웹 데이터에서의 사용자 탐색 패턴 발견 및 추천

구 흠 모*: 양 재 영 · 홍 광 희 · 최 중 민

Discovery and Recommendation of User Search Patterns from Web Data

Heummo Gu*: Jaeyoung Yang · Kwanghee Hong · Joongmin Choi

요약

웹 사용 마이닝은 데이터마이닝을 바탕으로 사용자의 로그 파일 정보를 이용하여 웹이 이용되는 패턴을 발견한다. 이를 이용하여 웹을 개선하여 사용자들이 보다 빨리 원하는 내용을 검색할 수 있도록 할 수 있으며 시스템 관리자에게는 효율적인 웹 구조를 위한 정보를 제공할 수 있다. 웹 사용 마이닝에서 사용하는 데이터는 정형화되어 있지 않으며 웹 사용 패턴을 분석하는데 방해가 되는 잡음 데이터까지 포함하고 있다. 이것은 기존에 개발된 여러 데이터마이닝 기법을 적용하는데 어려움으로 작용한다. 이러한 어려움을 해결하기 위해 본 논문에서는 새로운 방법을 도입한 SPMiner을 제안한다. SPMiner는 웹의 구조를 이용하여 로그 파일의 전처리 과정을 줄이며 사용자의 탐색 패턴 분석을 효율적으로 수행 할 수 있는 시스템이다. SPMiner는 WebTree 에이전트를 이용하여 웹 사이트 구조를 분석하여 WebTree를 생성하고 사용자 로그 파일을 분석하여 각 웹 페이지의 사용빈도에 대한 정보를 추출한다. WebTree와 로그 파일에서 추출된 웹 페이지에 대한 정보는 SPMiner에 의해 패턴을 분석할 때 이용될 수 있는 형태인 WebTree+로 병합된다. WebTree+는 패턴 발견을 쉽게 해주며 사용자에게 추천할 정보나 웹 페이지를 능동적으로 추천할 수 있게 만들어 준다.

Key words : Web Usage Mining, Sequential Mining, User Search Pattern Discovery, Recommendation

1. 서론

웹은 오늘날 정보를 보급하기 위한 가장 인기있고 대중적인 매체이다. 웹의 많은 사용으로 인하여 우리는 정보의 바다에서 살아가고 있다. 많은 사용자들이 웹을 이용하여 자신이 원하는 정보를 찾고 있지만 웹이 급속도로 증가하면서 정보의 양은 더욱 더 늘어나, 사용자가 정말 필요한 정보를 찾기 위하여 많은 시간과 노력을 기울여야 한다. 이런 상황 때문에 현재는 정보의 바다에서 헤엄치며 정보 과부하(information overload)에 직면해 있다[1]. 이러한 어려움을 해결하기 위하여 사용자가 원하는 정보를 쉽고 빠르게 찾기 위하여 자동화된 툴(tool)이 필요하게 되었고, 관리자들은 웹 사이트의 효율적인 운영과 더 나은 서비스를 위하여 사용자들의 사용 패턴을 분석하는 툴들을 필요로 하게 되었다. 웹 사이트를 분석함으로 우리가 얻을 수 있는 이점으로는 사용자 편의를 위한 웹 사이트의 디자인, 시스템의 성능분석, 웹 문서를 쉽게 적응하기 위한 동기 부여 등이 있다.

웹 접근 로그 데이터(web access log data)는 사이트에서 웹 문서를 사용자가 탐색한 것을 기록한 정보를 말한다. 이렇게 저장되는 정보의 양은 매우 빠르게 증가하고 있으며, 로그 데이터 자체로는 유용한 정보를 찾기에 어려움이 많이 있다. 이러한 로그 데이터를 분석하기 위하여 데이터 마이닝(data mining)기술들을 기초로 하는 웹 마이닝(web mining), 그 중 웹 사용 마이닝(web usage mining)을 간구하게 되었다.

데이터 마이닝은 대용량의 데이터베이스로부터 기존에 알려지지 않은, 즉 단순한 질의어로 추출할 수 없는 형태의 유용한 정보를 찾아내는 것이라 말할 수 있으며[2], 웹 마이닝은 데이터 마이닝 기술을 이용하여 웹으로부터 유용한 정보를 분석하거나 발견하는 것이다[3].

