index)화한 그래프이다. 본 논문에서 제안하는 방법은 한 번의 데이터베이스 스캔만을 수행하고 후보 항목집합들을 생성하지 않기 때문에 기존의 방법들보다 빠른 시간에 빈발 항목집합들을 찾을 수 있다.

키워드 : 데이터마이닝, 연관규칙, 이분할 그래프

Frequent Patterns Mining using only one-time Database Scan

Duck Jin Chai[†] · Long Jin[††] · Yongmi Lee[†††] · Buhyun Hwang[††††] · Keun Ho Ryu[†††††]

ABSTRACT

In this paper, we propose an efficient algorithm using only one-time database scan. The proposed algorithm creates the bipartite graph which indicates relationship of large items and transactions including the large items. And then we can find large itemsets using the bipartite graph. The bipartite graph is generated when database is scanned to find large items. We can't easily find transactions which include large items in the large database. In the bipartite graph, large items and transactions are linked each other. So, we can trace the transactions which include large items through the link information. Therefore the bipartite graph is a indexed database which indicates inclusion relationship of large items and transactions. We can fast find large itemsets because proposed method conducts only one-time database scan and scans indexed the bipartite graph. Also, it don't generate candidate itemsets.

Key Words : Data Mining, Association Rule, Bipartite Graph

1. 서 론

연관규칙 탐사는 데이터베이스를 이루는 트랜잭션들에서 사용자가 미리 정한 최소지지도(minimum support) 보다 많이 발생하는 빈발항목집합(large itemset or frequent itemset)들을 발견하는 문제로 정의할 수 있다. 지금까지 빈발항목집합들을 찾기 위한 다양한 알고리즘들이 제안되었다[1, 2, 3, 4, 5, 7, 8, 9, 10].

기존에 제안된 대부분의 알고리즘들은 Apriori 알고리즘[3]과 같은 휴리스틱(heuristic) 방법을 사용하였다. Apriori와 같은 휴리스틱 알고리즘들은 대용량의 데이터베이스를 반복적으로 스캔(scan)하는 것과 대단히 많은 후보항목집합(candidate itemset)들을 다루는 비용에 대한 문제를 가지고 있다. 따라서 기존의 알고리즘들은 이러한 많은 비용을 필요로 하는 두 가지 문제를 해결하기 위한 노력에 집중되었다. [5]에서는 후보항목집합들을 생성하지 않고 두 번의 데이터베이스 스캔으로 빈발항목집합들을 생성할 수 있는 FP-growth 알고리즘을 제안하였다. FP-growth 알고리즘은 빈발 패턴 트리(Frequent Pattern tree, FP-tree)구조를 이용하여 패턴들을 추출하게 된다. 후보항목집합 생성을 피함으로써 FP-growth 알고리즘은 많은 성능의 개선을 보였다. 그러나 두 번의 데이터베이스 스캔을 통해 FP-tree를 생성하고, 트리의 모든 노드를 검사하여 빈발항목집합들을 생성하는 기본적인 연산 비용 외에 모든 트랜잭션에 대해서 빈발하지 않는 항목들을 전지(pruning)하고 나머지 항목들을 정렬하는데 소요되는 많은 연산 비용이 필요하다.

본 논문에서는 한 번의 데이터베이스 스캔으로 빠르게 빈

※ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단(지방연구중심육성사업/충북BIT)과 BK의 지원으로 수행되었음.
† 준 회 원 : 충북대학교 BK21 PostDoc
†† 정 회 원 : 한국전자통신연구원 연구원
††† 준 회 원 : 충북대학교 전자계산학과 박사과정
†††† 종신회원 : 전남대학교 전산학과 교수
††††† 종신회원 : 충북대학교 전기전자 및 컴퓨터공학부 교수 (교신저자)
논문접수 : 2007년 3월 28일, 심사완료 : 2007년 9월 18일

DOI: 10.3745/KIPSTD.2008.15-D.1.15

발항목집합들을 발견할 수 있는 알고리즘을 제안한다. 제안하는 알고리즘은 이분할 그래프(bipartite graph) 구조를 이용한다. 본 논문에서 생성하는 이분할 그래프는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다. 본 논문에서는 지금부터 이 그래프와 제안하는 알고리즘을 각각 빈발항목 이분할 그래프(Large Items Bipartite graph, LIB-graph) 그리고 ALIB (Algorithm using LIB-graph) 알고리즘이라 명명한다.

제안하는 ALIB 알고리즘은 기존에 제안된 알고리즘과 비교하여 다음과 같은 특징을 갖는다. 첫째, 데이터베이스를 한 번만 스캔한다. ALIB 알고리즘은 처음 데이터베이스를 스캔하여 1-빈발항목들과 LIB-graph를 생성한다. LIB-graph는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다. 즉, 어떤 항목이 어떤 트랜잭션들에 포함되는지를 이 그래프를 통해 확인할 수 있다. 그러므로 대용량의 데이터베이스를 스캔하지 않고 LIB-graph를 통해 빈발항목집합들을 찾아낼 수 있다. 또한 우리의 고려 대상은 데이터베이스를 이루는 전체 항목이 아닌 오직 빈발항목이기 때문에 LIB-graph에서 1-빈발항목에 대한 엣지(edge) 만을 탐색하게 된다. 엣지를 통해 빈발항목이 포함된 트랜잭션들을 색인(index)할 수 있다. 그러므로 알고리즘의 수행이 빠르게 진행된다. 둘째, 후보항목집합들을 생성하지 않는다. 빈발항목집합들을 찾는 것은 어떤 항목집합에 대해서 그 항목집합을 포함하고 있는 트랜잭션의 개수가 미리 정의된 최소지지도를 만족하는지를 확인하는 것이다. LIB-graph에서 각 항목의 엣지를 통하여 어떤 항목집합이 전체 트랜잭션에 몇 번 포함되는지 알 수 있다. 또한, 어떤 항목집합이 같은 트랜잭션에 속해 있는지를 LIB-graph를 통하여 알 수 있다. 그러므로 후보항목집합들을 생성하지 않고도 빈발항목집합들을 찾을 수 있다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로써 연관 규칙 탐사에 대한 정의와 기존에 제안된 알고리즘들에 대해 설명한다. 3장에서는 ALIB 알고리즘의 수행 과정에 대해서 설명한다. 그리고 4장에서는 제안하는 알고리즘들과 기존에 제안된 알고리즘의 성능을 비교 분석한다. 5장에서는 결론 및 향후 연구에 대하여 논의한다.

