석 사 학 위 논 문

# 개인 맞춤형 웹서비스를 위한 효율적 연관규칙 탐색 알고리즘에 대한 연구

2015년



한성대학교 대학원
멀 티 미 디 어 공 학 과
멀 티 미 디 어 공 학 전 공
최 지 훈

석 사 학 위 논 문
지도교수 엄종석

# 개인 맞춤형 웹서비스를 위한 효율적 연관규칙 탐색 알고리즘에 대한 연구

A Study on Efficient Association Rule Mining Algorithm for Personalized Web Service

2014년 12월 일

한성대학교 대학원
멀 티 미 디 어 공 학 과
멀 티 미 디 어 공 학 전 공
최 지 훈

석 사 학 위 논 문
지도교수 엄종석

# 개인 맞춤형 웹서비스를 위한 효율적 연관규칙 탐색 알고리즘에 대한 연구

A Study on Efficient Association Rule Mining Algorithm for Personalized Web Service

위 논문을 공학 석사학위 논문으로 제출함

2014년 12월 일


한성대학교 대학원
멀 티 미 디 어 공 학 과
멀 티 미 디 어 공 학 전 공
최 지 훈

최지훈의 공학 석사학위논문을 인준함

2014년 12월 일

심사위원장 _____ 인

심 사 위 원 _____인

심 사 위 원 _____인

# 국 문 초 록

## 개인 맞춤형 웹서비스를 위한 효율적 연관규칙 탐색 알고리즘에 대한 연구

한성대학교 대학원
멀티미디어공학과
멀티미디어공학전공
최 지 훈

오늘날의 수 많은 홈페이지 사이트들은 다양한 컨텐츠로 인하여 사용자가 필요한 컨텐츠를 선택하기 어렵다. 또한 접속 기록은 매우 방대하여 사용자 개개인의 웹 접속 패턴을 찾는데 입출력과 복잡한 연산으로 인해 많은 시간과 하드웨어 자원이 요구된다. 여기서는 시간과 하드웨어 자원의 요구를 최소화 시키는 연관분석기법을 사용하여 방대한 웹 접속 기록을 분석하였다. 이 방법을 본 대학교 홈페이지 웹 접속 로그 분석에 적용하여 3 혹은 4 개의 빈발 발생 항목을 찾고, 또한 순차적으로 빈발하게 접근하는 항목을 찾았다. 이를 이용하여 웹 페이지 디자인에 반영하여 자주 사용되는 항목을 그룹화 하여 제공함으로 이용에 편리를 기할 수 있으며, 또한 사용자 개인에 대한 개인 맞춤형 웹서비스를 제공 할 수 있다.

【주요어】 연관규칙, 빈발항목, 개인 맞춤형 웹서비스, Apriori 알고리즘, 웹로그

# 목 차

# 표 목 차

# 그 림 목 차

# I. 서 론

## 1.1 연구 배경

데이터 마이닝(Data mining)은 데이터베이스(Database), 데이터웨어하우스(Data warehouse) 또는 여러 데이터 저장소의 방대한 데이터 속에서 유도된 새로운 데이터 관계들을 분석하여 정보를 추출하고 활용하는 과정을 말한다. 다른 말로는 KDD (데이터베이스 속의 지식 발견, Knowledge-discovery in database) 라고도 일컫는다(이재규 등, 2005). 데이터 마이닝의 어원은 광산에 매장된 자원을 채굴하는 것과 방대한 데이터 속에서 유용한 정보를 추출하는 것 사이의 유사점에서 유래되었다. 데이터 마이닝이라는 용어가 등장한 것은 20년 정도밖에 되지 않았고, 컴퓨터 공학과 인공지능 연구에 사용되는 기법이 통계학과 접목되면서 데이터 마이닝이 탄생하게 되었다(이지은 2002). 1943년 신경학자 워렌 맥컬리(W. McCulloch)와 논리학자 월터 피츠(W. Pitts)의 생체신경을 연구하며 인간 두뇌와 신경 세포간의 작용을 파악하기 위해 간단한 모델을 개발하였다. 이것이 데이터 마이닝의 시초라고 할 수 있으며 이 후 관계형 데이터베이스의 등장과 기업들의 방대한 데이터의 축척으로 인해 다양한 분야에 데이터 마이닝이 활용되었다. 데이터 마이닝은 전산화, 정보화가 진행되면서 인터넷의 보급이 확산되고 사회 각지에서 방대한 데이터의 양산과 함께 발전되었다. 2003년 당시 유통업체 월마트는 10테라바이트(terabyte) 크기의 데이터베이스에 매일 2천만건의 거래데이터를 저장하였다. 반면 1950년 큰 규모의 회사들은 전자문서 형태로 수십 메가바이트(megabyte) 정도의 데이터를 갖고 있었다. 반세기 동안 몇 십만에서 몇 백만배 까지 데이터 량이 증가하였다. Lyman and Varian (2003)은 1999년 생산된 정보의 두 배에 해당하는 정보가 2002년에 생산되었고 이 중 40%의 정보는 미국에서 생산되었다고 추정하였다(Galit Shmueli 등, 2009). 현재까지도 데이터의 양은 기하급수적으로 증가 하여 전세계가 현재 보유하고 있는 데이터의

90%는 지난 2년간 생성된 것이며, 매일 250경 바이트에 달하는 데이터가 새롭게 생성되고 있다고 한다. 향후 데이터의 증가량은 더욱 많아져 10년간 44배 이상 증가할 것으로 예측된다(함유근 등, 2012). 표1은 사회 각지의 데이터의 현황에 대해 보여준다(한영상 등, 2012).

| 데이터 소스 | 현황 |
|---|---|
| RF-ID Tag | 2005 년 13 억 개에서 2010 년 300 억 개에 달함 |
| Smart Phone | 전세계에 46 억대의 Smart Phone |
| Internet Users | 2011 부터 2013 년까지 전세계 20 억 인터넷 사용자의 연간 트랜잭션 수는 667 엑시바이트 수준 |
| Google | 구글은 하루에 24 페타바이트 이상의 데이터 처리 |
| Facebook | 페이스북은 매일 10 테라바이트의 데이터 처리 |
| Twitter | 트위터는 매일 25 억 트윗, 7 테라바이트의 데이터 처리 |
| CERN | CERN 의 하드론 입자가속기는 초당 40 테라바이트의 데이터 생성 |
| NYSE | 뉴욕증권 거래소는 매일 1 테라바이트의 거래 정보 보관 |

**표 1:** 사회 각지의 데이터 현황

데이터마이닝은 데이터간의 숨겨진 패턴이나 관계를 발견하여 사용자의 활동을 미리 예측하거나, 상품간의 관계를 규명 또는 특정한 부류를 분류하는 등 많은 분야에서 사용된다. 예를 들어 대형마트에서 특정시간에는 어떤 상품들이 잘 팔리고, 우유를 구입하는 사람은 빵을 구입할 확률과 같은 제품들 간의 관계를 규명하여 프로모션을 진행하거나 물건의 배치를 변경하는 등 판매효과를 높일 수 있다. 이외에도 데이터 마이닝의 응용분야로는 유전자 패턴분석, 신용평점시스템의 신용평가모형 개발, 쇼핑몰의 사용자 분류 및 장바구니 분석, 제품의 품질개선 등 많은 분야에서 사용된다. 그림 1에서는 각 데이터 마이닝 기법이 활용될 수 있는 분야를 도식화 하였고, 그림 2에서는 데이터 마이닝의 처리과정에 대해 도식화 하였다. 데이터 마이닝은 방대한 데이터에 의존하여 상황을 분석하고 적용하기 때문에

자료가 분석하기에 충분하지 않거나 현실을 충분히 반영하지 못한 상태에서 결론을 도출할 경우 잘못된 정보를 전달할 수 있는 오류를 범할 수 있다.

