논문 2007-44CI-1-10

점진적인 웹 마이닝을 위한 효율적인 후보패턴 저장 트리구조 및 알고리즘

(An Efficient Candidate Pattern Tree Structure and Algorithm for Incremental Web Mining)

강 희 성*, 박 병 준**

(HeeSeong Kang and ByungJoon Park)

요 약

네트워크 환경의 발전으로 다양한 콘텐츠와 다수의 사용자를 가지는 포탈, 대형 사이트 들이 증가 하게 되었고, 이러한 포털, 대형 사이트의 증가로 인해 서로 다른 성향을 띈 다수의 사용자들이 다양한 경로로 사이트를 이용하게 되었다. 이렇게 다양해진 경로 중에 빈번하게 발생하는 경로를 빈발패턴이라고 하며, 빈발패턴이 될 가능성이 있는 패턴을 후보패턴이라고 한다. 이러한 후보패턴들을 이용해 트리구조를 생성, 추가, 삭제, 검색 하는 것은 마이닝 과정 중의 한 부분으로서, 트리구조 및 알고리즘에 따라 마이닝의 성능에 영향을 미치게 된다. 본 논문에서는 이러한 후보패턴들을 이용하는 마이닝의 성능 향상을 위해 하나의 노드가 다수의 링크를 가지는 기존의 트리 구조와는 달리 하나의 노드가 3개의 링크를 가지고 있고, 각 노드들이 계층 구조로 이루어져 있어, 기존의 트리보다 정형화된 구조와 향상된 성능을 보이는 3차원 트리구조 및 생성, 추가, 삭제, 검색 알고리즘을 제안한다.

Abstract

Recent advances in the internet infrastructure have resulted in a large number of huge Web sites and portals worldwide. These Web sites are being visited by various types of users in many different ways. Among all the web page access sequences from different users, some of them occur so frequently that may need an attention from those who are interested. We call them frequent access patterns and access sequences that can be frequent the candidate patterns. Since these candidate patterns play an important role in the incremental Web mining, it is important to efficiently generate, add, delete, and search for them. This thesis presents a novel tree structure that can efficiently store the candidate patterns and a related set of algorithms for generating the tree structure, adding new patterns, deleting unnecessary patterns, and searching for the needed ones. The proposed tree structure has a kind of the 3 dimensional link structure and its nodes are layered

Keywords: Tree, Incremental, Candidate, Pattern, Web Mining

I. 서 론

웹 환경은 네트워크 환경의 발전으로 인해 더욱더 다양한 콘텐츠를 서비스 할 수 있게 되었고, 이런 다양한 콘텐츠를 다수의 사용자가 이용하게 됨으로 인해, 다양한 접근 패턴이 나타나게 되었다. 이렇게 발생한 대량의 접근 패턴으로 인해 접근 패턴을 이용한 마이닝 시에 소요되는 비용이 증가 하게 되었다.

발생한 모든 접근 패턴을 마이닝 하는 것 보다 특정 빈도 이상 발생하는 패턴만으로 후보 패턴을 구성하고, 그 후보 패턴으로 마이닝을 하는 것이 전체적인 비용을 줄 일 수 있다. 또한, 특정 빈도 이상 발생한 패턴만으로 구성된 후보 패턴을 통해 마이닝을 한 후에 새로운 접근 패턴이 발생하게 되면, 모든 접근 패턴에서 다시 후보 패턴을 추출하고 다시 마이닝 하는 방법보다는, 새로 발생한 접근 패턴을 기존에 구성한 후보 패턴에 추가하여 마이닝 하는 방법이 전체적인 비용을 줄 일

* 학생회원, ** 정회원, 광운대학교 컴퓨터과학과
(Dept. of Computer Science, Kwangwoon University)

※ 이 논문은 2005년도 광운대학교 교내학술연구비 지원에 의해 연구되었음.
접수일자: 2006년 10월9일, 수정완료일: 2007년1월4일

수 있다. 이런 방법을 점진적 증가 마이닝(Incremental Mining)[1][6]이라고 하며, 기존의 논문들은 이를 위해 다양한 트리구조를 제안하고 있다. FS-Tree 알고리즘 [1], FAP-Tree 알고리즘[10] 등 다양한 알고리즘이 존재하며, 이들은 모두 트리 구조로 이루어져 있으나, 겉으로 보이는 모양만 트리 구조일 뿐 일반적으로 사용하는 크기 비교 등을 통한 트리 검색이 아닌 순차적인 방법을 통해 검색을 하게 되어있다. 로그를 이용해 후보 패턴을 추출하고 이 후보 패턴을 트리구조에 저장하는 것은 검색과 삽입의 반복 작업으로 이루어지게 되는데 기존의 트리형태는 후보 패턴이 증가할수록 새로운 로그를 기존의 후보 패턴에 추가하는데 걸리는 시간이 증가하게 된다. 이로 인해서 후보 패턴을 구성하는데 걸리는 시간이 오래 걸리게 되며, 후보 패턴을 구성하는 시간이 오래 걸리면 마이닝 시간도 오래 걸리게 되는 것이다. 기존의 후보 패턴 추출 방법과 동일한 방법을 사용하고, 또한 기존의 마이닝 방법과 동일한 마이닝 방법을 사용 할 경우 추출한 후보 패턴을 구축하는 시간을 단축한다면 전체적인 마이닝 시간을 단축하는 효과를 보게 될 것이다.