웹 마이닝은 잘 정형화된 데이터 베이스가 아닌 잡음이 많은 데이터, 정형화되어 있지 않은 데이터로부터 유용한 정보를 찾아야 하므로 기존의 여러 데이터 마이닝 기술을 적용하기에는 다음과 같은 몇 가지 문제점들을 가지고 있다. 첫째 잡음이 많은 웹의 특성에 부적합하다. 둘째, 매우 방대한 후보 아이

* 한양대학교 컴퓨터공학과

템셋을 발생시킨다. 셋째, 반복적인 로그 데이터의 로딩으로 인하여 시간과 비용을 낭비한다. 넷째, 데이터 마이닝 기술을 사용하여 사용자들에게 정보를 온라인상에서 능동적으로 제공해주기가 어렵다.

따라서 본 논문에서는 이러한 문제를 해결 하기 위하여 새로운 방법을 도입한 SPMiner를 제안한다. SPMiner는 웹의 구조를 이용하여 로그 파일의 전처리 과정을 줄이며 사용자의 탐색 패턴 분석을 효율적으로 수행 할 수 있는 시스템이다. SPMiner는 WebTree 에이전트를 이용하여 웹 사이트 구조를 분석하여 WebTree를 생성하고 사용자 로그 파일을 분석하여 각 웹 페이지의 사용빈도에 대한 정보를 추출한다. WebTree와 추출된 웹 페이지에 대한 정보는 SPMiner에 의해 패턴을 분석할 때 이용될 수 있는 형태인 WebTree+로 병합된다. WebTree+는 패턴 발견을 쉽게 해주며 사용자에게 추천할 정보나 웹 페이지를 능동적으로 추천할 수 있게 만들어 준다.

본 논문의 순서는 2장에서 데이터 마이닝과 웹 마이닝의 정의와 기존 데이터 마이닝의 패턴 분석 방법들과 문제점에 대하여 알아본다. 3장에서는 SPMiner의 전체적인 시스템 구조를 알아보며 4장에서는 웹 사이트로부터 WebTree 생성을 위한 WebTree 에이전트에 관한 내용과 알고리즘을 알아본다. 5장에서는 로그 데이터의 정제와 빈도수 측정 방법을 알아보고, WebTree와 로그 파일에서 추출된 웹 페이지에 대한 정보를 병합해주는 WebTree+에 대하여 알아본다. 6장에서는 WebTree+로부터 패턴 발견 방법과 발견된 정보를 온라인 상에서 사용자에게 추천하는 방법을 알아본다. 7장에서는 SPMiner를 구현해보고 얻어진 결과를 알아보며, 마지막으로 8장에서는 결론 및 향후 연구 방향에 대하여 알아보도록 한다.

2. 관련연구

이 장에서는 최근 많이 사용되는 웹 마이닝 기술을 데이터 마이닝과 함께 알아보고 다음으로 이 논문과 관련된 몇몇 패턴 분석 알고리즘들의 특징과 이들의 문제점에 대하여 알아보기로 한다.

2.1 웹 마이닝

웹 마이닝(web mining)이란 데이터 마이닝 기술을 이용하여 웹으로부터 유용한 정보를 분석하거나 발견하는 것이다[3]. [4,5,6]에서는 웹으로부터 나타날 수 있는 데이터의 형태를 분류할 때 각각 다른 형태로 분류하고 있지만, 일반적으로 웹의 여러 가지 데이터에 따라 다음과 같이 세 가지로 나누어 볼 수 있다[7].

- 웹 내용 마이닝(web content mining)
- 웹 구조 마이닝(web structure mining)
- 웹 사용 마이닝(web usage mining)

2.1.1 웹 내용 마이닝

웹 내용 마이닝(web content mining)은 실제 웹 사이트를 구성하고 있는 데이터로부터 의미 있는 내용을 발견하는 것이다[4,8]. 이는 일종의 정보 추출(information extraction)이라고도 할 수 있다. 기본적으로 웹 내용은 텍스트, 이미지, 오디오, 비디오, 하이퍼텍스트(hypertext) 등 다양한 형태의 데이터를 포함한다. 하지만 웹 내용 마이닝은 텍스트와 하이퍼텍스트 위주로 이루어 지고 있다.