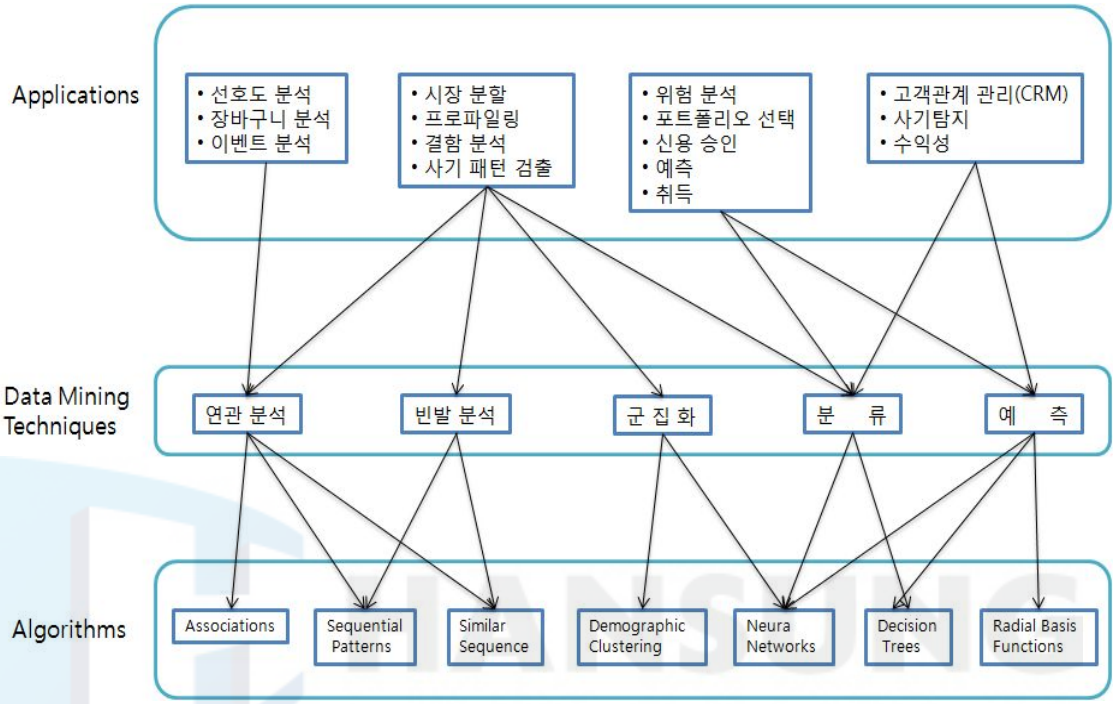



**그림 1:** 데이터 마이닝과 적용분야

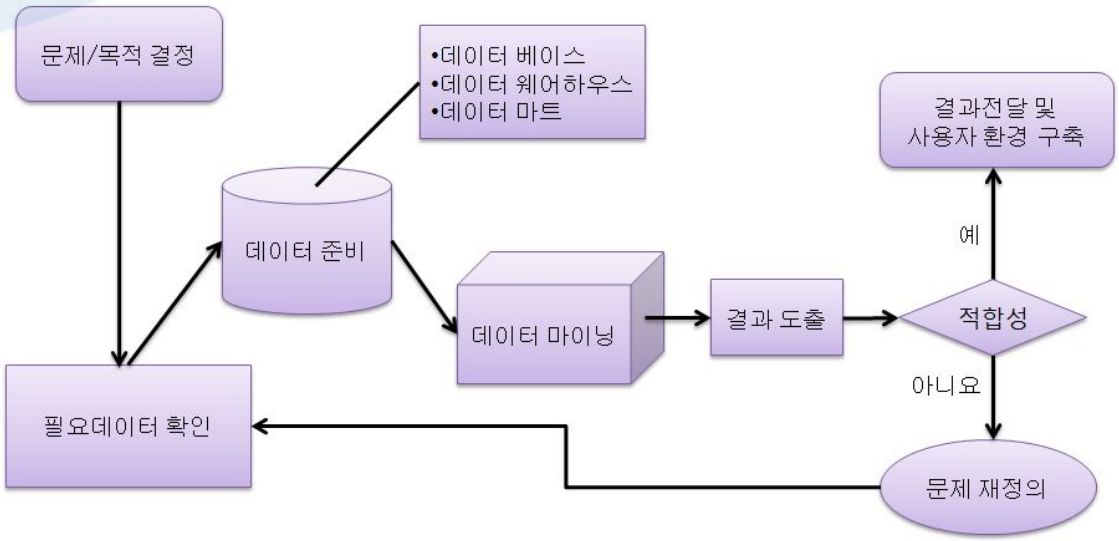



**그림 2:** 데이터 마이닝 처리과정

## 1.2 연구의 필요성

오프라인상의 마트, 백화점, 도서점 등을 이용하는 수 많은 활동들을 인터넷이 발전함에 따라 크고 새로운 활동무대가 만들어져 인터넷상에서도 할 수 있게 되면서 온라인과 오프라인의 경계가 모호해졌다. 온라인 공간이 점점 커지며 온라인상의 활동들을 지원하는 컨텐츠들은 방대하게 쏟아져 나오고 있다. 다양한 컨텐츠로 인해 사용자가 원하는 컨텐츠를 정확하게 선택하기 어려워지는 경우가 생겼다. 따라서 그림 3과 같이 사용자의 컨텐츠 이용 성향을 분석하여 맞춤형 서비스를 제공하며 사용자의 컨텐츠 선택의 폭을 줄여주여 선택에 도움을 주는 기술의 필요성이 제기되었다.

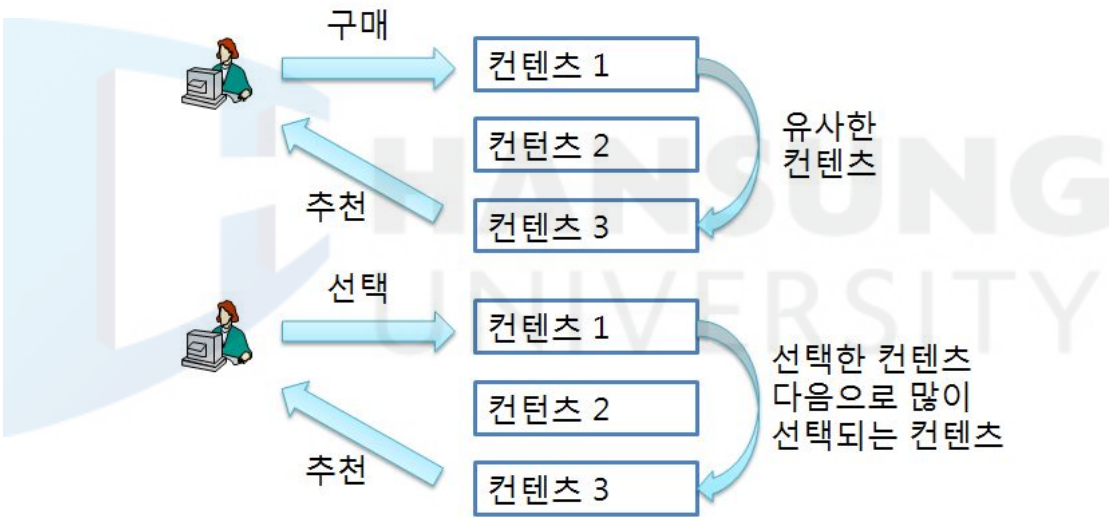



**그림 3:** 컨텐츠 이용 성향 분석

데이터 마이닝의 분야 중에서 연관규칙(association rule) 분석은 대표적인 기술로써 한 항목집합의 존재와 다른 항목집합과의 연관관계를 정의하는데 사용된다. 연관규칙의 적용 분야는 마케팅, 전자상거래, 생물학과 월드와이드웹(WWW)등과 같이 넓고 다양하기 때문에 많은 양의 데이터를 이용한 연관규칙 마이닝(association rule mining)은 많은 연구원에 의해 연구되어 왔다. 월드와이드웹이 광범위하게 쓰이면서 오늘날 많은 양의 데이터가 축척되었고, 웹상에서 사용 가능하게 되었다. 웹 마이닝(web mining)을 통해 로그

파일에 있는 사용자들의 활동에 대해 분석할 수 있고, 그 사용자들의 활동을 통해 유용한 정보를 발굴할 수 있다. 다시 말하면 웹 마이닝은 웹상의 데이터에서 사용자들의 숨겨진 정보를 발굴하고 추출한다는 것이다. 여기서 사용자들의 활동이란 사용자들이 웹페이지를 탐색하는 동안 선택하는 웹페이지 항목(item)들이라고 할 수 있다. 각 사용자들의 활동은 사용자의 IP주소에 따라 로그파일에 기록 되고, 이를 트랜잭션(transaction)이라고 부른다. 연관규칙 마이닝은 장바구니 분석이라고도 알려져 있고 1993년에 아그라왈(Agrawal 등, 1993) 에 의해 처음 소개되었다. 연관규칙 마이닝은 빈발항목집합(frequent itemset)을 찾는 것과 직접적인 연관이 있다. 빈발항목 집합을 찾는 것은 빈발항목집합을 구성하는 항목의 개수만큼 데이터베이스 스캔을 필요로 한다. 웹상의 로그파일은 거대한 데이터의 집합(dataset)이기 때문에 데이터베이스를 스캐닝 하는 것은 굉장히 많은 입출력 비용을 필요로 하고, 트랜잭션 내에서 발생한 항목집합을 계산하는 것은 많은 시간과 하드웨어 자원, 그리고 높은 메모리를 필요로 한다. 빈발항목 집합 후보의 수는 항목의 수라고 볼 수 있기 때문에 빈발항목집합을 찾기 위해 항목의 수를 줄이는 노력은 필수적이다. 로그파일에서 빈발항목 집합을 찾기 위해서는 두 가지 주요 이슈가 있다. 첫째는 입출력 비용을 최소화하기 위해 데이터베이스 스캐닝 횟수를 줄이는 것이고, 두 번째는 후보항목 집합과 트랜잭션을 선별하면서 필요한 탐색공간과 메모리를 줄이는 것이다. 연관규칙 마이닝의 몇 가지 알고리즘들이 개발되었고 각 알고리즘들은 장단점을 가지고 있다. 몇몇의 알고리즘은 모든 빈발항목 집합을 찾고(Zou, Q. 등, 2002), 몇몇의 알고리즘은 최대 빈발항목 집합을 찾거나 빈발항목 집합에 근사한 값(Pei, j. 등, 2000)을 찾는다. 이러한 알고리즘은 여러 측면에서 분석되고 분류되었다.

## 1.3 연구의 범위와 방법

본 논문에서는 효율적으로 탐색공간을 줄이면서 빈발항목 집합을 발견하기 위한 알고리즘을 제시한다. 제시된 알고리즘의 주요 장점은 알고리즘이 데이터베이스가 스캔되는 시점에 데이터베이스의 전체가 아닌 일부

만을 필요로 한다는 것이다. 저자는 데이터베이스 스캔을 최소화하기 위해 ItemsetCode 알고리즘(R. Ivancsy 등, 2006)을 적용하였다. ItemsetCode 알고리즘은 임의의 k- 빈발항목집합을 구하기 위해 $\log_2 k$ 번의 데이터베이스 스캔을 필요로 한다. 먼저 1 또는 2 빈발항목 집합을 구하기 위해 상 삼각행렬을 구성한다. 구성 과정에서 각각의 트랜잭션의 집합인 트랜잭션 리스트가 포함하고 있는 각각의 항목을 위한 아이템 트랜잭션 리스트(이하 ItemTransList)를 만든다. ItemTransList에는 항목 별로 항목을 탐색한 트랜잭션의 아이디(ID)들을 포함하고 있다. 구성 과정이 끝난 이후 얼마나 많은 1빈발항목 집합이 각 트랜잭션에 포함되어 있는지를 해시 테이블(hash table)을 생성하여 작성한다. 이 해시 테이블을 기준으로 트랜잭션들은 임의의 k-빈발항목 집합을 구성하는데 사용된다. 해당 트랜잭션을 선택하기 위한 원칙은 k-빈발항목 집합을 찾기 위해 트랜잭션이 적어도 k개의 1-빈발항목 집합들을 가져야 한다는 Apriori 알고리즘의 원칙을 기초로 한다. 이 논문에는 아래와 같은 구성을 따른다. 관련 연구 섹션은 연관규칙 마이닝의 기본 문제와 관련 연구를 설명하고, 연구 설계 섹션은 제시된 알고리즘을 설명한다. 실험과 결과 섹션에서는 제시된 알고리즘을 실제 웹 데이터에 적용하고, 그에 따른 결론을 도출하고 비교 결과를 보여준다.