본 논문에서는 후보 패턴의 구축 시간을 단축할 수 있는 노드와 계층 트리들로 구성된 3DFS-Tree(3D Frequent Sequence Tree)를 제안하며, 다른 트리들과의 성능 예측 및 비교 실험 결과를 도출할 것이다. 성능 비교 실험은 순수하게 트리의 성능을 테스트하기 위해 후보패턴이 되는 조건에 부합하는 시퀀스들을 가지고 생성, 점진적 증가, 삭제, 등에 대해 실행시간이 얼마나 차이가 나는지에 대해 실험 한다

Ⅱ. 관련 연구

본 장에서는 마이닝과 시퀀스 그리고 후보패턴에 대해서 설명하며, 후보 패턴을 이용해 트리를 구성하는 FS-Tree, FAP-Tree 알고리즘에 대해서 설명한다.

1. 마이닝

웹 환경은 다양한 유저의 요구를 만족시키기 위하여 변화 하고 있지만 유저수와 콘텐츠의 증가로 인해 다양한 유저들의 요구를 따라가기에는 무리가 생기게 되었다. 웹 사용 마이닝은 웹 로그를 분석하여 다양한 유저들의 요구를 파악하고 그에 맞게 웹 환경에 변화를 줄 수 있게 하기 위한 마이닝 기법이다.

2. 빈발 시퀀스 (Frequent Sequences)

시퀀스는 웹상의 페이지를 사용자가 방문한 순서를 의미하며, 사용자가 a, d, c의 순서로 페이지를 이동 했다면 < a, b, c> 로 표현된다.

이 시퀀스들의 빈도수를 이용해 어떤 접근 패턴이 빈발한지 분석해야 하는데, 이때 이 시퀀스들의 빈도수를 지지도(Support Count)라고 한다.

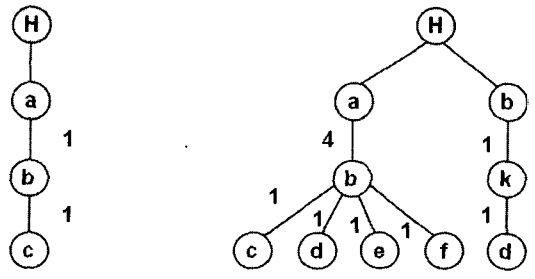
마이닝 시스템은 2개의 인수를 가지게 되는데 시스템에서 주어지는 값과 사용자로 인해 주어지는 값을 가지게 된다. 시스템 인수는 최소한의 후보 패턴을 추출하기 위한 지지도를 나타내며, 사용자 인수는 사용자가 요구하는 후보 패턴을 추출하기 위한 값을 나타낸다.

시스템 인수를 기준으로 후보가 될 수 있는 가능성이 있는 패턴들을 추출하고, 이 패턴들 중에 후보패턴을 추출하는 것은 사용자 인수를 기준으로 하며, 사용자 인수를 통해 추출된 후보 패턴을 빈발 시퀀스 라고 한다.

3. FS-Tree

가. FS-Tree 알고리즘

FS-Tree 알고리즘은 후보패턴의 조건을 만족하는 시퀀스를 이용해 트리를 구성하는 알고리즘으로 시퀀스 내의 아이템 순서대로 하위노드로 연결되는 구조를 가지고 있다. 예를 들어 시퀀스 <a, b, c>가 입력된다면 그림1(a) 처럼 트리가 생성되고, 여기서 추가로 시퀀스 <a, b, d>, <a, b, e>, <a, b, f>, <b, k, d> 등이 발생하면 그림1(b) 같은 트리가 생성 된다. 중복되는 시퀀스 <a, b>는 한번만 표현되고 중복횟수만큼 카운트가 증가 하는 구조를 가지고 있다.

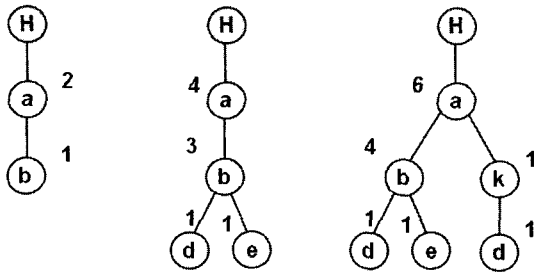


(a) (b)

그림 1. FS-Tree
Fig. 1. FS-Tree.

나. FAP-Tree 알고리즘

FAP-Tree 알고리즘은 FS-Tree와 같은 방법으로 트



(a) (b) (c)

그림 2. FAP-Tree
Fig. 2. FAP-Tree.