2. 관련 연구

이 절에서는 연관규칙 탐사의 이해를 위해 알고리즘들 중에서 가장 기본적인 탐사 방법인 Apriori 알고리즘에 대해서 설명하고 후보항목집합을 생성하지 않고 빈발항목집합들을 찾아낼 수 있는 FP-growth 알고리즘에 대해 설명한다.

2.1 Apriori 알고리즘

연관 규칙 탐사 알고리즘의 가장 전형적인 방법인 Apriori 알고리즘[3]은 AprioriTid, AprioriHybrid[3]로 확장되어 연구되었다. Apriori 알고리즘은 많은 논문에서 인용되고 있으며 많은 응용에 적절히 변형되어 사용되고 있다. Apriori 알고리즘은 k-빈발항목집합 L_k를 구하기 위해서 (k-1)-빈발항목집합 L_{k-1}로부터 k-후보항목집합 C_k를 구하고 C_k의 지지도를 계산하여 최소 지지도 이상을 만족하는 L_k를 구하는 과정을 반복한다. Apriori 알고리즘의 각 단계의 진행은 데이터 항목의 증가에 따라 반복적으로 진행된다. Apriori 알고리즘은 더 이상의 후보항목집합을 생성할 수 없을 때까지 반복되어 빈발항목들을 탐사한다. 후보항목집합의 구성은 전 단계의 빈발항목집합의 조인(join) 연산과 전지 과정을 통해 생성된다. 조인 연산은 두 집합의 곱집합을 구하는 것과 같다. 전지 과정은 조인을 통해 생성된 후보항목집합의 부분 집합이 전 단계의 빈발항목집합의 원소가 아닌 경우, 그 항목은 삭제하는 과정이다. 그 이유는 전 단계에서 빈발하지 못하는 항목은 다음 단계에서도 빈발하지 못하기 때문이다. 전지 과정은 불필요한 후보항목의 수를 줄여 데이터베이스를 읽는 횟수를 감소시키기 위한 것이다.

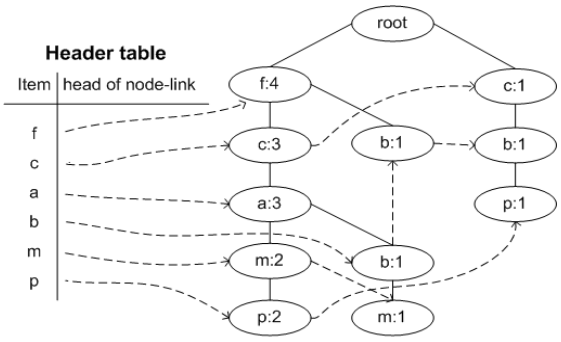
2.2 FP-growth 알고리즘

FP-growth 알고리즘[5]은 빈발 항목들에 대한 정보를 가지고 있는 FP-tree를 생성하고 최소지지도를 만족하는 빈발항목집합들을 생성한다. 또한, 데이터베이스의 반복되는 스캔을 피하고 후보항목집합들의 생성을 회피함으로써 빈발항목집합들을 구하는 비용을 줄였다.

FP-tree 생성은 먼저, 데이터베이스를 스캔하여 1-빈발항목들과 지지도를 구한다. 그리고 1-빈발항목들을 지지도에 따라 정렬하여 목록을 작성한다. 다음은 루트노드인 “null” 노드를 생성하고 데이터베이스의 각 트랜잭션을 스캔한다. FP-tree 생성은 각 트랜잭션에 대해서 1-빈발항목들의 목록에 속하지 않는 항목들을 전지하고 남은 항목인 빈발항목들은 다시 1-빈발항목들의 정렬방법과 같은 방법으로 정렬한다. 이렇게 정렬된 각 트랜잭션은 트리 생성에 이용된다. 이때, 각 트랜잭션의 항목들은 정렬된 상태이므로 먼저 발생한 항목이 부모 노드가 되고 다음 항목은 자식 노드가 되는 식으로 트리가 생성된다. <표 1>의 데이터베이스 예에 대한 FP-tree의 예는 (그림 1)에서 볼 수 있다. 생성된 FP-tree는 연관 규칙들을 생성하는데 이용된다. 생성된 FP-tree의 헤더 테이블(header table)에서 각 항목의 head of node-link에서부터 bottom-up 방식으로 링크를 따라 빈발항목집합들을 생성하게 된다. 생성된 FP-tree에서 각각의 경로는 하나의 트랜잭션을 이루는 항목들의 관계를 나타내고 있기 때문에 각 노드가 가지고 있는 지지도를 검색하여 쉽게 도출할

〈표 1〉 FP-growth 알고리즘에 대한 데이터베이스의 예[5]

TID	Items Bought	(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 1) FP-tree 생성의 예[5]

수 있다.

FP-growth 알고리즘은 두 번의 데이터베이스 스캔을 하지만 후보항목집합들을 생성하지 않고 빈발항목집합을 생성할 수 있는 효율적인 알고리즘이다. 그러나 FP-tree 생성 과정에서 크게 두 가지의 문제점이 있다. 첫 째, 하나의 트랜잭션을 스캔할 때 빈발하지 않는 항목들을 전지하는 연산과 전지되고 남은 빈발항목들을 정렬하는 두 가지의 연산이다. 이러한 연산은 단순히 임의의 항목들을 삽입하고 삭제하는 비교적 가벼운 연산이 아닌 비교 연산을 수행하기 때문에 상당한 시간적 비용을 수반하게 된다. 특히, 대용량 데이터베이스에서의 이러한 작업은 더욱더 큰 비용을 필요로 한다. 둘 째, 트리를 생성하기위해 트랜잭션을 스캔할 때마다 이전에 검색한 트리의 노드들을 계속적으로 반복 검색하며, 경우에 따라 공간적 비용이 커질 수 있다는 것이다. 하나의 트랜잭션을 스캔하여 트리를 갱신(update)할 때마다, 루트의 자식노드부터 비교해 가면서 같은 노드 항목이면 지지도 카운트를 하나 증가시키거나 노드 항목이 다를 때에는 새로운 노드를 생성하게 된다. 이는 트랜잭션을 스캔할 때마다 계속적으로 되풀이되는 과정이다. 또한, 공간상에서도 같은 노드 항목이 존재해도 조상이 틀리면 새로운 노드 항목으로 생성되어져야 하기 때문에 항목들의 분포에 따라 무수히 많은 노드들이 생성될 수 있다.