# II. 관 련 연 구

## 2.1 문제 기술

I = $\{i_1, i_2, \cdots, i_n\}$ 은 웹 페이지 내의 모든 항목의 집합이고
T = $\{t_1, t_2, \cdots, t_N\}$ 은 데이터베이스의 모든 트랜잭션의 집합이다. 고유의 식별자(TID)를 가진 각 트랜잭션 $t_i$은 사용자 자신으로부터 선택된 항목들의 하위집합을 포함한다. X는 항목집합에서 포함된 항목의 집합이다. 항목 집합이 임의의 k개의 항목을 포함하고 있다면 k-항목집합이라 부른다. X가 트랜잭션 $t_i$의 하위 집합일 경우 X는 트랜잭션 $t_i$에 포함된 빈발항목이라 할 수 있다. 항목집합 X를 지지하는 트랜잭션의 개수는 항목집합을 포함하고 σ(X)라 표기한다. 항목집합은 최소지지도 임계치인 minsup 이상일 경우 빈발항목이라 한다. k개의 빈발한 항목을 가진 항목집합을 k-빈발항목집합이라고 부른다. 연관규칙은 분리된 항목집합 X와 Y를 X→Y의 형태로 표현하는 식을 의미한다. 규칙의 지지도는 s(X→Y) = $\frac{\sigma(X \cup Y)}{N}$ 이고 규칙의 신뢰도는 c(X→Y) = $\frac{\sigma(X \cup Y)}{\sigma(X)}$ 이며 이는 트랜잭션에 X가 포함되어 있을 경우 Y가 포함되어 있을 조건부 확률이다.
신뢰도는 규칙에 의한 추론을 통해 신뢰성을 측정하고, 최소신뢰도 임계점 이상인 경우에 의미를 가질 수 있다. 연관규칙 마이닝은 두 단계의 과정으로 구성된다. 첫 번째는 빈발항목 집합들을 찾아내고 이 후 빈발항목 집합들로부터 연관규칙을 생성한다. 빈발항목 집합을 찾는 과정에서 소요되는 연산과 입출력 비용은 규칙을 찾는 과정보다 더 많은 자원이 소요되기 때문에 빈발항목 집합을 찾기 위한 많은 알고리즘들이 개발되었다. 빈발항목 집합을 생성하는 과정에서 항목집합의 후보 수를 줄이기 위해 Apriori 원리가 사용되며, 항목집합이 빈발한 경우 모든 하위 집합들도 빈발하다고 할 수 있다. 반대로 항목집합이 빈발하지 않을 경우 모든 확대집합들도 빈발하지 않다고 할 수 있다.

## 2.2 Apriori 알고리즘

Apriori 알고리즘은 연관규칙의 대표적인 형태로 가장 잘 알려진 알고리즘이다. 연관규칙을 찾는 알고리즘 중에서 가장 먼저 개발되었으며 또한 가장 많이 쓰이는 알고리즘이다. Apriori 알고리즘은 항목들에 대한 선택 횟수를 기반으로 항목들 간의 연관규칙을 발견하기 위해 사용된다. 대표적으로 마켓에서의 소비자 물건 구매 패턴을 예로 들 수 있다. 분유를 구입하는 소비자는 기저귀를 같이 구매할 경우가 많다. 이러한 상품들의 연관관계를 파악하여 연관성이 있는 물건들을 서로 가까운 위치에 배치시키면 더 효율적으로 물건을 판매 할 수 있다.

Apriori 알고리즘은 첫 번째 데이터베이스 스캔에서 1-빈발항목 집합을 스캔한다. 그 이후 1-빈발항목 집합을 결합하여 2-빈발항목 집합의 후보 1-항목집합을 생성한다. 두 번째 데이터베이스를 스캔하는 동안, 후보 항목집합을 카운트 하고 최소지지도를 넘지 않는 항목집합을 잘라낸다. Apriori 알고리즘의 흐름의 예는 그림 4에 도식화하였다.


**그림 4:** Apriori 알고리즘 흐름도

Join Step : $C_k$ is generated by joining $L_{k-1}$ with inself

Prune Step : Any (k−1)−itemset that is not frequent cannot be a subset of a frequent k−itemset

Pseudo−Code

$C_k$ : Candidate itemset of size k

$L_k$ : frequent itemset of size k

$L_1$ = {frequent items};

for(k = 1; $L_k$ != ∅; k++) do begin

$C_{k+1}$ = candidates generated from $L_k$;

for each transaction t in database do

increment the count of all candidates in $C_{k+1}$ that are contained in t

$L_{k-1}$ = candidates in $C_{k+1}$ with min_sup

end

return $\cup_k L_k$;

**표 2:** Apriori 알고리즘 의사코드

위와 같은 Apriori 알고리즘의 프로세스는 임의의 k−빈발항목 집합을 찾기 위해서는 k번의 데이터베이스 스캔이 요구되므로 높은 입출력 비용이 소요된다. Apriori 알고리즘은 구현하기 쉽고, 이해하기 쉬우며 어느 정도 만족할 만한 결과를 얻을 수 있는 장점이 있는 반면, 후보 집합 생성시에 항목의 개수가 많아지면 계산의 복잡도가 매우 증가하게 되는 단점을 가지고 있다. Apriori 알고리즘의 입출력 비용을 개선하기 위해 많은 알고리즘이 제안되었다.

## 2.3 DHP 알고리즘

DHP(Direct Hashing and Pruning) 알고리즘(Park, J.S. 등, 1995) 은 Apriori 알고리즘의 발전된 변형 형태이다. DHP 알고리즘은 두 개의 주요한 특징이 있다. 첫 째는 효율적으로 후보 항목집합을 생성하고 점진적으로 각각의 불필요한 트랜잭션의 크기를 줄여나가는 것이고, 두 번째는 데이터베이스 내의 트랜잭션의 수를 줄임으로서 데이터베이스의 크기와 검색공간을 줄이는데 초점을 맞추었다. 초기 단계의 방대한 양의 후보 항목집합에서 빈발항목 집합들을 찾아내는 과정은 전체 프로세스의 성능을 좌우하는 요인이 된다. 미리 해시 테이블을 준비해 놓고 DHP는 임의의 k번째 데이터를 스캔하며 후보 k-항목집합에서 k-빈발항목 집합을 찾을 때 (k+1)-항목집합에 대한 정보를 해시 테이블에 수집한다. 이러한 과정을 통해 초기단계의 두 번째 단계까지만 데이터베이스 전체를 스캔하고 이후 단계에서는 데이터베이스의 크기를 줄여나가므로 Apriori와 비교하였을 경우 실행시간이 크게 감소될 수 있다. 해시 테이블의 크기가 클수록 이후 단계의 데이터베이스 스캔 시 검색공간이 줄어든다. 특히 2-빈발항목 집합을 생성하는 과정은 전체 알고리즘의 성능을 좌우할 만큼 중요한 단계를 수행한다. 2-빈발 항목집합을 생성한 이후의 다음 단계에서 성능을 크게 향상 시킬 수 있다.  그림 5은 DHP 알고리즘을 사용하여 트랜잭션을 감소하는 방법에 대해 도식화 하였다. t[i]는 트랜잭션 ID가 $i$에 위치한 항목이 k-항목집합 들의 한 항목으로 사용된 내역을 보여준다. t[i]에 위치한 항목들이 최소지지도를 넘지 않는 경우 t[i]는 (k+1)-빈발항목의 후보가 될 수 없기 때문에 트랜잭션 검색 시에 제외하게 된다. DHP 알고리즘은 해시 테이블의 크기에 따라 Apriori 알고리즘에 비해 전체 성능이 증가 할 수 있다. 그러나 Apriori와 마찬가지로 k-빈발항목 집합을 찾기 위해서 k번의 데이터베이스의 스캔이 필요하다

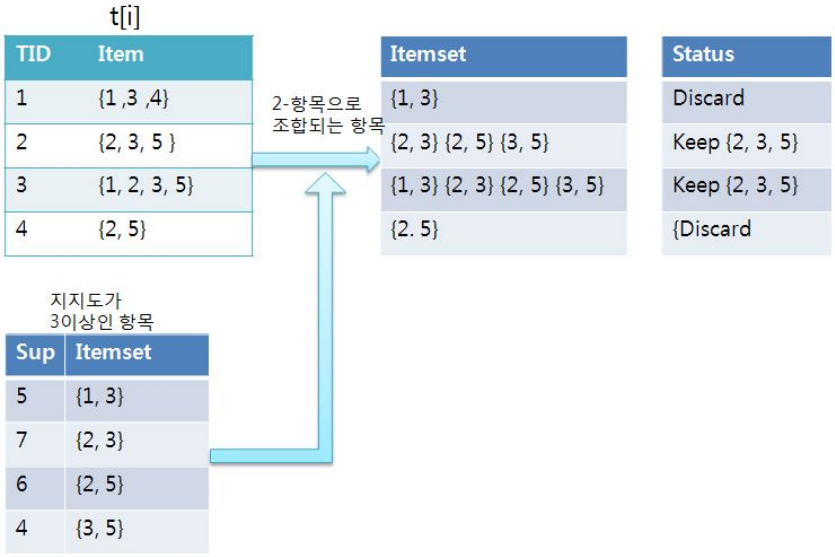



**그림 5:** DHP 알고리즘의 트랜잭션 감소

## 2.4 DIC 알고리즘

DIC(Dynamic Itemset Counting) 알고리즘(Brin, S. 등, 1997)은 각 단계에서 카운트 되는 항목집합들의 수를 줄임으로서 데이터베이스의 스캔을 수를 줄이는 것에 초점을 맞춘다. Apriori 알고리즘은 단계별로 진행하면서 빈발항목 집합을 찾는 반면, DIC는 항목집합을 증가시키는 동시에 빈발항목 집합을 찾는다. Apriori의 모든 항목집합들은 데이터베이스 스캔 시 지지도 계산을 위해 사용되고 지지도 계산이 끝나면 해당 항목집합들을 삭제한다. 따라서 k-항목집합을 찾기 위해서 k번의 데이터베이스 스캔을 수행해야 한다. DIC 알고리즘에서는 항목집합을 찾는 중간지점에 저장하여 지지도 계산을 처리하고 스캔 과정이 끝에 도달하면 항목집합들을 삭제한다. 예를 들어 그림 6와 같이 40,000개의 트랜잭션을 탐색하고 새로운 항목집합들의 지지도 계산을 시작하는 지점의 개수 M을 10,000이라고 설정한다. 전체 트랜잭션에서 1-항목집합들의 지지도를 계산하며 10,000개의 트랜잭션을 스캔을 마치면 2-항목집합들의 지지도 계산을 시작하며, 20.000개 트랜잭션 스캔을 마치면 3-항목집합들의 지지도 계산

을 시작한다. 모든 스캔이 완료되면 1-항목집합들의 지지도 계산은 끝날 것이고, 2-또는 3-항목 집합들의 지지도 계산을 위해 시작점으로 돌아가 작업을 수행한다. 10,000 트랜잭션 이후에는 2-빈발항목의 지지도 계산이 완료되고, 20,000 트랜잭션  이후에는 3-빈발항목의 지지도 계산이 완료된다(박종수, 1998). 전체적으로 Apriori와 비교했을 경우 데이터베이스 스캔의 횟수는 절반 정도로 줄어드는 효과를 볼 수 있다. 그러나 데이터베이스의 스캔의 수를 감소하더라도, 동시에 여러 크기의 항목집합을 카운팅 하는 것은 계산의 복잡성과 높은 메모리 비용이 발생한다.