리를 구성하지만 상위 노드에 중복된 ID가 존재 하면 해당 노드로 이동하는 순환이라는 방법을 사용한다.

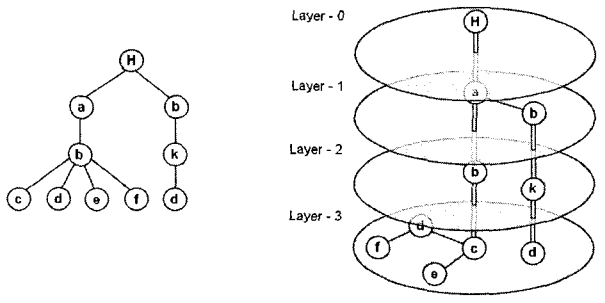
III. 3D Frequent Sequence Tree 알고리즘

1. 3DFS-Tree 알고리즘의 소개 및 구성

가. 3DFS-Tree 알고리즘의 소개

3DFS-Tree는 관련연구에서 설명한 FS-Tree, FAP-Tree와 같이 입력된 시퀀스들을 저장하는 하는 트리구조를 갖고 있지만, 3DFS-Tree는 시퀀스의 추가, 삭제, 검색에 좀 더 효율적인 트리구조를 가지고 있다.

시퀀스 <a, b, c>, <a, b, d>, <a, b, e>, <a, b, f>, <b, k, d>가 입력됐을 때, FS-Tree, FAP-Tree, 3DFS-Tree는 그림3과 같이 나타내어 진다.



(a) FS-Tree, FAP-Tree (b) 3DFS-Tree

그림 3. Fs-Tree, FAP-Tree와 3DFS-Tree의 비교
Fig. 3. Comparison of Fs-Tree, FAP-Tree and 3DFS-Tree.

(1) 계층 트리(Layer Tree)

계층 트리는 계층이 같은 노드로 구성된 트리를 의미하는 것으로, 각각의 계층 트리는 각 노드들의 계층이 모두 같아야 하고, 모든 노드들은 노드 링크로 연결되어야 한다. 3DFS-Tree는 이렇게 구성된 계층 트리들이 계층 링크로 연결되어 있는 구조로 이루어져 있으며, 각각의 계층 트리는 상·하위 계층링크를 통해 계층별 이동이 가능하지만, 같은 계층 끼리는 이동이 불가능한 구조를 띄게 된다. 각각의 계층 트리는 최적화된 검색 및 삽입을 위해 완전한 이진트리 구조로 이루어져 있으며 본 논문에서는 삽입, 삭제 오버헤드가 적은 비트트리를 사용해 구현 되었다.

(2) 형제 트리(Siblings Tree)

계층 트리를 구성하는 트리 구조인 형제트리는 이진트리 형태를 띄고 있으나, 일반적인 이진트리에서 나타나는 좌우 노드의 불균형으로 인한 성능 저하를 없앤 형태이다. 형제트리의 삽입은 각 비트를 기준으로 비트의 값이 0이면 왼쪽, 1이면 오른쪽이라는 기준을 잡고 트리를 구성해 나가는 것이 형제트리의 생성 방식이며, 만약 첫 번째 비트 값으로 오른쪽이나 왼쪽으로 이동했는데 해당 위치에 노드가 존재 한다면 두 번째 비트 값으로 또 다시 오른쪽이나 왼쪽으로 이동 한다 이런 방식으로 단순 비트 값 비교만으로 좌우로 배치되므로 인해 밸런싱에 적은 비용을 소모하고 거의 완전한 이진트리를 구성할 수 있게 된다.

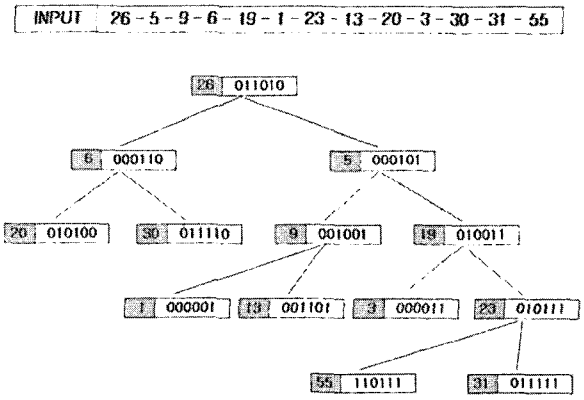


그림 4. 형제트리 생성
Fig. 4. Construction Siblings Tree.

2. 3DFS-Tree 알고리즘

가. 3DFS-Tree 알고리즘

본 논문에서 제안하는 3DFS-Tree 알고리즘은 입력된 시퀀스들의 추가, 삭제, 검색 알고리즘으로, 시퀀스를 추가하는 Insert3DFSTree(), 노드를 찾는 FindNode(), 계층 트리를 추가하는 InsertLayerTree(), 노드를 추가, 삭제 하는 InsertNode()와 DeleteNode()로 구성된다.