3. 그래프 구조를 이용한 빈발항목집합 추출

이 절에서는 한 번의 데이터베이스 스캔으로 빈발항목집합들을 찾아낼 수 있는 ALIB 알고리즘을 제안한다. 빈발항목집합을 찾는 문제는 찾고자하는 빈발항목집합이 어떤 트랜잭션 안에 존재하며 그 빈발항목집합을 포함하고 있는 트랜잭션들의 수가 사용자가 정한 최소지지도를 만족하는지를 검사하는 문제이다. 그러므로 데이터베이스를 이루는 각 항목이 어떤 트랜잭션에 존재하는지에 대한 정보를 가지고 있는 자료구조를 생성한다면 그 이후에는 데이터베이스를 스캔하지 않고 생성된 자료구조를 이용해서 빈발항목집합들을 찾아낼 수 있다.

제안하는 알고리즘은 데이터베이스에 존재하는 빈발항목과 그 빈발항목을 포함하고 있는 트랜잭션 정보를 쉽게 찾기 위해 LIB-graph라는 이분할 그래프 구조를 생성하고 이용한다. LIB-graph 는 데이터베이스를 이루는 항목들과 그들이 속한 트랜잭션들과의 관계를 그래프로 나타낸 것이다.

3.1 이분할 그래프 구조의 생성

이분할 그래프는 그래프 G=(V, E)에서 V(vertices)가 두 그룹 집합 X와 Y(=V-X)로 분할되어, E(edges)의 각 엣지가 X 내의 정점(vertex)과 Y 내의 정점의 쌍으로 나타내어지는 그래프이다.

본 논문에서 생성하는 LIB-graph는 이분할 그래프로서 두 개의 그룹 집합으로 이루어진다. 하나는 데이터베이스를 이루는 전체 항목이고 다른 하나는 전체 트랜잭션 식별자(Transaction IDentification, TID)이다. 두 그룹 사이에 엣지는 한 항목과 그 항목을 포함한 트랜잭션 사이에 관계를 나타낸다. 이 그래프 구조는 다음과 같은 특징을 갖는다. 첫째, LIB-graph의 정점은 두 개의 그룹 집합으로 이루어진다. 하나는 데이터베이스를 이루는 전체 항목들이고 다른 하나는 전체 트랜잭션 식별자들이다. 둘째, LIB-graph의 엣지는 두 그룹 사이의 포함 관계를 나타낸다. 즉, 한 항목과 그 항목을 포함하고 있는 트랜잭션 식별자 사이의 관계를 나타낸다. 셋째, 각 빈발 항목의 엣지의 수는 그 항목의 지지도를 나타낸다. 그래프 구조를 설명하기 위해서 [5]에서 이용되었던 <표 1>의 데이터베이스 예제를 사용하고 최소 지지도는 3으로 설정하였다. 그래프 구조 생성 방법은 다음과 같다.

데이터베이스를 스캔하여 빈발 항목들을 찾아낸다. 이때, 발생하는 빈발 항목들은 {(a, 3), (b, 3), (c, 4), (f, 4), (m, 3), (p, 3)}이다. 여기서의 숫자는 괄호안의 항목에 대한 지지도를 나타내고 표기는 편의상 알파벳 순서로 한다. 빈발 항목들을 발견하기 위해 데이터베이스를 스캔할 때 데이터베이스의 트랜잭션들이 차례대로 스캔된다. 이때, 하나의 트랜잭션을 스캔할 때마다 두 그룹 사이의 엣지가 형성된다.

그래프 생성 알고리즘에 의해서 첫 번째 트랜잭션 레코드를 스캔한다. 이때, 트랜잭션에 포함된 항목들은 {f, a, c, d, g, i, m, p}이다. 이들은 모두 트랜잭션 식별자가 100인 트랜잭션에 포함되어 있으므로 트랜잭션 식별자 그룹의 100이라는 정점으로 연결된다. 두 번째 트랜잭션에 포함된 항목들은 {a, b, c, f, l, m, o}이다. 이들은 같은 방법으로 모두 정점 200에 연결된다. 모든 트랜잭션이 스캔되면 (그림 2)와 같은 하나의 그래프 구조가 생성되고 동시에 빈발항목들이 추출된다. 그림 2에 표현된 항목들은 빈발항목들이다. 빈발항목집합 발견 과정에서 관심대상은 오직 빈발항목들이기 때문에 빈발하지 않는 항목과 그 항목의 엣지는 생략하였다.

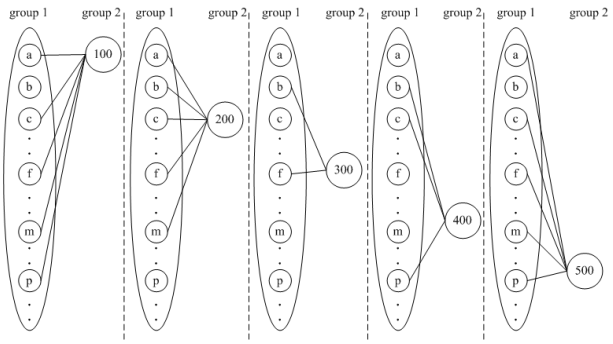
그래프 생성 알고리즘에서 알 수 있듯이, 정확히 데이터베이스를 한 번 스캔한다. 한 번의 데이터베이스 스캔으로 1-빈발항목들과 LIB-graph를 생성한다. 이 과정에서 별도의 연산을 필요로 하지 않는다. 그러므로 그래프 생성 알고리즘을 수행하는데 드는 비용은 O(|D|) 이다. 여기서 |D|는 데이터베이스를 이루는 전체 트랜잭션들의 수이다. 또한, 이 그래프는 빈발항목들과 이 빈발항목들이 포함된 트랜잭션들과의 관계를 표현하고 있기 때문에 이후의 빈발항목집합 추

출과정은 데이터베이스의 스캔 없이 그래프를 가지고 수행된다. 따라서 LIB-graph 는 데이터베이스안에 존재하는 빈발항목들과 트랜잭션들의 포함관계를 엣지를 이용하여 연결한 구조이다.