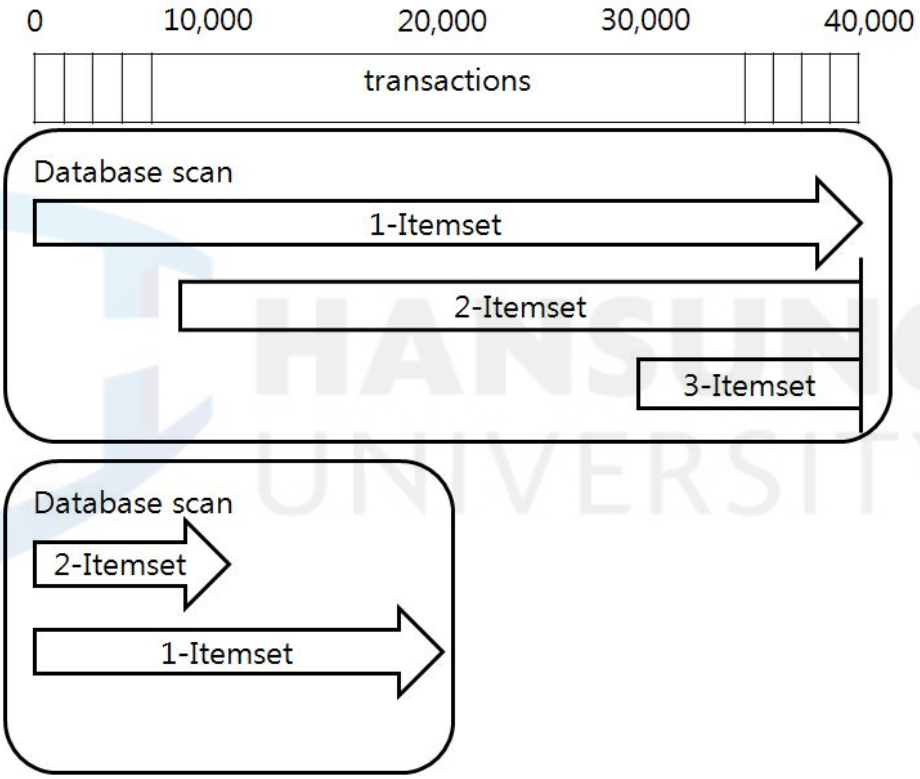



**그림 6:** DIC 알고리즘의 트랜잭션 탐색

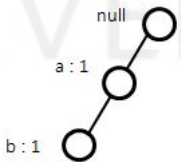
## 2.5 FP-growth 알고리즘

FP-growth(Frequent Pattern-growth) 알고리즘(Han, J. 등, 2004)은

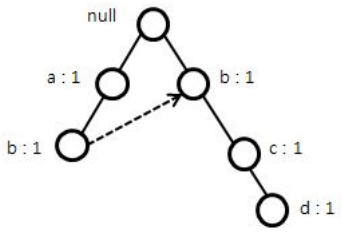
생성 및 시험 패러다임 방식의 Apriori 알고리즘과 다른 접근방식을 사용하고 있다. FP-growth에서는 Apriori 알고리즘이 후보 항목의 생성 시에 발생되는 시간이 크다는 단점을 고려하여 후보 항목을 생성하지 않고 빈번한 항목집합을 모두 생성한다. FP-growth는 $\{i_1,i_2,\cdots,l_k\}$ 이 빈발 항목집합일 경우 $\{i_1,i_2,\cdots,l_{k-1}\}$ 도 빈발항목 집합이라는 원리이다. 첫 번째 데이터 베이스 스캔시 1-빈발항목 집합의 지지도를 계산한다. 두 번째 데이터 베이스를 스캔하는 동안, 메인 메모리에 FP-tree라고 불리우는 소형 데이터 구조를 이용하여 데이터 집합을 인코딩하고 FP-tree로부터 직접 빈발항목의 집합을 추출한다. 데이터베이스의 크기가 적당한 경우 FP-growth 알고리즘은 Apriori 알고리즘보다 좋은 성능을 보일 수 있고 빈발항목 패턴 마이닝의 다른 방법들 보과 비교하여 보아도 효과적이다. 그러나 FP-growth 알고리즘의 성능은 데이터 집합의 압축인자에 의해 결정된다(Tan Pang-ning, 2006). 만약 생성된 FP-tree들이 너무 작다면, 알고리즘을 수행하는 메모리 용량의 성능과 비용이 현저하게 저하되는 단점이 있다.



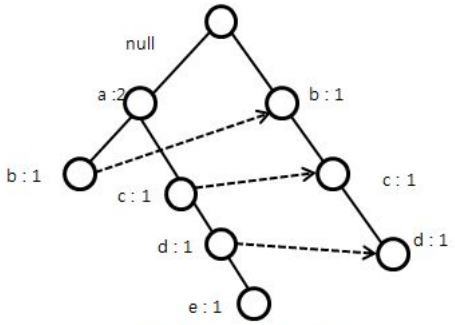
| TID | Item |
|---|---|
| 1 | {a,b} |
| 2 | {b, c, d} |
| 3 | {a, c. d. e} |
| 4 | {a, d, e} |
| 5 | {a, b, c} |




(1) After reading TID=1




(2) After reading TID=2




(3) After reading TID=3

**그림 7:** FP-tree 구성

## 2.6 ItemsetCode 알고리즘

ItemsetCode 알고리즘은 빈발 항목집합을 찾는 과정에서 시간과 사용 메모리를 줄이는 데 효율적인 알고리즘이다. ItemsetCode 알고리즘은 k-빈발항목을 구하기 위해서 데이터베이스의 접근 횟수가 $\log_2 k$로 감소한다. 또한 2-빈발항목 집합인 $L^2$에 코드 C를 도입하여 메모리 요구량을 줄였다. $L^2$내의 모든 요소 항목들을 부호화 한후, 후보 생성 규칙에 따라 $L^2$ 내의 두 개의 코들을 조합하여 3-빈발항목과 4-빈발항목의 후보를 생성한다. ItemsetCode 알고리즘은 항목 행렬 M을 생성하여 상 삼각행렬을 통해 대각 항목에는 1-빈발항목에 대해 카운트하고 비 대각 항목에는 2-빈발항목에 대한 카운트를 진행한다. 항목들은 사전 정의에 의해 색인이 되어 있기 때문에 항목 쌍들은 사전식으로 정렬된다. 항목행렬을 사용함으로써 1-또는 2-빈발항목들을 쉽게 찾을 수 있다. 항목행렬 M 내의 값 M[i,j](0≤i≤n, i≤j) 가 최소지지도 minsup 이상일 경우 2빈발항목 집합 {i,j}는 코드로 할당되고 최소지지도 미만일 경우 -1값으로 할당된다. 2-빈발항목 집합을 코드화 한 결과, 3-또는 4-빈발항목 집합을 구성시에 미리 형성된 코드 쌍을 카운트하며 효율적으로 후보 항목집합을 찾을 수 있다. 실험 결과는 실행 시간 및 메모리 사용량의 상당한 감소를 보여준다.

# III. 연 구 설 계

## 3.1 ItemTransList 알고리즘

이 절에서는 데이터베이스 스캔 중에 데이터베이스에서 트랜잭션의 선택적 로드를 통해 검색 공간을 줄이는 하나의 알고리즘을 제안한다. 연관규칙 알고리즘 중에 ItemsetCode 알고리즘이 데이터베이스 스캔 횟수를 최소화하고 메모리 사용량을 많이 줄일 수 있기 때문에, 지지도 카운팅을 효과적으로 하기 위해 이 알고리즘을 적용하였다. 연관규칙 탐사에 많이 사용되는 Apriori 알고리즘은 빈발항목의 개수가 하나 증가할 때마다 지지도를 산정하기 위해 데이터를 매 번 스캔해야 한다. ItemsetCode 알고리즘을 사용하면 k-빈발항목집합을 찾기 위해 데이터베이스 접근횟수가 $\log_2 k$로 감소한다. 데이터베이스를 스캔하여 항목 별로 해당 항목을 포함한 트랜잭션 리스트를 만든다. 그리고 항목에 대한 최소지지도 minsup을 산정하여 최소지지도 보다 큰 지지도를 갖는 항목을 빈발항목으로 선정한다. 그리고 빈발항목을 포함하지 않는 트랜잭션은 다음 데이터베이스 스캔 대상에서 제외시킨다. k-빈발항목을 찾기 위한 과정에서 k값이 작을 때는 항목별 리스트를 작성하는데 시간과 자원에 부담에 따르지만 k값이 커질수록 스캔 대상 트랜잭션의 감소가 커지고 이에 따른 탐색 공간의 축소로 시간과 연산의 복잡도가 감소된다. 제안 알고리즘은 4단계로 구성되어 있고 각 단계에 대한 설명은 다음절에서 설명한다.

## 3.2 ItemTransList 알고리즘 처리 1 단계

홈페이지에서 사용 중인 전체 항목의 개수를 $n$이라고 하면 그림 8과 같이 $n \times n$ 항목 2차원 배열 M을 생성한다. 홈페이지의 항목들은 사전적 순서대로 0부터 n-1 까지 ID를 가지고 있고 생성된 배열에 index와 동일하게 홈페이지 항목들을 위치시킨다. 항목 행렬의 각 공간에 아이디가 부여

된 형태는 그림 9에 도식화 하였다.