2.1.2 웹 구조 마이닝

웹 구조 마이닝(web structure mining)은 웹의 하이퍼링크 구조를 기반으로 웹의 구조적인 모델을 만드는 것이다. 이 모델은 웹 페이지를 분류하는데 이용할 수 있으며, 다른 웹 사이트 사이의 유사성을 파악하기 위하여 사용할 수 있다. 웹 구조 마이닝 자체 보다는 다른 마이닝과 함께 연계하여 사용하는 경우가 많다.

2.1.3 웹 사용 마이닝

웹 사용 마이닝(web usage mining)은 웹 사용자의 사용 패턴을 분석하는 것이다. 웹 내용 마이닝과 웹 구조 마이닝은 웹 실제의 데이터를 사용하지만 웹 사용 마이닝은 사용자와 웹의 상호작용에 의하여 만들어진 데이터를 이용한다. 이 데이터는 웹 서버 접근 로그, 프락시 서버 로그, 브라우저 로그, 사용자 프로파일, 쿠키 등을 나타낸다. 이렇게 발견된 패턴들은 전문가의 분석을 통하여 사용자에게 더욱 편리한 서비스를 위하여 웹 페이지를 재구성하거나, 웹 서버 로드 밸런스, 사용자별 맞춤형 웹 페이지 구성, 관심 있는 자료에 대한 추천 등에 이용된다.

2.2 패턴 분석 알고리즘

패턴 분석에는 서로간의 연관관계를 찾아내는 연관규칙(association rules)과, 순차적으로 일어난 패턴을 찾아내는 순차 패턴(sequential pattern)으로 크게 나누어 볼 수 있다. 연관규칙에서 대표적인 알고리즘에는 Apriori 알고리즘과 FP-growth 알고리즘 등이 있다. 순차 패턴 분석에서 대표적인 알고리즘으로는 GSP 알고리즘, PrefixSpan 알고리즘, SPADE 알고리즘 등이 있다.

패턴 분석 알고리즘에서 패턴을 분석할 때 찾아낸 패턴을 평가하는 방법에는 크게 지지도(support)와 신뢰도(confidence)의 두 가지가 있다. 지지도는 전체 트랜잭션(transaction)에서 아이템 A와 B를 동시에 포함하는 트랜잭션의 비율을 말하며 신뢰도는 아이템 A를 만족한 트랜잭션 중에서 아이템 B를 포함하는 트랜잭션의 비율을 말한다. 여기에서 트랜잭션이란 한번에 일어날수 있는 아이템이나 이벤트의 집합을 말한다. 지지도와 신뢰도를 식으로 나타내면 다음과 같다.

$$\text{지지도}(Support) = \frac{P(TransactionA \cap TransactionB)}{P(\Omega)}$$

$$\text{신뢰도}(Confidence) = \frac{P(TransactionA \cap TransactionB)}{P(TransactionA)}$$

Ω는 전체 트랜잭션을 나타내며 $P(TransactionA)$는 트랜잭션 A가 나타날 확률을 말한다. 웹 사용 마이닝에서 지지도는 문서의 빈도수(frequency)로 많이 나타낸다. 패턴 분석 알고리즘에 대하여 좀더 자세히 알아보자.

2.2.1 Apriori 알고리즘

Apriori 알고리즘은 패턴 분석에서 가장 일반적이며, 간결한 알고리즘이다. 이 알고리즘은 1994년 Agrawal과 Srikant이 제안하였다[9].

기본 아이디어는 가장 작은 크기의 아이템셋(itemset)에서 집합의 크기를 하나씩 증가시켜 각 아이템셋에 대한 지지도를 찾고 최소 지지도 이상의 아이템셋을 찾아 나가는 것이다. 여기에서 아이템셋의 크기를 점점 크게 할 수 있는 이유는 어떤 아이템셋이 자주 발생하는 패턴이라면 그의 하위 아이템셋 또한 항상 자주 발생하는 패턴이라는 원리를 이용하였다.