3.2 LIB-graph 를 이용한 빈발 항목집합 추출

LIB-graph에서 두 가지 중요한 특성을 발견할 수 있다. 첫 번째는 빈발 항목과 트랜잭션과의 포함관계이다. group 1은 빈발항목들의 집합이고 group 2는 Group 1에 속한 각 빈발항목을 포함하고 있는 트랜잭션 식별자들의 집합이다. Group 1에서 빈발 항목의 엣지는 그 항목을 포함하고 있는 트랜잭션 식별자에 링크되고 트랜잭션 식별자들의 엣지는 그 트랜잭션에 존재하는 항목들로 링크된다. 그러므로 어느 한 빈발항목의 엣지에 연결된 트랜잭션 식별자 엣지들을 추적하면 그 빈발항목을 포함하고 있는 모든 트랜잭션들과 그 트랜잭션에 포함된 다른 빈발항목들을 알 수 있다. 나머지 과정은 발견된 트랜잭션에서 빈발항목집합들을 찾아내는 것이다. 두 번째는 그래프의 group 1에 속하는 항목들이 정렬되어 있다는 것이다. group 1의 항목들에 대한 정렬은 지지도에 의한 것이 아니라 처음 그래프 구조를 설정할 때 미리 만들어지므로 별도의 연산 비용을 필요로 하지 않는다. 대용량의 트랜잭션 데이터베이스에서 빈발항목집합을 추출하고자 할 때, 이러한 정렬에 대한 비용은 알고리즘의 성능에 큰 영향을 준다. 따라서 본 논문에서 제안하는 알고리즘은 전체 빈발항목집합 탐사과정에서 큰 연산비용을 줄이게 된다.

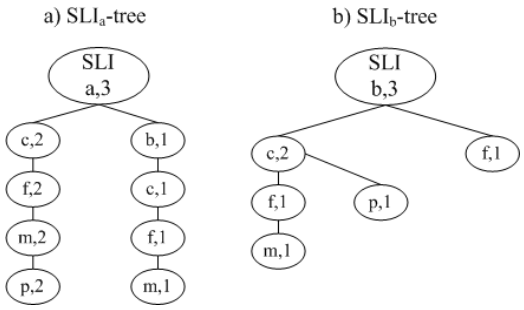
LIB-graph 탐사는 group 1에 포함된 빈발항목으로부터 시작된다. 정점 a에 대해서, 빈발 항목 a로부터 시작되는 엣지의 수는 3이다. 즉, 빈발 항목 a의 지지도는 3이라는 것을 나타내고 그 엣지로 연결된 트랜잭션은 {100, 200, 500}이다. 만약 a로 시작되는 빈발 항목집합이 존재한다면 a가 속한 트랜잭션마다 나머지 항목집합이 최소지지도 이상 존재해야만 된다. 그러므로 (그림 2)의 예에서 100, 200, 500의 트랜잭션에서 a와 함께 모두 같이 존재하는 항목들을 찾아내면 그 항목들은 a로 시작되는 빈발항목집합이 된다. (그림 2)에 보이는 것과 같이 트랜잭션 100에 링크되어 있는 엣지의 수는 a를 제외한 4개이다. 그림 상에서는 트랜잭션 100의 엣지의 수는 5개이지만 실제적으로는 더 많은 엣지를 갖는다. 그러나 우리의 관심 대상은 빈발항목들에 해당하므로 나머지 엣지들은 전지시킨다. 트랜잭션 100에 존재하는 엣지의 수는 5개이고 이들의 엣지를 따라가 보면 {a, c, f, m, p} 집합을 얻을 수 있다. 이 집합은 하나의 트랜잭션에 존재하는 모든 빈발항목들을 나타낸다. 여기의 예에서, 빈발항목 a를 포함하고 있는 여러 개의 트랜잭션 중에서 하나의 트랜잭션은 100이라는 식별자를 갖는 트랜잭션이고 그 트랜잭션에 포함된 모든 빈발항목은 a, c, f, m, p라는 것을 알 수 있다. 같은 방법으로, a가 포함된 두 번째 트랜잭션 200과 세 번째 트랜잭션 500의 엣지에서 각각 {a, b, c, f, m}과 {a, c, f, m, p}를 얻을 수 있다.



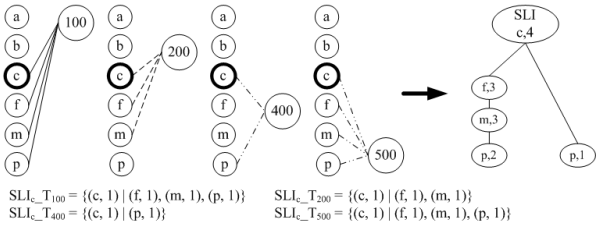
(그림 2) 예제 데이터에 대한 이분할 그래프 생성 예

추출된 빈발항목들의 집합으로부터 원하는 빈발항목집합들을 추출하기 위해 FP-tree를 응용한 트리 구조를 사용한다. [5]에서의 FP-tree는 전체 빈발항목들을 대상으로 하지만 본 논문에서의 트리는 하나의 빈발항목에 대해서 트리를 생성한다. 본 논문에서는 이 트리의 루트노드로 빈발항목을 갖는다는 의미에서 SLI_{item}-*tree(SLI, Start Large Item)*라 한다. 이때, SLI_{item}의 item은 하나의 빈발항목을 나타내고 이 항목은 트리의 루트 노드가 된다. SLI_{item}-*tree*는 위의 빈발항목 a에 대한 예와 같이 하나의 빈발항목에 대한 트랜잭션의 역 엣지를 스캔할 때 생성된다. 위의 빈발항목 a에 대한 SLI_a-*tree*의 생성은 처음, SLI_a-*tree*의 루트 노드를 생성한다. 트리의 각 노드는 항목과 그 항목의 지지도 값을 갖는다. SLI_a-*tree*의 루트 노드는 항목 a가 되고 지지도 값은 0으로 초기화된다. 빈발 항목 a와 첫 번째 엣지로 연결된 트랜잭션 100의 역 엣지를 스캔하면 트리의 첫 번째 가지(branch)가 생성된다. 그것은 루트 노드로부터 시작하는 <(a,1), (c,1), (f,1), (m,1), (p,1)>이다. 두 번째 트랜잭션 200에 대해서, 그것의 리스트는 {a, b, c, f, m}이다. 루트 노드 a를 제외한 나머지 리스트는 존재하는 패스(path)와 같지 않으므로 두 번째 가지가 생성된다. 이때, a의 지지도는 1 증가되고 생성된 가지, <(b,1), (c,1), (f,1), (m,1)>는 루트 노드 (a, 2)에 링크된다. 세 번째 트랜잭션 500에 포함된 빈발항목 리스트는 {a, c, f, m, p}이다. 이들은 첫 번째 가지와 동일한 패스이기에 그 패스를 따라서 각 노드의 지지도를 1씩 증가시키면 된다.