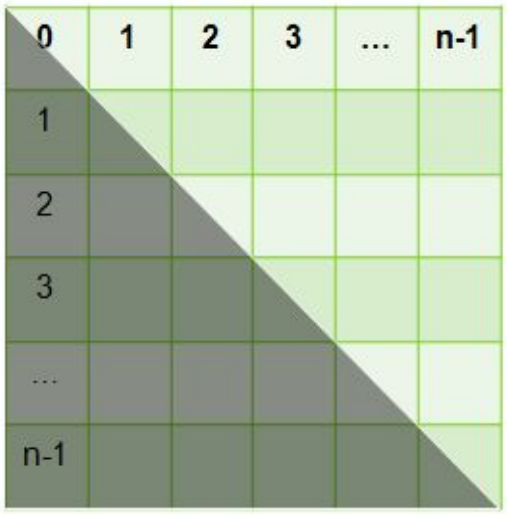



**그림 8:** $n \times n$ 항목 행렬 M

| (0,0) | (0,1) | (0,2) | (0,3) | … | (0, n-1) |
|---|---|---|---|---|---|
| null | (1,1) | (1,2) | (1,3) | … | (1, n-1 |
| null | null | (2,2) | (2,3) | … | (2, n-1 |
| null | null | null | (3,3) | … | (3, n-1 |
| null | null | null | null | … | (4, n-1 |
| null | null | null | null | null | (n-1, n-1) |

**그림 9:** M에 부여된 ID 구조

첫 번째 데이터베이스 스캔에서 트랜잭션에서 저장된 $i$항목을 항목배열의 대각항 $(i,i)$에 ID가 $i$인 항목의 발생횟수를 카운트 한다. $i$항목을 카운트 한 이후 같은 트랜잭션 내에 저장된 $j$항목에 대해 비대각항 $(i,j)$에는 2개의 항목 ID $(i,j)$의 발생 횟수를 카운트 한다. 여기서 트랜잭션 데이터의 순서는 고려하지 않으므로 항목 행렬은 상 삼각 행렬이 된다. 그림 8과 같이 어두운 삼각형 부분의 공간은 사용하지 않는다. 이러한 과정을 반복하며 M 배열의 해당 항목이 위치된 값을 카운트 처리하며 트랜잭션을 탐색한다. M 배열을 처리하는 과정에서 각 항목별로 항목을 포함하는 트랜잭션 ID의 리스트 배열 ItemTransList을 생성한다. ItemTransList의 인덱스는 홈페이지 항목 ID와 동일하게 사용한다. 앞의 과정이 모두 완료되면 M 배열의 대각항에는 1-항목집합의 발생횟수가 카운트 되어 저장되고, 비 대각항에는 2-항목집합들이 카운트 되어 저장된다. ItemTransList 배열에는 항목 별로 항목을 포함한 트랜잭션의 ID들이 저장된다. ItemTransList 배열에는 각 항목별로 중복된 트랜잭션 ID가 발생할 수 있으니 중복 트랜잭션 ID를 제거해준다.

| | | | | | |
|---|---|---|---|---|---|
| 120 | 89 | 92 | 43 | ... | 45 |
| null | 87 | 66 | 78 | ... | 16 |
| null | Null | 24 | 12 | ... | 9 |
| null | null | Null | 52 | ... | 23 |
| null | null | Null | Null | ... | 87 |
| null | null | null | null | Null | 189 |

**그림 10:** M 배열 공간이 완성된 구조

| Item ID | TID |
|---|---|
| 0 | {1, 4, 5 ,9} |
| 1 | {2, 8, 9, 10 ,11 } |
| 2 | {4, 6} |
| ... | ... |
| n-1 | t-1 |

**그림 11:** ItemTransList 구조

```
M: createMatrix(n);
ItemTransList := new ArrauList[n];
for each transaction t
    for each i ∈ t do
    ItemTransList[i].Add(t.ID);
        for each j ∈ t ,j > i do
            M[i,j] ++
ItemTransList : = duplicatedRemove();
```

**표 3:** ItemTransList 알고리즘 1 단계 의사 코드

## 3.3 ItemTransList 알고리즘 처리 2 단계

1 단계 과정이 완료되고 트랜잭션별로 1-빈발항목의 개수를 포함하는 TransItemArray 배열을 생성한다. 배열의 인덱스는 트랜잭션의 ID와 같다. 2-빈발항목의 개수를 나타내기 위한 counter 변수를 초기화 하고 최

소지지도 minsup을 설정한다. minsup은 프로그램 외부에서 설정하도록 하여 minsup의 값을 변경할 수 있도록 한다. 항목 카운트 배열 M의 대각항을 검사하여 1-빈발항목을 찾는다. 1-빈발항목을 찾는 과정에서 1-빈발항목의 최소지지도를 상회하는 항목을 포함하는 ItemTransList의 행을 찾게 되면 해당 리스트에 포함되어 있던 트랜잭션ID를 호출한다. TransItemArray에 호출한 트랜잭션 ID의 인덱스로 접근하여 1-빈발항목을 포함한 개수를 카운트한다. minsup을 상회하는 1-빈발항목을 포함한 M 배열의 행에서 비대각항을 검사하여 최소지지도를 상회하는 2-빈발항목을 찾아  결과 리스트에 저장한다. 저장된 비대각항의 M 배열의 위치에는 ItemsetCode 알고리즘에서 제시한대로 항목 행렬에 코드를 할당한다. 코드는 다음과 같다.

$c_i = n+i,\ i = 0,1,\cdots,K-1$  $K$ 는 2-빈발항목의 총 개수

2-빈발항목 minsup를 넘지 않는 M 배열의 위치에는 -1값을 부여한다. 코드 할당 과정을 예를 들어 설명하면 홈페이지 전체 항목의 개수 n이 300이고 M 배열에서 2-빈발항목으로 선정된 첫 번째 공간에는 300의 값으로 재 설정해준다. 2-빈발항목으로 선정되지 않는 공간에는 -1값을 주어 추후 M 배열을 재 사용시에 걸러내어 준다.

```
TransItemArray := new int[t-1]
counter :=0
for(i=0; i<n-1;i++)
    if M[i,i] > minsup then
    TransItemArray[ItemTransList[i].getId()]++
    for(j=i+1; j<n;j++)
        if M[i,j] > minsup then
        resultList.Add({[i,j], M[i,j]})
        M[i,j] := n+counter
        counter++
        else
```

```
        M[i,j] := -1
codemax := counter
```

**표 4:** ItemTransList 알고리즘 2 단계 의사 코드

## 3.4 ItemTransList 알고리즘 처리 3 단계

2-빈발항목을 이용하여 3- 또는 4-빈발항목을 찾기 위하여 지지도 계산 행렬 L 배열을 을 만든다. L의 행렬은 $c_i c_j$로 이루어진 3-또는 4-빈발항목의 앞부분의 2-빈발항목인 $c_i$로 구분되어 $c_i$행에 대한 인덱스는 $c_i = n+i,\ i = 0,1,\cdots,K-1$이 된다. 각 행의 길이는 (K-$i$+1) 이지만 메모리 공간을 절약하기 위해 3-빈발항목을 만들 수 있는 $c_i$중에 아래의 <규칙 1> 을 만족하는 최소인 코드부터 최대 코드인 $c_{k-1}$까지 포함하는 리스트를 사용한다. L 행렬에 대한 구조는 그림 12과 같다.

<규칙 1> $c_i c_j$형태로부터 만들어지는 3-빈발항목의 후보는 ($c_i$의 두 번째 항목) = ($c_j$의 첫 번째 항목) 이다.

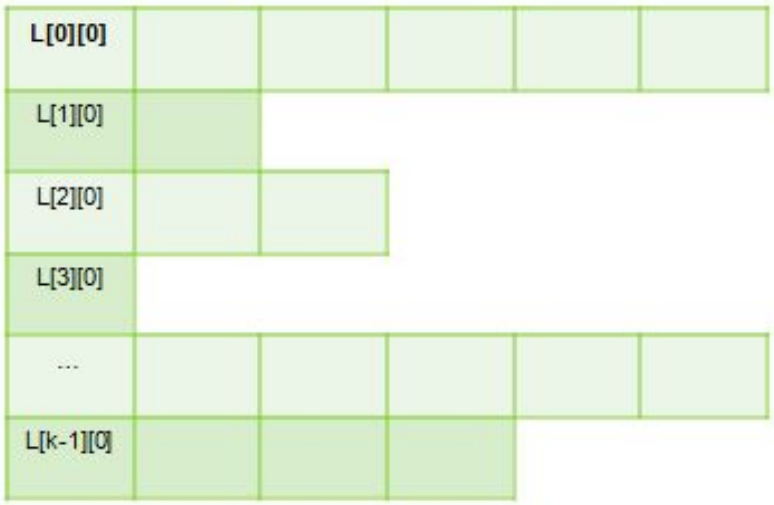



**그림 12:** L배열 구조의 예

2차원 배열 L 생성 후 M 배열을 탐색하며 L 배열의 각 공간에 위치하는 항목을 카운트하는 작업을 수행한다. 먼저 M 배열의 대각항을 검사하여 항목이 minsup을 넘는 $i$행을 찾고, $i$행을 검사하여 code가 −1로 할당되지 않는 값 중 최소 값과 2−빈발항목의 총개수 counter와 뺀 값을 L배열에 크기로 할당한다.

```
L := new object[codemax]
for(i=0 ; i < n; i++)
    if(M:[i,i] > minsup)
    for(j=i+1; j<n;j++)
    code = M[i,j]
    if code != −1
    begencode = min(M[i,:])
    L[i] = new int[codemax − begincode]
```

**표 5:** ItemTransList 알고리즘 3 단계 의사 코드

## 3.5 ItemTransList 알고리즘 처리 4 단계

처리 2단계에서 생성한 TransItemArray를 이용하여 빈발항목이 2개 이하인 트랜잭션은 두 번째 데이터베이스 스캔 시 제외시킨다. 트랜잭션의 크기를 줄임으로써 단계 4 처리 과정의 전반적인 실행시간과 탐색 공간을 감소시킨다. 트랜잭션을 호출 후 먼저 T1리스트를 생성하고 트랜잭션을 탐색하며 2단계에서 설정한 항목 행렬 M의 해당공간에 설정된 2−빈발항목의 코드 $c_i$를 T1리스트에 넣어준다. 이 후 T1 리스트를 탐색하며 2개의 빈발항목 $c_ic_j$포함한 L행렬의 $(i,j)$공간에 위치된 값을 증가시켜준다. L행렬의 값 L$(i,j)$가 최소 지지도를 상회하면 3−또는 4−빈발항목의 후보가 될 수 있다. L$(i,j)$은 2−빈발항목의 코드 값인 $c_i$와 $c_j$값으로 설정되어 있

으므로 단계 2의 resultList에서 저장하였던 2-빈발항목과 매칭된 값을 불러온다. 앞서 제시한 <규칙 1>의 조건을 만족시키게 되면 3-빈발항목, <규칙 2>를 만족시키면 4-빈발항목으로 선정할 수 있다.