(1) Insert3DFSTree()

```
# 의사코드 내의 Sequence p-P에서 p는 시퀀스의 첫 번째 아이템을 P는 서브시퀀스를 나타내며, 시퀀스의 아이템은 INT형으로 되어 있다.
Function Insert3DFSTree(Node N, Sequence p-P)
    Create Node D
    D = FindLayerTree(N.LayerDownLink, p)
    if(D is Not Null)
        N = D
        N.Count = N.Count + 1
    else
        N = InsertLayerTree(N, p)
    end if
    if(P is not Null) Insert3DFSTree(N, P)
End Function
```

그림 5. Insert3DFSTree()
Fig. 5. Insert3DFSTree().

(2) FindLayerTree()

```
Node Function FindLayerTree(Node N, INT p)
    Create INT I, INT Data, Node P
    if(N is Null) return Null
    Data = p
    P = N
    While(P is Not Null)
        if(P.Data is p) return P
        I = GetFirstBit(Data)   // 첫번째 비트를 반환해주는 함수
        if(I is True)
            P = P.RightNode
        else
            P = P.LeftNode
        end if
    End While
End Function
```

그림 6. FindLayerTree()
Fig. 6. FindLayerTree().

(3) InsertLayerTree()

```
Node Function InsertLayerTree(Node N, INT p)
    Create Node T
    T = MakeNode(p)
    if(N.LayerDownLink is Null)
        N.LayerDownLink = T
    else
        InsertNode(N.LayerDownLink, T)
    end if
    return T
End Function
```

그림 7. InsertLayerTree()
Fig. 7. InsertLayerTree().

(4) DeleteNode()

```
Function DeleteNode(Node H, Node N)
    Create INT Data, INT I, Node P, Node T
    Data = N.Data
    P = H
    While(P is Not Null)
        if(P.Data is Data)
            if(P.Count is 1)
                Delete P
                Return
            else
                P.Count = P.Count - 1
            end if
        end if
        I = GetFirstBit(Data)   // 첫번째 비트를 반환해주는 함수
        if(I is True)
            P = P.RightNode
        else
            P = P.LeftNode
        end if
    end while
end Function
```

그림 8. DeleteNode()
Fig. 8. DeleteNode().

나. 3DFS-Tree 생성 과정 예시

3DFS-Tree 알고리즘의 실 예를 들어보면 발생한 시퀀스 < a, b, c > 라면 그림 9(a)와 같이 추가 되고 이어서 < a, k, j >가 발생하면 그림 9(b)과 같이 된다.

두 번째 시퀀스도 a로 시작하므로 같은 데이터가 존재하게 되며 이때는 카운트가 증가되어 a가 2가 된다. k는 a의 하위 계층 링크에 연결된 계층 트리에 없고 계층이 2이므로 b의 오른쪽 또는 왼쪽 링크에 연결 되어 b와 k는 하나의 계층 트리를 이루게 된다. k 다음에 발생한 j는 k의 하위계층 트리에 추가 된다.

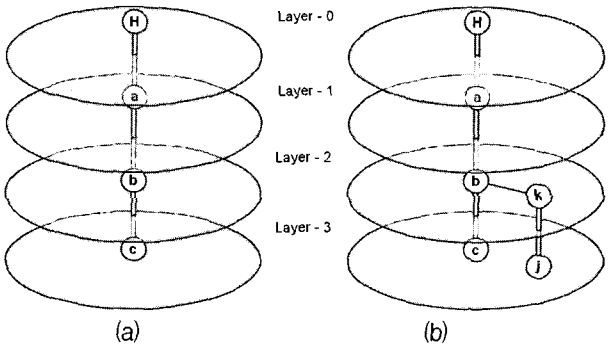


그림 9. 3DFS-Tree 생성
Fig. 9. Construction 3DFS-Tree.

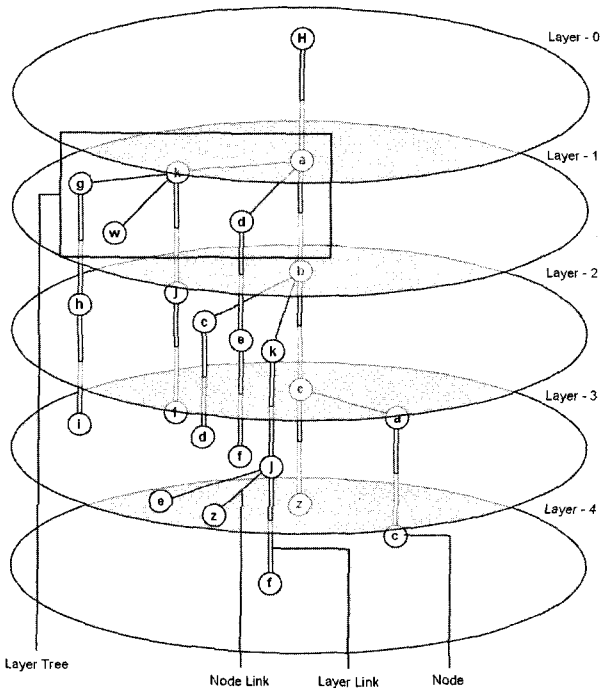


그림 10. 3DFS-Tree 모식도
Fig. 10. 3DFS-Tree Diagram.