트리의 탐색은 루트노드로부터 생성된 자손들을 비교하면 쉽게 구할 수 있다. 이 트리는 두 개의 패스 <(a,2), (c,2), (f,2), (m,2), (p,2)>와 <(a,1), (b,1), (c,1), (f,1), (m,1)>을 생성한다. 첫 번째 패스는 전체 데이터베이스에서 두 번 나타난다는 것을 의미한다. a 항목은 데이터베이스에서 세 번 나타나지만 {c, f, m, p}와는 전체 데이터베이스에서 두 번 나타난다는 것을 의미한다. 두 번째 패스는 전체 데이터베이스에서 한 번 나타난다는 것을 의미한다. SLI_a-*tree*에서의 빈발항목집합 탐색은 전체 데이터베이스에서 빈발항목 a를 포함하는 모든 빈발항목집합들을 찾는 것이다. 두 개의 가지에서 최소지지도를 만족하는 최대 길이의 집합은 {a, c, f, m}이고 이 집합의 지지도는 3이다. 빈발항목집합의 부분집합은 역시 빈발항목집합이므로 빈발항목 a를 포함하고 있는 모든 빈발항목집합들은 {(ac), 3}, {(af), 3}, {(am), 3}, {(acf), 3}, {(acm), 3}, {(afm), 3}, {(acfm), 3}이 된다. (그림 3)은



(그림 3) *SLIa-tree*와 *SLIb-tree*의 예



(그림 4) 빈발항목 c에 대한 빈발항목집합 탐색의 예

빈발항목 a와 빈발항목 b에 대해서 LIB-graph를 탐사한 후의 생성된 트리를 보여준다. *SLI$_b$-tree*는 단말 노드로부터 세 개의 패스가 존재하지만 모두 최소지지도를 만족하지 못하기 때문에 그림에서 볼 수 있듯이 빈발항목 b로부터 생성될 수 있는 빈발항목집합은 존재하지 않는다.

정점 c에 대한 빈발 항목집합 추출 과정이 (그림 4)에 나타나있다. 빈발 항목 c는 모두 4개의 트랜잭션 {100, 200, 400, 500}에 존재한다. 그래프에서 빈발 항목 c를 포함하고 있는 트랜잭션을 스캔하여 *SLI$_c$-tree*를 생성한다. 그림 4에서 볼 수 있듯이, 각 트랜잭션에 빈발항목 a와 b에 대한 엣지가 존재하지만 이전에 이 두 개의 빈발항목에 대한 탐사를 완료했기 때문에 빈발항목 c에 대한 탐사에서는 빈발항목 a와 b에 대한 엣지에 대한 탐사를 할 필요가 없다. 그러므로 (그림 4)에서 보여 지듯이 빈발항목 c를 포함하고 빈발항목집합들은 {(cp), 3}, {(cf), 3}, {(cm), 3}, {(cfm), 3}이다. 이렇듯 빈발항목탐사 과정이 진행됨에 따라 연산해야할 항목의 수가 줄어들기 때문에 탐사과정의 후반으로 갈수록 연산은 더욱더 빠르게 진행된다. ALIB 알고리즘은 <표 2>에 기술되어진다.

4. 성능 평가

이 절에서는 본 논문에서 제안하는 ALIB 알고리즘에 대해서 최근에 제안된 FP-growth 알고리즘과의 성능을 비교한다. 성능 평가는 시간 복잡도 이론을 이용하여 두 알고리즘의 성능을 분석하고 시뮬레이션 실험을 통해 성능 측정을 수행한다.

4.1 시뮬레이션 실험 환경

모든 실험은 Microsoft Windows XP 운영체제, 1 GB 메인 메모리, 2GHz Pentium PC를 가지고 수행되었다. 프로그램은 Microsoft/Visual C++6.0을 사용하여 작성되었다. 실험에서 기록한 시간은 전체 실행 시간을 의미한다. 즉, 데이터를 입력 받을 때부터 최종 알고리즘 수행이 끝나고 결과를 출력할 때까지를 기록하였다. 그리고 실험에서 사용한 데이터는 기존의 알고리즘들에서 성능 평가를 위해 사용하였던 데이터 생성기를 사용하여 생성하였다.

실험은 두 개의 데이터 집합을 가지고 수행한다. 첫 번째 데이터 집합은 전체 항목들의 개수가 1000개인 T25.I10.D10K 이다. 이 데이터 집합은 D_1으로 표시된다. 이 데이터 집합은 평균 트랜잭션 크기가 25이고 잠재적으로 가능한 최대 빈발항목집합의 크기가 10이라는 것이다. D10K는 전체 트랜잭션의 개수가 10000개라는 것을 나타낸다. 두 번째 데이터 집합은 전체 항목들의 개수가 10000개이고 전체 트랜잭션의 개수가 100000개인 T25.I20.D100K 이다. 이 데이터 집합은 D_2로 표시된다.