<규칙 2> $c_ic_j$형태로부터 만들어지는 4-빈발항목의 후보는 ($c_i$의 첫 번째 항목) < ($c_i$의 두 번째 항목) < ($c_j$의 첫 번째 항목) < ($c_j$의 두 번째 항목)


**그림 13:** <규칙 1> 조건


**그림 14:** <규칙 2> 조건

```
for each transaction t
    t1 := createList()
    for(i=0; i<t.size-1; i++)
        for(j=i+1; j<t.size; j++)
        if M[t[i],t[j]] != -1 then
            t1.Add(M[i,j])
    for(i=0; i<t1.size-1; i++)
        for(j=i+1; j<t1.size; j++)
            L[t1[i]][t1[j]]++
for(i=0; i<L.size-1; i++)
    if L[i] != null
    for(j=0; j<L[i].size; j++)
    if L[i][j] > minsup then
        c1 = i+n
        c2 = j + L[i][0]
        sup = L[i][j]
```

```
p := resultList(c1−n).getItems()
q := resultList(c2−n).getItems()
if p[1] != q[1] and p[2] != q[2] then
    if p[2] = q[1] then
    resultList.Add({{p[1], p[2], q[2]}, sup})
    else
    if p[1] < p[2] < q[1] < q[2] then
    resultList.Add({{p[1], p[2], q[1], q[2]}, sup})
```

**표 5:** ItemTransList 알고리즘 4 단계 의사 코드

## 3.6 ItemTransList 알고리즘 처리 결과

제안된 ItemTransList 알고리즘은 단계 1 과정에서 ItemTransList를 생성하고 단계 2 과정에서 TransItemArray를 생성함으로써 메모리를 사용량을 증가시켜 성능을 저하시켰다. 그러나 생성된 TransItemArray를 이용하여 알고리즘은 2번째 데이터베이스 스캔에서 1−빈발항목이 적어도 3개 이상 포함하지 않는 트랜잭션을 제거하여 탐색영역이 축소되며, 이로 인해 입출력 비용과 연산의 복잡도가 감소하였다. $2^k$−빈발항목을 찾기 위해 k번의 데이터베이스를 스캔 할 시에도 TransItemArray를 지속적으로 적용하여 트랜잭션의 호출 수를 줄이기 줄일 수 잇고, 그 성능은 k가 증가함에 따라 현저하게 증가된다. 단계 처리 프로세스에 따른 전체 의사코드는 표 6에 제시되었고 ItemTransList의 전체 프로세스 과정은 그림 15에 제시되었다.

```
procedure ItemTransList()
M: createMatrix(n);
ItemTransList := new ArrauList[n];
for each transaction t
```

```
    for each i ∈ t do
    ItemTransList[i].Add(t.ID);
        for each j ∈ t ,j > i do
            M[i,j] ++
ItemTransList : = duplicatedRemove();
TransItemArray := new int[t-1]
counter :=0
for(i=0; i<n-1;i++)
    if M[i,i] > minsup then
    TransItemArray[ItemTransList[i].getId()]++
    for(j=i+1; j<n;j++)
        if M[i,j] > minsup then
        resultList.Add({[i,j], M[i,j]})
        M[i,j] := n+counter
        counter++
        else
        M[i,j] := -1
codemax := counter
L : = createCounterStructure()
for each transaction t
    t1 := createList()
    for(i=0; i<t.size-1; i++)
        for(j=i+1; j<t.size; j++)
        if M[t[i],t[j]] != -1 then
            t1.Add(M[i,j])
    for(i=0; i<t1.size-1; i++)
        for(j=i+1; j<t1.size; j++)
            L[t1[i]][t1[j]]++
for(i=0; i<L.size-1; i++)
```

```
    if L[i] != null
    for(j=0; j<L[i].size; j++)
    if L[i][j] > minsup then
    decode(i+n, j+L[i][0], L[i][j])
```

```
procedure createCounterStructure()
L := new object[codemax]
for(i=0 ; i < n; i++)
    if(M:[i,i] > minsup)
    for(j=i+1; j<n;j++)
    code = M[i,j]
    if code != -1
    begencode = min(M[i,:])
    L[i] = new int[codemax - begincode]
```

```
procedure decode(c1, c2, sup)
p := resultList(c1-n).getItems()
q := resultList(c2-n).getItems()
if p[1] != q[1] and p[2] != q[2] then
    if p[2] = q[1] then
    resultList.Add({{p[1], p[2], q[2]}, sup})
    else
    if p[1] < p[2] < q[1] < q[2] then
    resultList.Add({{p[1], p[2], q[1], q[2]}, sup})
```

**표 6:** ItemTransList 의사코드

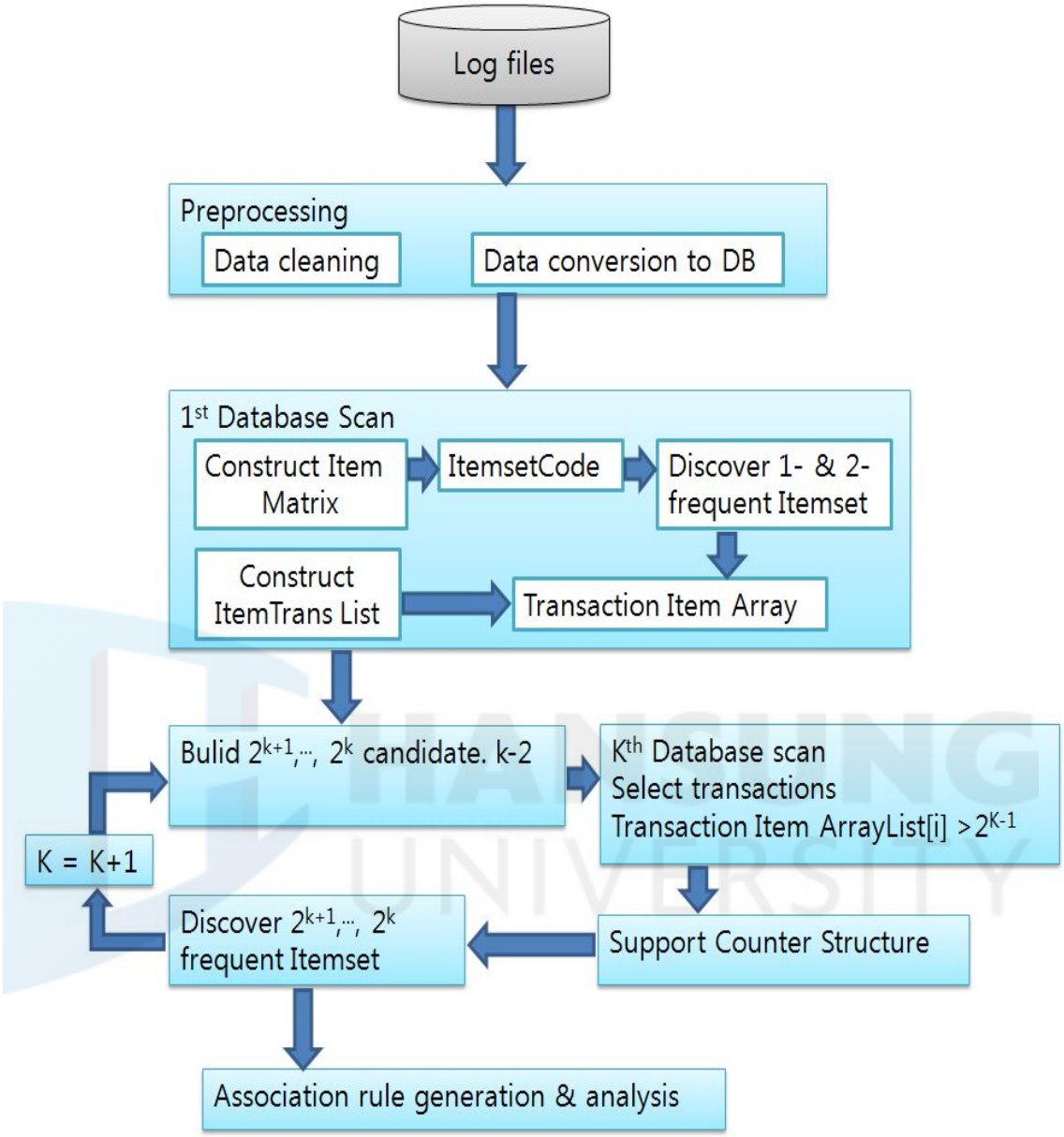



**그림 15:** ItemTransList 처리과정

# IV. 사 례 연 구

## 4.1 전처리 과정

연구는 저자의 대학교 홈페이지의 사용자 웹 로그 데이터에서 진행되었다. 모든 트랜잭션들의 IP는 개인정보 보호를 위해 특정 방식으로 변환하였다. 웹 로그 데이터는 대학교 홈페이지 사용자의 웹 접속로그 데이터인 아파치(apache) 로그 데이터를 데이터베이스에 입력 한 후 전처리 과정을 통해 웹 로그 데이터로 변경하였다. 아파치 로그에는 이미지, wma, js 등을 포함한 모든 접근 데이터의 기록이 남기 때문에 불필요한 로그데이터 및 URL경로에 붙어 있는 불필요한 매개 변수 값이 포함되어 있다. 표 8는 데이터 정제 전의 아파치 로그 데이터를 보여준다. 데이터 정제 후 데이터는 알고리즘에 대한 적절한 형태를 갖기 위해 오라클 데이터베이스로 변환한다. 표 9은 오라클 데이터베이스에서 처리된 데이터를 보여주며 표 10은 웹 로그 연구에 사용되는 대학 홈페이지 페이지 명, ID, 페이지 URL을 보여준다. 홈페이지의 전체 항목의 수는 300이고 각 항목은 자신만의 고유한 URL을 가진다. 표 11는 데이터 정제 과정에서 홈페이지에 접근한 하루 동안의 아파치 로그 기록으로부터 제거된 데이터의 형태와 양을 보여준다. 개발 환경 구축을 위해 언어는 JAVA DBMS는 ORACLE 11G를 사용하였다.  모의 실험은 CPU 3.1Gz의 데스크 탑으로 시행되었다.