< a, b, c, z >, < a, k, j, f>, < d, e, f >, < g, h, l >, < a, c, d >, < a, c, z >, < a, k, e >, < k, j, f >, < a, b, a, c >, < a, k, z > 등의 좀 더 많은 양의 시퀀스를 발생시키면 그림10과 같이 된다.*

IV. 실 험

1. 실험 데이터 및 실험 방법

가. 실험 데이터

본 논문의 실험은 FS-Tree, FAP-Tree, 3DFS-Tree에 시퀀스를 추가, 검색, 삭제하여 각 트리 알고리즘의 성능을 실험 하는 것이다. 이때 트리에 추가 되는 시퀀스는 후보패턴의 조건에 부합하는 시퀀스들이며 FS-Tree나 FAP-Tree, 3DFS-Tree 모두 다음과 같은 유사한 과정을 거쳐 트리에 추가될 시퀀스를 결정하게 된다.

후보패턴의 지지도가 2이상일 경우에 시퀀스 <c, a, b>, <a, b, c>, <b, c, a, d> 가 발생하면, c-a 2번, a-b 2번, b-c 2번, a-d 1번 발생하게 된다. 이때 지지도가 2 이상일 경우를 각 시퀀스별로 확인하게 되는데 <c, a, b>인 경우 <c, a>, <a, b>, <a, b, c>인 경우 <a, b>, <b, c>가 모두 2 이상이므로 <c, a, b>, <a, b, c> 모두 트리에 추가 된다. 하지만 <b, c, a, d>인 경우에는 <b, c>와 <c, a>는 2이상이지만 <a, d>인 경우는 1이므로 <b, c, a>만 트리에 추가 되고 <a, d>는 추가 되지 않는다. 이렇게 각 시퀀스의 서브시퀀스가 지지도 이상일 경우에만 시퀀스를 트리에 추가 한다.

나. 실험 데이터의 구성 요소

본 논문의 실험에서는 후보 패턴의 조건을 만족하는 시퀀스들이 존재 한다고 가정하였으며, 랜덤하게 임의의 시퀀스를 생성하여 실험 하였다. 한 데이터는 3개의 구성요소를 가지며 각 구성요소는 다음과 같다.

1) 페이지 총수 - 발생한 페이지 개수를 의미하며 발생한 시퀀스가 <a, b, d, a> 라면 페이지 총수는 4이다. 페이지 총수는 전체 데이터의 크기를 의미한다. 사용자 수 증감에 따른 성능 예측에 이용 된다.

2) 서로 다른 페이지 - 서로 다른 페이지가 많을수록 사용자는 더욱더 다양한 액세스 패턴을 보이게 된다. 사이트의 규모가 커질수록 서로 다른 페이지가 늘어나게 됨으로 사이트 규모에 따른 성능 예측에 이용 된다.

3) 시퀀스 길이 - 하나의 시퀀스가 얼마만큼의 길이를 가지는지를 나타내며, 평균 길이를 나타낸다. 빈번하게 발생하는 다수의 페이지 링크가 많을 경우의 성능을 비교 판단하게 된다.

다. 실험 방법

이렇게 주어진 3가지 구성요소를 통해 다양한 데이터 셋을 생성하여 실험하였다. No는 1~10까지 페이지 총수는 No가 증가할때마다 100만개씩 증가하였다. 사용한 데이터는 표1에 나타난 값을 모두 조합 하였다.

표 1. 실험 데이터
Table 1. Experimental Data.

No	페이지 총수	서로 다른 페이지	시퀀스 길이
1	1,000,000	50, 500, 1,000	4, 12, 20
…	…		
10	10,000,000		

라. 비교 알고리즘과 항목

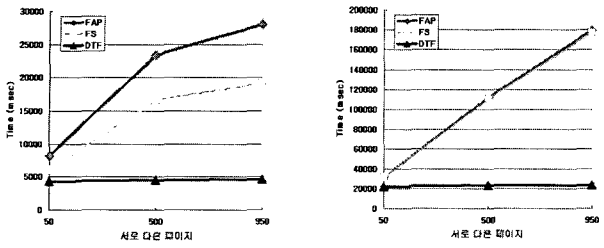
FS-Tree, FAP-Tree, 3DFS-Tree의 생성, 점진적 증가, 삭제 성능을 실험하며, 이 3가지 실험을 할 경우 페이지 총수, 서로 다른 페이지 수, 시퀀스 길이에 따른 성능 평가 및 성능 예측이 가능해진다.

* 그림11은 3DFS-Tree구조를 알아보기 쉽게 하기 위해 카운트를 표시하지 않았음.