[그림 1] Apriori 알고리즘

```
Ck: Candidate itemset of size k
Lk : frequent itemset of size k
L1 = {frequent items};
for (k = 1; Lk !=∅; k++) do begin
    Ck+1 = candidates generated from Lk;
    for each transaction t in database do
            increment the count of all candidates in Ck+1
            that are contained in t
    Lk+1  = candidates in Ck+1 with min_support
    end
return ∪k Lk;
```

[그림 1]의 Apriori 알고리즘은 두 단계로 구성되어 있다. 첫 단계에서는 새로운 후보 집합 중에서 최소 지지도 설정값에 따라 빈도수가 높은 항목의 아이템셋들을 찾아낸다. 두 번째 단계에서는 집합들로부터 신뢰도를 만족하는 연관규칙을 모두 찾아낸다. 이 두 단계를 반복하게 된다.

2.2.2 FP-growth 알고리즘

FP-growth 알고리즘은 기존 Apriori 알고리즘의 문제점인 많은 스캔과 후보 아이템 생성을 줄이기 위하여 FP-tree를 이용하였다. 이 알고리즘은 2000년 Jiawei Han, Jian Pei, Yiwen Yin이 제안하였다[10]. 이 알고리즘의 FP-tree를 생성하는 알고리즘과, 생성된 FP-tree에서 패턴을 찾아내는 알고리즘으로 나누어진다.

FP-tree를 생성하는 단계에서는 먼저 입력 트랜잭션 단위로 저장된 데이터로부터 크기가 1인 최소 지지도 이상의 아이템을 모두 구한다. 다음으로 빈도(frequency)가 높은 아이템 순으로 내림차순 정렬된 헤더 테이블(header table)을 만든다. 이는 패턴을 찾는 단계에서 탐색을 편하게 하기 위해서이다. 데이터를 다시 스캔하면서 트리를 생성한다. 최초 빈(null) 트리에서 트랜잭션 별로 경로(path)를 만들게 된다. 트리에서 아이템이 있다면, 빈도를 증가시키고 없다면 새로운 아이템 이름을 가지는 노드를 추가시킨다. 이렇게 하여 FP-tree를 생성하게 되며, 여기에서 빈도 높은 패턴을 찾는다.

2.2.3 GSP 알고리즘

GSP 알고리즘은 **G**eneralized **S**equential **P**attern의 약자이며 가장 일반적인 순차 패턴 마이닝 알고리즘이다. 이 알고리즘은 1996년 Agrawal과 Srikant이 제안하였다[11].

Apriori 알고리즘에 아이템들의 순서를 고려하여 패턴을 찾는 방법이다. 후보를 생성할 때 Apriori는 아이템셋들의 집합을 만들지만 GSP는 순차 패턴들의 집합을 만든다. 예를 들어 a, b, c 세개의 아이템이 있다라고 가정하자. 이때, 크기가 2인 후보를 만들어 보면 Apriori는 {a, b}, {a, c}, {b, c} 3개의 후보 아이템셋을 만들지만, GSP는 <a→b>, <b→a>, <a→c>, <c→a>, <b→c>, <c→b> 6개의 후보 패턴을 만들게 된다.

2.2.4 PrefixSpan 알고리즘

PrefixSpan 알고리즘은 **Prefix**-Projected **S**equential **patter**n mining의 약자이며, 2001년 Jian Pei, Jiawei Han 등이 제안하였다[12].