| |
|---|
| 87.217.136.176 - - [05/Apr/2014:00:01:16 +0900] "GET /web/www/intro_01_06_02_t2?p_p_id=EXT_BBS&p_p_lifecycle=0&p_p_state=normal&p_p_mode=view&p_p_col_id=column-1&p_p_col_count=1&_EXT_BBS_struts_action=%2Fext%2Fbbs%2Fview_message |

**표 8:** 아파치 로그 데이터

| IP | REGIST_DATE | URL |
|---|---|---|
| 87.217.136.176 | 2014-04-05 | /web/www/intro01_06_02_t2 |
| 190.173.98.232 | 2014-04-05 | /web/www/home |
| 190.173.98.232 | 2014-04-05 | /web/www/college_01 |
| 165.200.100.51 | 2014-04-05 | /web/www/cmty_03_03 |
| 122.76.270.183 | 2014-04-05 | /web/www/cmty_03_01 |

**표 9:** 오라클 데이터베이스 데이터

| PAGE_NAME | PAGE_URL | ID |
|---|---|---|
| Design art | web/www/enter_03 | 0 |
| Korean Language programs | web/www/enter_04 | 1 |
| Undergraduate | web/www/college_01 | 2 |
| Graduate School | web/www/college_02 | 3 |

**표 10:** 대학교 홈페이지 URL

| Date | 2014-04-06 |
|---|---|
| Total Data | 1,635,099 |
| JPG | 345,533 |
| GIF | 539,166 |
| PNG | 124,982 |
| JS | 365,279 |
| CSS | 186,201 |
| ICON | 3,448 |
| TXT | 145 |
| … | … |
| Transaction used | 13,214 |

**표 11:** 제거된 데이터의 형태와 수

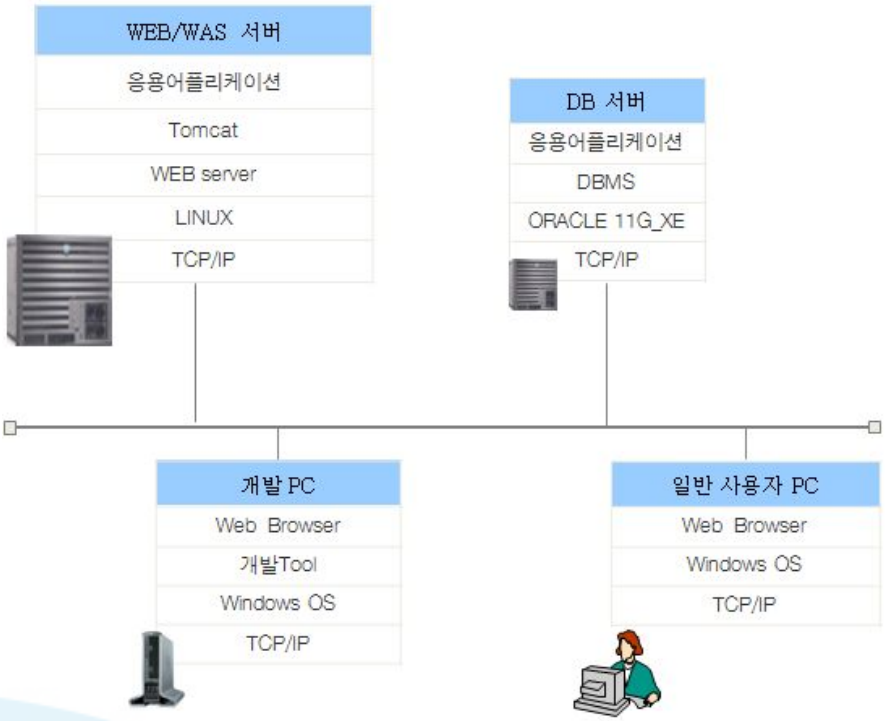



**그림 16: 시스템** 개발 환경

## 4.2 결과 분석

본 논문에서 적용한 ItemsetCode 알고리즘에서 항목 행렬 M은 첫 번째 데이터베이스 스캔시에 생성되었다. ItemTransList의 데이터 리스트는 데이터 내의 페이지 URL에 해당하는 웹 페이지의 주소와 홈페이지 내의 전체 URL 항목들과 같은 크기로 생성되었다. 여기서 페이지 URL은 항목을 의미한다. ItemTransList의 요소는 항목이 포함된 IP에 해당하는 트랜잭션 ID이다. 트랜잭션 ID는 항목 행렬을 생성하는 과정에 할당되는 IP의 인덱스(index)이고  트랜잭션에 포함된 항목이 ItemTranList요소에 있다면 해당 트랜잭션 ID를 ItemTransList에 저장한다. 각 트랜잭션들이 로드되면, 트랜잭션 내의 항목들은 표 10의 주소에 일치하는 항목 ID 형태로 변환된다. 항목 행렬 M에서 상삼각 행렬을 기준으로 하여 대각항과 비대각항은 카운트 된다. 그와 동시에 트랜잭션 ID는 ItemTransList의 항목 ID에 주소화 된 리스트의 요소로서 저장된다. 첫 번째 데이터베이스 스캔

이후, 항목 행렬 M과 ItemTransList가 생성되고 주어진 최소지지도 minsup에 의해 1-또는 2-빈발항목 집합을 찾는다. 여기서는 알고리즘의 카운터 구조 L을 사용한다. 1-빈발항목 집합인 ItemTransList을 기반으로 transItemArray를 생성한다. 1-빈발항목 집합의 트랜잭션 ID들은 각 트랜잭션을 포함하는 1-빈발항목 집합의 개수를 포함하는 transItemArray 을 만들기 위해 사용된다. 두 번째 데이터베이스 스캔하는 동안, 3개 이상의 1-빈발항목 집합을 가진 트랜잭션들만 3-또는 4-빈발항목 집합을 구하기 위해 사용된다. 표 12은 3개 미만의 1-빈발항목 집합을 가진 트랜잭션 ID 그룹이 제거된 수치이며 표 13은 트랜잭션의 데이터가 제거된 수치이다. 트랜잭션 ID 그룹의 총 개수는 8,320개 이며 트랜잭션의 데이터  총 개수는 95,115개이다. 표는 최소지지도 minsup가 증가함에 따라 증가되는 검색 공간의 감소율을 보여준다. 최소지지도 minsup이 증가할수록 빈발항목의 개수가 줄어들고 이를 포함하지 않는 트랜잭션의 수는 증가하며, 두 번째 데이터베이스 스캔시 제거해야할 트랜잭션의 수가 증가하여 탐색영역이 축소된다. 이로 인해 데이터 입출력 비용과 연산의 복잡도를 감소시킨다.

| Support(%) | 0 | 1 | 2 | more than 3 | Reduced(%) |
|---|---|---|---|---|---|
| 0.5% | 339 | 3,691 | 2,007 | 2,283 | 72.5% |
| 0.6% | 456 | 3,703 | 2,008 | 2,153 | 74.1% |
| 0.7% | 511 | 3,707 | 2,009 | 2,093 | 74.8% |
| 0.8% | 539 | 3,717 | 2,020 | 2,044 | 75.4% |
| 0.9% | 595 | 3,730 | 2,086 | 1,909 | 77.0% |
| 1.0% | 650 | 3,748 | 2,099 | 1,823 | 78.1% |

**표 12:** 지지도 증가에 따른 트랜잭션 ID 그룹의 제거 수

| Support(%) | 0 | 1 | 2 | more than 3 | Reduced(%) |
|---|---|---|---|---|---|
| 0.5% | 645 | 13,993 | 12,373 | 68,104 | 28.4% |
| 0.6% | 1,037 | 14,462 | 12,999 | 66,617 | 30.0% |
| 0.7% | 1,268 | 14,757 | 13,085 | 66,005 | 30.6% |
| 0.8% | 1,344 | 15,003 | 13,423 | 65,345 | 31.3% |
| 0.9% | 1,724 | 15,114 | 14,070 | 64,207 | 32.5% |
| 1.0% | 1,981 | 15,217 | 14,325 | 63,592 | 33.1% |

**표 13:** 지지도 증가에 따른 트랜잭션 데이터의 제거 수

메모리 사용량을 비교하기 Apriori 알고리즘, ItemsetCode 알고리즘과 제안된 ItemTransList알고리즘을 웹 로그 데이터에 적용하였다. 그림 17는 세 개의 알고리즘에 대한 메모리 사용량을 나타낸다. 두 번째 데이터베이스 스캔시, ItemTransList 알고리즘은 트랜잭션의 감소율로 인하여 ItemsetCode 알고리즘에 비해 약 30%정도의 적은 메모리를 사용한다.

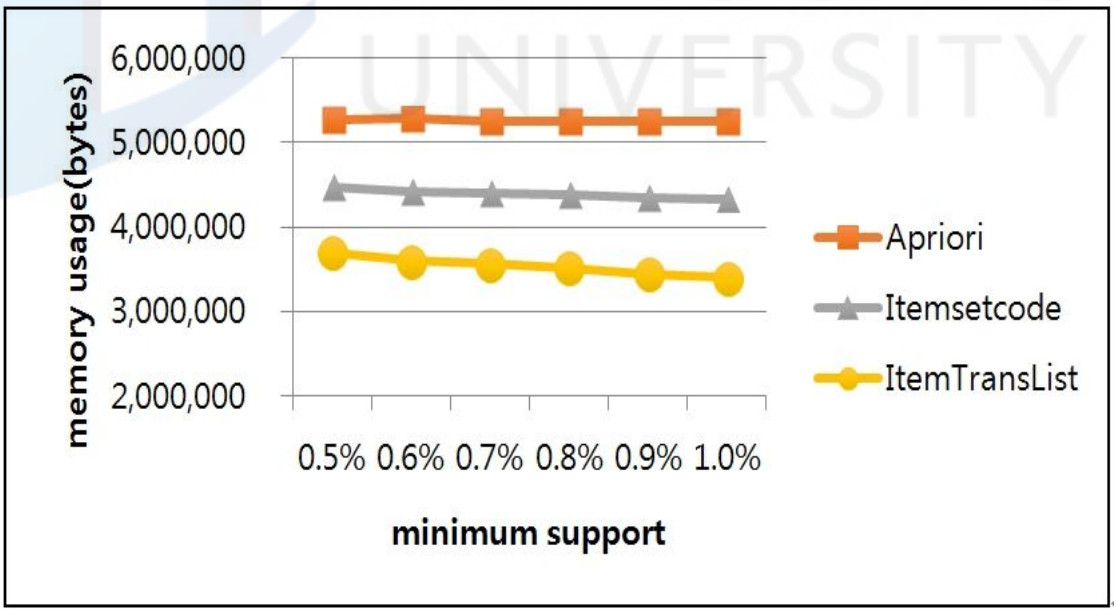



**그림 17:** 3개 알고리즘의 메모리 사용량

대학교 웹로그 데이터를 적용하여 ItemsetCode 알고리즘과 ItemTransList 알고리즘의 실행시간을 비교 분석하였다. 그림 18은 두 개의 알고리즘의 실행시간을 나타낸다. 두 알고리즘 모두 지지도 증가에 따라 낮은 폭으로 실행시간이 감소하였다. 제안된 ItemTransList 알고리즘은 실행 시간을 30% 이상 감소시키며 ItemsetCode 알고리즘에 비해 실행 속도가 빠른 것을 볼 수 있다.