〈표 2〉 ALIB(mining Algorithm using LIB-graph) 알고리즘

```
Input : transaction database and minimum support(ε)
Output : frequent patterns

Procedure ALIB(transaction database, ε) {
    for each transaction do
        scan transaction
    create LIB-graph and large items
    for each large item do
        for each large item do
            scan edges of large item
            create SLI_item-tree
        if child node of SLI_item-tree IS NOT NULL then
            create path from root node of SLI_item-tree
            create combination of SLI_item-large-branch
          for each combination do
                        if MIN(support of combination) >= ε then
                                store the combination and
                            MIN(support of combination)
}
```

4.2 시간 복잡도에 의한 성능 분석

시간 복잡도 이론을 이용하여 ALIB 알고리즘과 FP-growth 알고리즘의 성능 분석을 수행한다. 시간 복잡도에 의한 성능 분석은 크게 데이터베이스를 스캔하는 비용과 새로운 데이터 구조를 생성하고 생성된 데이터 구조를 검색하는 비용으로 나눠볼 수 있다. 시간 복잡도 분석을 위해 전체 데이터베이스를 이루는 트랜잭션의 개수를 D이라 하고 각 트랜잭션에 존재하는 항목들의 평균 개수를 M개, 그리고 빈발 항목의 개수를 L개라고 가정했을 때, 두 알고리즘의 시간 복잡도는 다음과 같다.

4.2.1 FP-growth 알고리즘의 시간 복잡도

FP-growth 알고리즘의 수행은 크게 네 가지 연산으로 나누어 볼 수 있다. 첫째, 1-빈발항목들을 발견하기 위해 데이터베이스를 스캔한다. 둘째, 트리를 생성하기 위해 미리 트랜잭션을 스캔하여 빈발하지 않는 항목들을 전지하고 나머지 빈발한 항목들을 지지도에 따라 정렬시킨다. 셋째, 정렬된 각각의 트랜잭션을 스캔하여 FP-tree를 생성시킨다.

넷째, 생성된 트리를 검색하여 빈발 항목집합들을 발견한다. 이 4가지 연산에 대한 시간 복잡도는 <표 3>과 같다.

4.2.2 ALIB 알고리즘의 시간 복잡도

ALIB 알고리즘의 수행은 크게 세 가지 연산으로 요약해 볼 수 있다. 첫째, 데이터베이스를 스캔하고 그래프를 생성하는 연산이다. 이 연산에는 데이터베이스를 스캔하는 연산과 1-빈발항목들을 발견하는 연산, 그리고 그래프를 생성하는 연산이 포함된다. 이들 각각의 연산이 독립적으로 실행될 시에는 각 연산에 대한 별도의 연산비용이 필요할 것이다. 그러나 1-빈발항목들을 발견하는 것과 그래프를 생성하는 것은 미리 생성된 자료 구조에 트랜잭션들을 스캔하여 읽어진 항목들을 링크 시키면 되기 때문에 별도의 연산 비용이 필요 없는 연산들이다. 그러므로 데이터베이스를 스캔하는 시간 복잡도만으로 그래프 생성 연산까지 해결이 가능하다. 둘째, 그래프를 검색하여 트리를 생성하는 연산이다. 셋째, 생성된 트리를 검색하여 한 항목에 대한 빈발 항목집합들을 발견하는 것이다. 이 세 가지 연산에 대한 시간 복잡도는 <표 4>와 같다.

<표 3>과 <표 4>를 비교했을 때, FP-growth 알고리즘과 ALIB 알고리즘은 생성하는 데이터 구조나 방법은 조금 다르지만 새로운 데이터 구조를 생성하고 그 데이터 구조를 이용하는 등의 비슷한 연산과정을 수행한다는 것을 알 수 있다. 그러나 ALIB 알고리즘은 최소한 FP-growth 알고리즘에서 수행하는 빈발하지 않는 항목들을 전지하고 난 후 나머지 빈발항목들을 정렬하는 연산을 수행하지 않는다. 그러므로 각 트랜잭션에서 빈발항목들을 지지도에 따라 정렬해야하는 연산 비용만큼 좋은 성능을 보인다는 것을 알 수 있다.

4.3 시뮬레이션 실험에 의한 성능 측정

FP-growth 알고리즘은 최근에 제안된 효과적인 알고리즘이다. FP-growth 알고리즘은 데이터베이스의 스캔횟수를 줄이기 위해 FP-tree라는 새로운 구조를 제안하였다. 데이터베이스를 최소한 스캔하여 FP-tree를 생성하고 이후의 빈발 항목집합 추출과정은 이 트리를 이용함으로써 데이터베이스 스캔을 최소화 하였다.

실험 결과에서 알 수 있듯이 FP-growth 알고리즘과 본 논문에서 제안하는 ALIB 알고리즘이 빈발 항목집합을 추출하는데 매우 효과적이다. (그림 5)와 (그림 6)은 각각 D_1과 D_2데이터집합을 가지고 수행한 성능 평가의 결과를 보여준다. 그림에서 볼 수 있듯이 FP-growth 알고리즘과 ALIB 알고리즘이 좋은 성능의 알고리즘이라는 것을 보여준다. 그러나 제안하는 ALIB 알고리즘의 성능이 더 좋다는 것을 그림에서 볼 수 있다.

두 알고리즘의 성능상의 차이는 크게 데이터베이스 스캔횟수와 새로운 구조를 생성하는데 드는 연산 비용이 FP-growth 알고리즘보다 ALIB 알고리즘이 더 적기 때문일 것이다. FP-growth 알고리즘 수행에서 주로 차지하는 컴퓨팅 비용은 데이터베이스를 스캔하고 빈발 항목들의 지지도에 따라 트랜잭션을 정렬하는 것과 정렬된 트랜잭션을 다시 스캔하여 FP-tree를 생성해 나가는 것이다. FP-growth 알고리즘이 후보 항목집합을 생성하지 않고 데이터베이스를 적게 스캔하지만 대용