2. 실험 내용

가. 생 성

1) 페이지 총수 100만개와 1000만개 일때, 시퀀스 길이 평균 4 일 경우 서로 다른 페이지 수에 따른 성능 비교



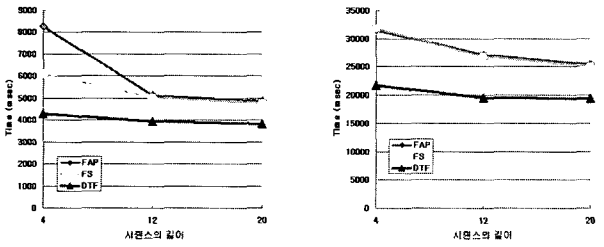
(a) 시퀀스 총 100만개 (b) 시퀀스 총 1000만개

그림 11. 서로 다른 페이지 수에 따른 성능 비교
Fig. 11. compare with performance of distinct pages.

이 실험은 비슷한 로그수를 보이는 두 개의 사이트가 있을 때 사이트 규모에 따른 성능을 테스트 하는 실험이다. 이 실험을 통해 FS와 FAP는 사이트 규모가 커질수록 성능이 저하되며, 이는 순차검색이므로 자식노드가 증가 하는 만큼 검색시간이 더 걸리기 때문이다.

3DFS-Tree는 내부적으로 계층 트리를 이용한 이진검색을 하기 때문에 사이트 규모에 크게 영향을 받지 않고 고른 성능을 나타내게 된다.

2) 페이지 총수 100만개와 1000만개, 서로 다른 페이지 수 50개 일 때 시퀀스 길이에 따른 성능 비교



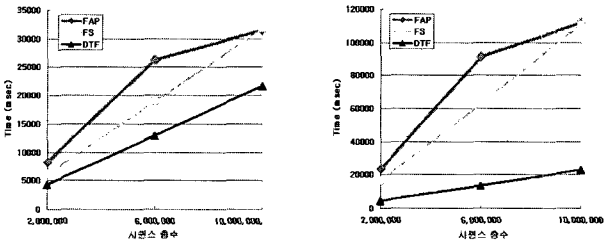
(a) 시퀀스 총 100만개 (b) 시퀀스 총 1000만개

그림 12. 시퀀스의 길이에 따른 성능 비교
Fig. 12 compare with performance of sequence length.

시퀀스의 길이는 사이트에 오랫동안 머물며 서핑 할 경우에 증가 하게 되며 이는 사이트에 다양하고 많은 정보가 있을 경우 발생 하므로, 이 실험은 다양하고 많은 정보가 있는 사이트일 경우, 사용자가 서핑을 오래 해야 하는 특성을 가진 사이트의 성능을 알 수 있게 된다.

3DFS-Tree는 시퀀스의 길이에 상관없이 고른 성능 분포를 나타내며, 시퀀스의 길이가 12이상이 되면, FAP와 FS 또한 안정적인 성능 분포를 보여 주고 있다. 또한, 두 알고리즘 모두 시퀀스의 길이가 증가할수록 좋은 성능을 나타낸다.

3) 서로 다른 페이지 50개, 500개, 시퀀스 길이 평균 4로 고정 일 때 페이지 총수에 따른 성능 비교



(a) 서로 다른 페이지 50개 (b) 서로 다른 페이지 500개

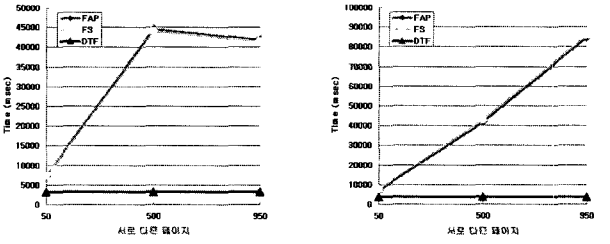
그림 13. 페이지 총수에 따른 성능 비교
Fig. 13. compare with performance of Page Total.

페이지 총수의 변화에 따른 실험은 사이트의 사용자 수에 따른 성능을 비교하기 위한 것이다. FAP와 FS는 페이지 총수의 증가량에 거의 비례하며, 3DFS- Tree는 페이지 총수가 증가해도 안정적인 성능을 보인다.

나. 점진적 마이닝 (Incremental Mining)

이 실험은 1000만개의 시퀀스로 트리를 생성한 후에 이 트리에 새로운 시퀀스가 추가될 경우의 성능을 측정한 것으로, 점진적 증가 성능을 측정할 수 있다. 실제 실험은 1000만개의 시퀀스로 생성한 트리에 200만개의 시퀀스를 추가하여 실험했으며, 결과는 생성과 동일하다.