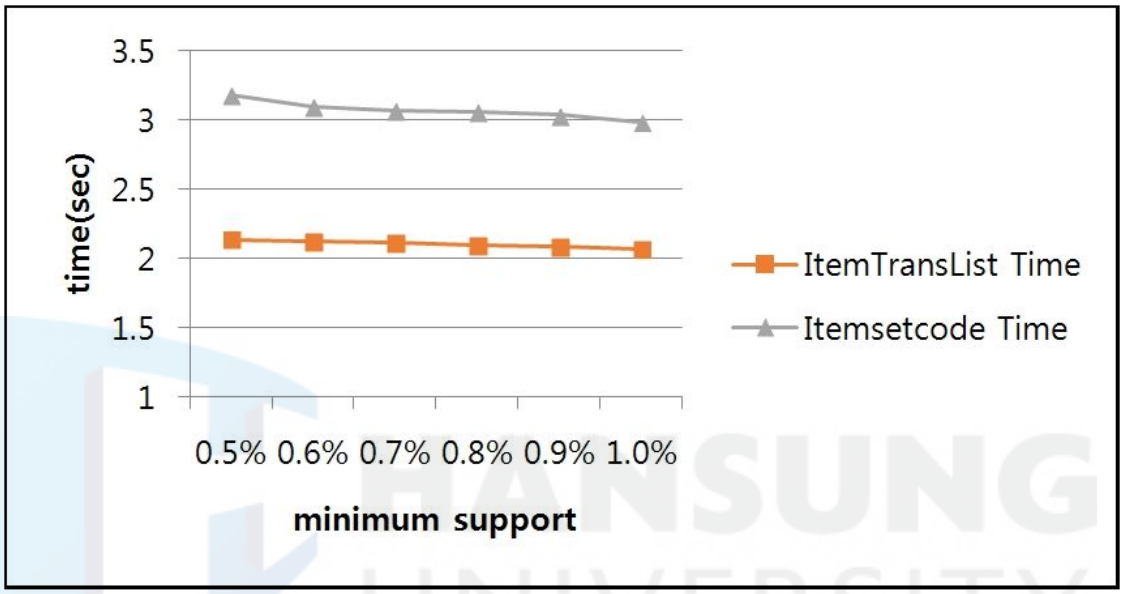



**그림 18:** 2개 알고리즘의 실행시간

# V. 결 론

## 5.1 평가 및 고찰

웹 로그는 상당히 많은 양의 데이터이기 때문에 사용자들의 웹 접근패턴을 분석하기 위해서는 많은 계산시간과 하드웨어 리소스를 필요로 한다. 연관 규칙을 찾아내는 것은 굉장히 비싼 비용과 높은 입출력 비용이 소요된다. 이 논문에서 우리는 탐색공간을 줄임으로써 빈발 항목집합을 효율적으로 찾아 낼 수 있는 알고리즘을 제안한다. 제안된 알고리즘의 주요 장점은 알고리즘이 데이터베이스 스캔하는 과정에서 그 일부 데이터베이스만을 필요로 한다는 것이다. 제안된 알고리즘 ItemTransList는 본 대학교 홈페이지의 웹 접속로그 분석에 적용하여 3-또는 4-빈발항목을 찾고, 또한 순차적으로 5-, 6-, 7-, 8-빈발항목을 3번째 데이터베이스 스캔시 찾을 수 있다. 이 과정에서 2차 데이터베이스를 스캔할 때 1-빈발항목을 적어도 3개 이상 포함하지 않는 트랜잭션을 제거하여 탐색 영역을 축소할 수 있다. 연구 결과는 ItemsetCode 알고리즘을 사용했을 때와 비교해보면 시행된 시간과 입출력 비용이 줄어든 것을 보여준다. 연구케이스에서 트랜잭션의 약 30%가 제거 되었고, 두 번째 데이터베이스 스캔시에 탐색공간은 30%정도로 감소되었다. 찾은 빈발항목 집합은 웹페이지 디자인에 반영하여 자주 사용되는 항목을 그룹화 하여 제공함으로서 이용에 편리를 기할 수 있으며, 또한 사용자 개인에 대한 개인 맞춤형 웹서비스를 제공할 수 있다. k-빈발 항목을 찾기 위해 k를 증가시키면서 알고리즘의 효율성은 높아질 수 있다.

## 5.2 추후 연구 및 적용분야

추후 연구 분야는 ItemTransList를 기반으로 어떻게 트랜잭션의 길이를 감소시킬 수 있는지와 k가 방대할 때 k-빈발항목 집합을 찾을 수 있도록

계산 구조를 발전시키는 것이다. 또한 홈페이지 전체 항목수가 크거나 트랜잭션의 크기가 방대한 규모를 가질 때 ItemTransList 알고리즘에서 생성하는 ItemTransList 와 TransItemArray의 메모리의 효율 구조를 발전시켜 메모리 에러의 발생요건을 저하시키는 것이다.

제시된 알고리즘은 컨텐츠의 내용정보, 사용자의 인구통계학적 정보를 필요로 하지 않으며 사용자들의 행동기록인 로그데이터만을 가지고 정확도가 높은 추천이 가능하기 때문에 적용범위가 넓다. 하지만 텍스트 정보 이외의 내용정보를 추천하기 어렵다. 온톨로지(ontology)를 구성하여 정보간의 관계성, 유사성을 파악할 수 있게 된다면 더욱 더 효과적인 추천이 가능해 질 것이다.

빈발항목을 찾는 개수가 점차 증가함에 따라 홈페이지에서 사용자들의 활동은 예상이 가능해진다. 따라서 개인화된 웹서비스가 가능해져 사용자들의 이용의 편리를 제공할 수 있으며, 또한 다양한 마케팅이나 연구 등 다양한 분야에 적용하여 활용할 수 있을 것이다.

# 참 고 문 헌

## 1. 국내문헌


이재규, 권순범, 임규건. (2005).『경영정보시스템원론(2판)』, p.534. 법영사.

이지은. (2002),『Intelligent Miner 의 마이닝 알고리즘 III (연관성 분석)』. 한국 아이비엠

함유근, 채승병. (2012).『빅데이터, 경영을 바꾸다』. 삼성 경제연구소

한형상, 이창호. (2012).『빅데이터 분석의 현황과 발전 전략』. 한국산업 기술평가관리원(KEIT)

박종수. (1998).『연구규칙 탐사 알고리즘에 대한 조사』. 성신여자 대학교

## 2. 국외문헌


Galit Shmueli, Nitin R. Patel, Peter C. Bruce. (2009). Data Mining for Business Intelligence

R. Agrawal, T. Imilienski, and A. Swami. (1993). “Database Mining: A Performance Perspective,” IEEE Transactions on Knowledge and Data Engineering, 5(6): pp. 914-925.

R. Agrawal, T. Imilienski, and A. Swami. (1993). "Mining Association Rules between Sets of Items in Large Databases." Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. pp.207-216.

Zou, Q., Chu, W.W. and Lu, B. (2002). “SmartMiner: a depth first algorithm guided by tail information for mining maximal frequent itemsets,” Proc. of the 2002 IEEE International Conference on Data Mining (ICDM'02), Maebashi City, Japan, December 09-12, pp.570-578.

Pei, j., Han, J. and Mao, R. (2000). “CLOSET: an efficient algorithm for mining frequent closed Itemsets,” Proc. Of the 2000 ACM_SIGMOD Int. Conf. on Management of Data, May, Dallas, Texas, USA.

Tan Pang-ning, Steinbach Micheal ,Kumar Vipin. (2006). "Introduction to data mining" Addison-Wesley.

Ivancsy, R. and Vajk, I. (2004). “An analysis of association rules mining algorithms,” CD-ROM Proc. Of Fourth International ICSC Symposium on Enginerring of Intelligent Systems (EIS 2004), February 29-March 2, Island of Madeira, Portugal.

R. Ivancsy and Vajk, I. (2006). “Time- and Memory-Efficient Frequent Itemset Discovering Algorithm for Association Rule Mining,” International Journal of Computer Application in technology, Vol. 27, No. 4, pp. 270-279.

Park,J.S., Chen, M. and Yu,P.S. (1995). 'An effective hash based algorithm for mining association rules', Proc. Of the 1995 ACM SIGMOD Int. Conf. on Management of Data, San Jose, California, USA, pp.175-186.

Brin,S., Motwani, R., Ullman,J.D. and Tsur, S. (1997). "Dynamic itemset counting and implication rules for market basket data", Proc. Of the 1997 ACM SIGMOD Int. Conf. on Management of Data, Tucson, Arizona, United States, pp.255-264.

Han, J., Pei, J. and Yin, Y. (2004). "Mining frequent patterns without candidate generation: a frequent-pattern tree approach', Data Mining and Knowledge Discovery, Journal of Kluwer Academic Publishers, Vol. 8, pp.53-87.

# ABSTRACT

## A Study on Efficient Association Rule Mining Algorithm for Personalized Web service

Choi, Ji-Hoon
Major in Multimedia Engineering
Dept. of Multimedia Engineering
The Graduate School
Hansung University

Mining the user's web access pattern requires a lot of computing time and hardware resource, since web log is quite a large data. Discovering association rule is computationally expensive and has high I/O cost. Here we propose an efficient algorithm which reduces search space and the experimental result shows the reduction of the execution time and memory usage. The proposed algorithm is applied to the web log of the university homepage and we find 3- or 4-frequent itemset. Discovered frequent itemset can be used for the design of the web page by presenting the group of the frequent itemset and personalized web page can be serviced based on the association rule. The efficiency of the algorithm increases as the k increases for finding k-frequent itemset