〈표 3〉 FP-growth 알고리즘 시간 복잡도

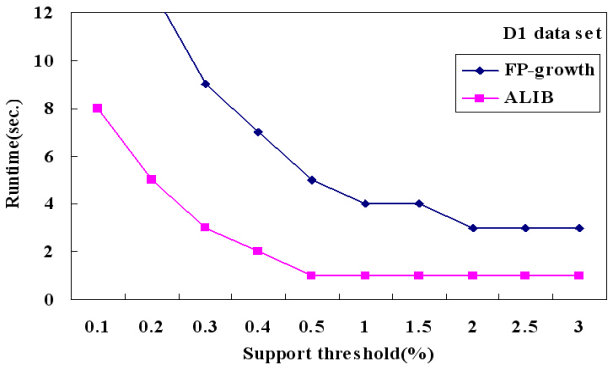
트랜잭션의 개수 : D, 각 트랜잭션의 평균 항목 수 : M, 빈발항목의 수 : N

main operation	sub operation	description	time complexity
데이터베이스 스캔	database scan	빈발항목 발견 연산을 위한 데이터베이스 스캔	O(D × M)
항목 전지 및 정렬	database scan	전지와 정렬을 위한 데이터베이스 스캔	O(D × M)
	non-frequent item pruning	빈발하지 않는 항목 전지 연산	O(D × M)
	frequent item sorting	전지되고 남은 빈발항목 정렬 연산	O(D× LlogL)
트리 생성	FP-tree generation	FP-tree 생성 연산	O(D × M)
빈발항목집합 추출	FP-tree scan	빈발항목집합을 발견하기 위한 FP-tree 스캔	O(L × logL)
	frequent pattern mining	추출된 빈발 시퀀스로부터 빈발항목집합 추출	O(L2)

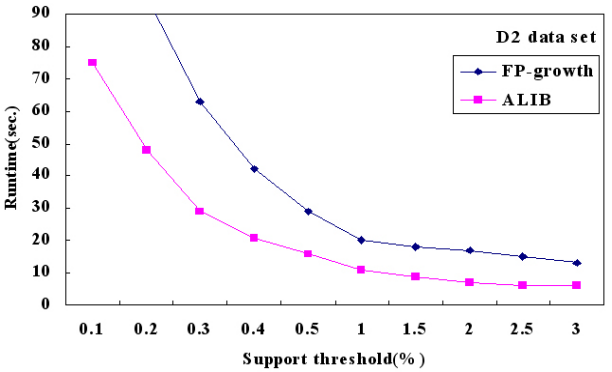
〈표 4〉 ALIB 알고리즘 시간 복잡도

트랜잭션의 개수 : D, 각 트랜잭션의 평균 항목 수 : M, 빈발항목의 수 : N

main operation	sub operation	description	time complexity
데이터베이스 스캔	database scan	빈발항목 발견과 그래프 생성을 위한 데이터베이스 스캔	O(D × M)
트리 생성	graph scan	트리 생성을 위한 그래프 스캔	O(L × D)
	non-frequent edge pruning	빈발하지 않는 항목에 대한 엣지들의 전지 연산	O(D × M)
	tree generation	SLI-tree의 생성	O(L × M)
빈발항목집합 추출	SLI-tree scan	빈발항목집합을 발견하기 위한 SLI-tree 스캔	O(L × logL)
	frequent pattern mining	추출된 빈발 시퀀스로부터 빈발항목집합 추출	O(L2)



(그림 5) FP-growth와 ALIB의 성능 비교



(그림 6) FP-growth 와 ALIB의 성능 비교

량의 데이터베이스에서 빈발하지 않는 항목들을 전지하고 빈발한 항목들을 정렬하는데 드는 비용은 여전히 많은 비용을 차지한다.

5. 결론 및 향후 연구 방향

임의의 k-빈발항목집합은 하나의 트랜잭션에 존재하며 빈발항목의 개수가 k 개인 항목들의 집합을 의미한다. 우리는 빈발항목집합을 생성하기 위해 데이터베이스를 이루는 모든 트랜잭션들을 검사한다. 빈발항목집합들을 발견하는 문제는 데이터베이스에서 어떤 항목집합이 최소지지도 이상으로 트랜잭션에 존재하는가를 찾는 문제라 정의할 수 있다.

빈발항목집합들을 발견하는 과정에서 데이터베이스를 스캔하는 것은 필수적이다. 그러나 대용량의 데이터베이스의 스캔 횟수가 많으면 많을수록 빈발항목집합을 찾는 연산 비용은 증가하게 된다. 그러므로 기존의 알고리즘들은 대용량의 데이터베이스를 스캔하는 횟수를 줄이는데 초점을 맞추고 있다.

데이터베이스의 스캔횟수를 줄이고 후보항목집합을 생성하지 않음으로써 많은 성능 개선을 보인 대표적인 알고리즘이 FP-growth 알고리즘이다. 이 알고리즘은 데이터베이스를 두 번 스캔한다. 그리고 FP-tree라는 데이터베이스를 압축한 형태의 자료구조를 생성한다. 이후의 빈발항목집합 추출과정은 이 FP-tree를 가지고 수행함으로써 알고리즘 수행이 빠르다. 그러나 FP-tree를 생성하기 위해서 데이터베이스의 각 트랜잭션을 조작하는데 많은 컴퓨팅 비용을 요구한다.

본 논문에서는 대용량 데이터베이스를 스캔횟수를 줄이고, 빈발항목집합을 빠르게 생성하기 위한 ALIB 알고리즘을 제안하였다. ALIB 알고리즘은 데이터베이스를 단지 한 번만 스캔함으로써 원하는 빈발항목집합을 찾아낼 수 있다. 이러한 연산은 데이터베이스를 한 번 스캔하여 생성된 LIB-graph를 가지고 가능해진다. LIB-graph는 빈발항목과 트랜잭션과의 관계를 알 수 있는 그래프 구조이다. 데이터베이스에서 우리가 찾고자 하는 것은 빈발항목집합이다. 이 빈발항목집합은 빈발항목으로 이루어져있다. 데이터베이스의 각 트랜잭션을 스캔하고 그 트랜잭션을 이루는 항목들이 빈발한지를 확인하기 위한 연산에 많은 비용을 소비한다. ALIB 알고리즘은 빈발항목에 대한 정보를 가지고 있는 LIB-graph를 이용함으로써 빈발하지 않는 연산에 불필요한 항목들에 대한 연산 비용을 줄일 수 있다. 또한, ALIB 알고리즘은 후보항목집합을 생성하지 않는다. 기존의 알고리즘에서 빈발항목집합 탐사과정의 가장 많은 연산 비용을 차지했던 부분이 후보항목집합의 생성과 그들이 빈발한지 안하는지를 확인하는 작업이었다. ALIB 알고리즘은 후보항목집합을 생성하지 않으므로 빈발항목집합을 찾아내기 위한 연산 비용을 크게 줄일 수가 있었다. 시뮬레이션 실험을 통하여 제안한 알고리즘이 FP-growth 알고리즘 보다 우수한 성능을 갖는다는 것을 보였다.