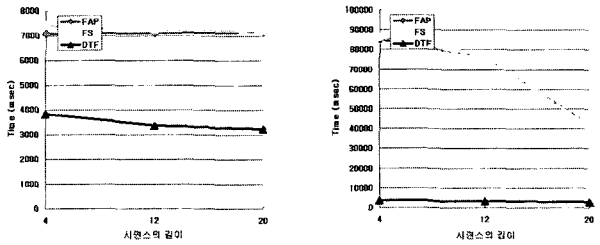
1) 시퀀스 1000만개 생성 후 200만개 증가 시 서로 다른 페이지 수에 따른 성능 비교



(a) 시퀀스 길이 평균 4 (b) 시퀀스 길이 평균 20

그림 14. 서로 다른 페이지 수에 따른 성능 비교
Fig. 14. compare with performance of distinct pages.

2) 시퀀스 1000만개 생성 후 200만개 증가 시 시퀀스 길이에 따른 성능 비교



(a) 서로 다른 페이지 50개 (b) 서로 다른 페이지 500개

그림 15. 시퀀스의 길이에 따른 성능 비교

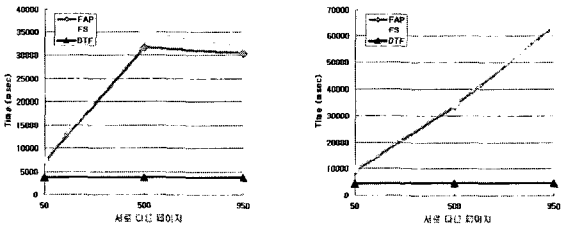
Fig. 15. compare with performance of sequence length.

다. 삭 제

이 실험은 트리를 생성한 후에 이 트리에 시퀀스가 삭제 될 경우의 성능을 측정한 것으로, 삭제 시 걸리는 시간으로 성능을 측정했다.

실험결과의 해석은 생성과 동일하다.

1) 페이지 총수 1000만개로 생성 후, 200만개의 페이지 삭제시 서로 다른 페이지 수에 따른 성능 비교

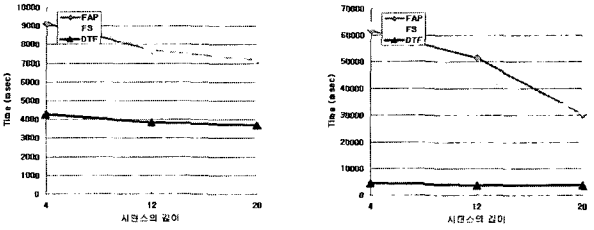


(a) 시퀀스 길이 평균 4 (b) 시퀀스 길이 평균 20

그림 16. 서로 다른 페이지 수에 따른 성능 비교

Fig. 16. compare with performance of distinct pages.

2) 페이지 총수 1000만 새성 후, 200만개의 페이지 삭제시 시퀀스 길이에 따른 성능 비교



(a) 서로 다른 페이지 50개 (b) 서로 다른 페이지 500개

그림 17. 시퀀스 길이에 따른 성능 비교

Fig. 17. compare with performance of sequence length.

3. 결과 분석

FAP-Tree는 페이지 총수가 적거나, 서로 다른 페이지수가 적을 경우에는 낮은 성능을 보이지만, 페이지 총수와 서로 다른 페이지수가 증가 할수록 FS-Tree와 거의 비슷한 성능을 보여준다. FAP-Tree는 메모리를 적게 차지하기 때문에 대형 웹 사이트에 적용하면 보다 적은 메모리로 좋은 효율을 얻을 수 있을 것으로 추정할 수 있다.

3DFS-Tree는 모든 실험에서 고른 성능을 나타내는데. 좀 더 자세히 살펴보면 시퀀스의 길이나 서로 다른 페이지 수의 변화에 거의 상관없이 고른 성능을 나타냈고, 페이지 총수의 증가에 대해서는 낮은 기울기를 보이고 있다. 이를 통해 사이트의 구조와 접근 패턴의 영향을 거의 받지 않고 고른 성능을 나타냄을 알 수 있다.

삭제 실험에서도 3DFS-Tree는 주어진 조건에 관계없이 일정한 성능을 보였다.

FAP-Tree는 순환하는 구조로 인해 성능이 좀 떨어지는 경향이 있지만 시퀀스의 총수, 시퀀스의 길이, 서로 다른 페이지 수가 증가할수록 순환의 비용이 노드수가 늘어났을 경우 필요한 비용보다 줄어들기 때문에 FS-Tree와 비슷하거나 좋은 성능을 보인다.

3DFS-Tree는 사용자수가 늘어날수록 성능이 저하되긴 하지만 사용자 수의 증가 비율에 비해 낮은 기울기로 성능이 저하되며, 또한 FS-Tree, FAP-Tree 보다 높은 성능을 보이며 증가 비율 또한 더 낮은 기울기를 보여 주고 있고, 시퀀스의 길이, 서로 다른 페이지 수에는 거의 영향을 받지 않고 안정적인 성능을 나타낸다.

표 2. FS-Tree, FAP-Tree, 3DFS-Tree 실험 결과 비교

Table 2. Comparison of experimental evaluationl FS-Tree, FAP-Tree and 3DFS-Tree.