ALIB 알고리즘이 시간상에서는 우수하다고 할 수 있으나 FP-growth보다 공간상의 문제를 가질 수 있다. FP-growth 알고리즘에서 생성하는 FP-tree의 최대 깊이는 M이고 자식 노드의 최대 개수는 L이므로 FP-growth 알고리즘의 공간복잡도는 $O(LM^2)$이다. ALIB 알고리즘에서, LIB-graph를 생성하는 것의 공간복잡도는 전체 트랜잭션의 개수 D와 빈발항목의 개수 L에 대한 O(D+L)이다. 그리고 하나의 SLI-tree를 생성하는데 공간복잡도는 트랜잭션을 이루는 항목들의 평균 개수가 M개 이므로 $O(M^2)$이다. 하나의 SLI-tree는 휘발성으로, 빈발항목집합을 찾는 일이 종료되면 메모리에서 사라지기 때문에 SLI-tree에 대한 전체 공간복잡도는 $O(M^2)$이다. 따라서, D의 개수와 L의 개수가 커지면 커질수록 ALIB 알고리즘이 FP-growth보다 효율적이지 못할 수 있다. 향후엔 더 빠른 성능의 알고리즘에 대한 연구와 메모리의 효율성을 높일 수 있도록 연구가 진행되어야 할 것이다.

참 고 문 헌

[1] R. Agrawal, C. Aggarwal and V. V. V. Prasad, “A tree projection algorithm for generation of frequent itemsets,” In Journal of Parallel and Distributed Computing, Volume 61, Issue 3, pp.350-371, March, 2001.

[2] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases,” In Proceedings of the ACM SIGMOD, Washington D.C., pp.207-216, May, 1993.

[3] R. Agrawal, R. Srikant, “Fast Algorithms for Mining Association Rules in Large Databases,” In Proceedings of the VLDB, Santiago, Chile, pp.487-499, September, 1994.

[4] G. Grahne, L. Lakshmanan and X. Wang, “Efficient mining of constrained correlated sets,” In Proceedings of the ICDE, pp.512-521, February, 2000.
[5] J. Han, J. Pei and Y. Yin, “Mining frequent patterns without candidate generation,” In Proceedings of the ACM SIGMOD, pp.1-12, June, 2000.
[6] M. Klemettinen, h. Mannila, P. Ronkainen, h. Toivonen and A.I. Verkamo, “Finding interesting rules from large sets of discovered association rules,” In Proceedings of the CIKM, pp.401-408, November, 1994.
[7] B. Lent, A. Swami and J. Widom, “Clustering association rules,” In Proceedings of the ICDE, pp.220-231, April, 1997.
[8] B. Liu, W. Hsu and Y. Ma, “Mining association rules with multiple minimum supports,” In Proceedings of the ACM SIGKDD, pp.337-341, August, 1999.
[9] R. Ng, L. V. S. Lakshmanan, J. Han and A. Pang, “Exploratory mining and pruning optimizations of constrained associations rules,” In Proceedings of the ACM SIGMOD, pp.13-24, June, 1998.
[10] J.S. Park, M.-S. Chen, and P.S. Yu, “An Effective Hash-Based Algorithm for Mining Association Rules,” In Proceedings of the ACM SIGMOD, pp.175-186, May, 1995.
[11] S. Sarawagi, S. Thomas and R. Agrawal, “Integrating association rule mining with relational database systems: Alternatives and implications,” In Proceedings of the ACM SIGMOD, pp. 343-354, June, 1998.
[12] A. Savasere, E. Omiecinski and S. Navathe, “An efficient algorithm for mining association rules in large databases,” In Proceedings of the VLDB, pp.432-444, September, 1995.
[13] R. Srikant, Q. Vu and R. Agrawal, “Mining association rules with item constraints,” In Proceedings of the Knowledge Discovery and Data Mining, pp.67-73, August, 1997.

채 덕 진

e-mail : djchai2520@hotmail.com
1999년 동신대학교 컴퓨터학과(학사)
2001년 전남대학교 대학원 전산통계학과(석사)
2006년 전남대학교 대학원 전산학과(박사)
2006년~현 재 충북대학교 BK21 PostDoc
관심분야 : 데이터 마이닝, 스트림 데이터 마이닝, 멀티미디어 데이터베이스

김 룡

e-mail : kimlyong@dblab.chungbuk.ac.kr
2000년 연변과학기술대학교 전자전산학과(학사)
2003년 충북대학교 대학원 전자계산학전공(석사)
2007년 충북대학교 대학원 전자계산학전공(박사)
2007년~현 재 한국전자통신연구원 연구원
관심분야 : 스트림 데이터 마이닝, 시공간 데이터 마이닝, 시공간 데이터 베이스, 스트림 데이터 처리, 센서 데이터 처리

이 용 미

e-mail : ymlee@dblab.chungbuk.ac.kr
2002년 충북대학교 컴퓨터과학과(학사)
2005년 충북대학교 전자계산학과(석사)
2005~현 재 충북대학교 전자계산학과 박사과정
관심분야 : 시공간 데이터베이스, 스트림 질의 처리, 스트림 데이터 마이닝

황 부 현

e-mail : bhhwang@chonnam.chonnam.ac.kr
1978년 숭실대학교 전산학과(학사)
1980년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1980~현 재 전남대학교 전산학과 교수
관심분야 : 분산시스템, 분산 데이터베이스 보안, 객체지향 시스템, 전자상거래, 스트림 데이터 마이닝

류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr
1976년 숭실대학교 전산학과(이학사)
1980년 연세대학교 산업대학원 전산전공 (공학석사)
1988년 연세대학교 대학원 전산전공 (공학박사)
1976~1986년 육군군수 지원사 전산실(ROTC 장교), 한국전자통신연구원(연구원), 한국방송통신대학교 전산학과(조교수) 근무
1989년~1991년 Univ. of Arizona Research Staff (TempIS 연구원, Temporal DB)
1986년~현 재 충북대학교 전기전자 및 컴퓨터공학부 교수
관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 유비쿼터스 컴퓨팅 및 스트림 데이터 처리, 지식기반 정보검색 시스템, 데이터베이스 보안, 데이터 마이닝, 바이오인포메틱스