	FS-Tree	FAP-Tree	3DFS-Tree
사이트 규모	규모가 클수록 성능 저하	규모가 클수록 성능 저하	규모에 관계없이 높은 성능 유지
다수의 사용자	증가 비율과 비슷한 비율로 성능 저하	증가 비율과 비슷한 비율로 성능 저하	증가 비율 보다 낮은 비율로 성능 저하
장시간 사용자	많을수록 성능 증가	많을수록 성능 증가	관계없이 높은 성능 유지

V. 결 론

본 논문에서는, 기존의 마이닝에서 사용하는 시퀀스

의 트리구조를 개선하여 더 결과적으로 마이닝 속도를 향상 시키는 트리구조를 제시하였다. 실험을 통해 알 수 있듯이 기존의 트리구조에 비해 데이터의 크기에 관계없이 안정적이고 좋은 성능을 보여주었다. 성능 면에서는 기존의 트리에 비해 높은 성능을 보이지만, 기존의 트리구조에 비해 가독성이 떨어지고 노드 구성이 복잡하여 전반적인 알고리즘의 이해와 계층 노드와 일반 노드의 구분을 명확히 구분할 줄 알아야만 하고, 기존의 트리 구조와 거의 비슷하거나 좀 더 많은 메모리를 차지함으로 인해서 메모리 효율성은 개선된 사항이 없었다.

본 논문에서는 기존의 시퀀스 패턴 발견 규칙과 마이닝 방법을 그대로 사용 한다는 전제하에 논문이 작성되었다. 이로 인해 마이닝 방법에 따라 노드의 구성요소에 약간의 변화가 주어지는데 본 논문은 새로운 트리구조를 제안하는 것이므로 전체 구조에 큰 영향을 미치지 않는다. 하지만 기존의 마이닝 방법이 본 논문에 최적화 되진 않았기 때문에 향 후 에는 본 논문에서 제안하는 3DFS-Tree에 맞는 마이닝 기법을 제안 하여, 마이닝에 더욱더 효율적인 3DFS-Tree 구조로 보완이 이루어 져야 할 것이다.

참 고 문 헌

[1] Maged El-Sayed, Carolina Ruiz, and Elke A.Rundensteiner, "FS-Miner : Efficient and Incremental Mining of Frequent Sequence Patterns in Web logs", WIDM, pp.128-135, 2004.

[2] Wolfgang Gaul, Lars Schmidt-Thieme, "Mining Web navigation path fragments", Institut fur Entscheidungstheorie und Unternehm ens for schung. pp.1-6, 2000.

[3] R. Cooley, B. Mobasher, and J. Srivastava, "Web Mining : Information and Pattern Discovery on the World Wide Web", Proc. of the 9th IEEE International Conference on Tools with Artificial Intelligence, pp.558-567, Nov 1997.

[4] Ramakrishnan Srikant, and Rakesh Agrawal, "Mining Sequential Patterns : Generalizations and Performance Improvements", IBM Almaden Research Center 650 Harry Road, San Jose, CA 95120, pp.3-17, 1996.

[5] Apache Korea Group, http://www.apache.kr.net

[6] S.Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas, "Incremental and Interactive Sequence Mining", CIKM, pp.251-258, 1999.

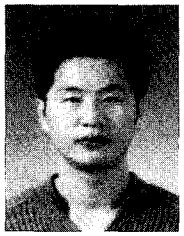
[7] Mathias Gery, Hatem Haddad, "Evaluation of Web Usage Mining approaches for user's next request prediction", WIDM, pp.75-76, 2003.

[8] Jaideep Srivastava, R. Cooley, M. Deshpande, P-T. Tan, "Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data", SIGKDD Explorations, Volume 1, Issue 2, pp.12-23, Jan 2000.

[9] 코리아인터넷 마케팅센터, http://www.webpro.co.kr

[10] Xidong Wang, Yiming Ouyang, Xugang Hu, Yan Zhang, "Discovery of User Frequent Access patterns on Web usage Mining", IEEE, pp. 765-769, 2003.

저 자 소 개



강 희 성(학생회원)
2004년 광운대학교 컴퓨터과학과 학사 졸업.
2006년 광운대학교 컴퓨터과학과 석사 졸업.
2006년~현재 광운대학교 컴퓨터과학과 박사과정
<주관심분야 : Algorithm, Artificial Intelligence, Web Mining, Neural Network>

박 병 준(정회원)
1984년 서울대학교 컴퓨터공학과 학사 졸업
1988년 University of Minnesota, Computer Science 석사 졸업
1989년~1996년 US Army Construction Engineering Research Lab. Researcher.
1997년 University of Illinois at Urbana-Champaign, Computer Science 박사 졸업
1998년~2000년 SPSS Inc, Senior Researcher
2000년~현재 광운대학교 컴퓨터과학과 교수
<주관심분야 : Artificial Intelligence, Data/web Mining, Knowledge-Based System>