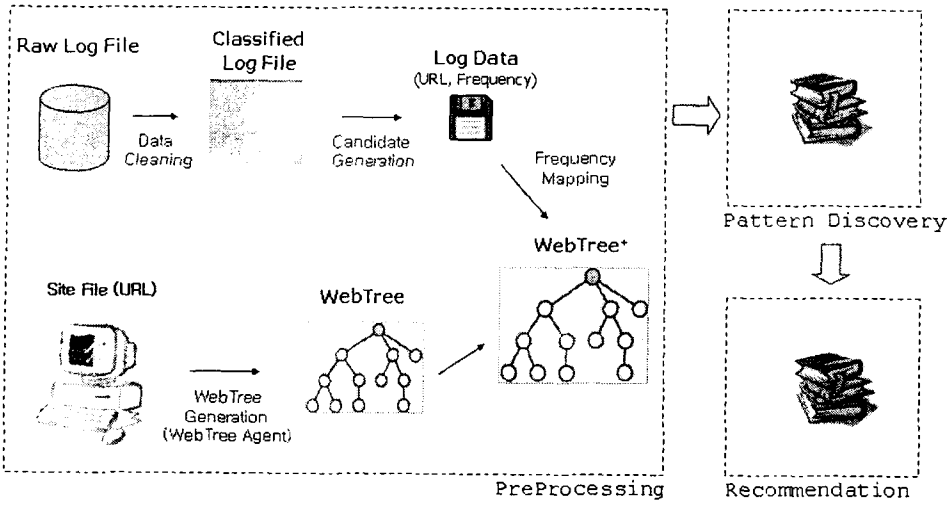
이 알고리즘에 대하여 알아보면, 처음 데이터베이스를 스캔 하여 길이가 1인 순차 패턴을 모두 찾는다. 이때 FP-growth 알고리즘과 같이 각 패턴의 빈도수를 구하고 미리 정해둔 최소 지지도 이상의 패턴들을 찾아낸다. 각각의 패턴을 접두사(prefix)로 하는 서브셋(subsets)을 만들어 각각 분리한다. 예를 들어, <a→b→c→d→e>로 이루어진 순차 패턴이 있을 때 a를 접두로 하는 하위 순차 패턴은 <b→c→d→e>이며, c를 접두로 하는 하위 순차 패턴은 <d→e>이다. 이 과정을 각각 분리된 서브셋에 대하여 계속 반복하게 된다. 이렇게 하여 생성된 접두사들이 유용한 패턴들이 된다.

2.2.5 SPADE 알고리즘.

SPADE 알고리즘은 **S**equential **Pa**ttern **D**iscovery using **E**quivalence classes의 약자이며, 1998년 Zaki, M.J에 의하여 제안되었다[13].

이 알고리즘은 기존의 알고리즘과는 달리 데이터베이스를 트랜잭션 단위가 아닌 아이템 단위의 데이터베이스로 변형하여 사용한다. 이후의 알고리즘은 GSP 알고리즘과 비슷한 형태 형태를 보인다. 다른 점은, 패턴의 길이를 확장시킬 때 모든 트랜잭션

[그림 2] SPMiner 시스템 구조



을 스캔 하는 것이 아닌 아이템 별로 분류된 데이터베이스를 스캔 하는 것이다. 이것은 연관규칙이 아닌 순차 패턴의 후보 패턴들을 찾아내기 위한 방법으로 사용된다.

2.3 기존 알고리즘의 문제점

앞에서 살펴본 여러 알고리즘에서 나타나는 문제점을 정리해보면 다음과 같다.

(1) 웹의 특성에 부적합: 기존 알고리즘들은 대부분 데이터 마이닝을 위한 알고리즘들이다. 웹 데이터들은 데이터 베이스를 목적으로 만들어 지지 않았으며, 웹 특성상 로그 데이터내의 잡음을 많이 가지게 된다. 또한 웹 구조를 반영하지 못해 기존 알고리즘에 의하여 나타난 유용한 패턴들이 실제 웹 구조와 다른 형태로 나타날 수 있다. 이로 인해 패턴의 믿음이 떨어지고 실제 적용 후 잘못된 결과를 유발할 수 있다.

(2) 매우 방대한 후보 아이템셋 집합들을 발생: Apriori, GSP, SPADE 알고리즘들은 후보 아이템셋을 생성하여 패턴을 찾는다. 예를 들어, 크기 1인 아이템셋을 10^4 개 가지는 집합이 있을 때, 크기가 2인 후보 아이템셋은 10^7 보다 더 많은 개수를 가지는 집합을 만들게 된다. 또한 크기가 100인 아이템셋을 만들려고 할 때 만약 크기가 1인 아이템셋이 100개 있다면, 결과적으로 만들게 되는 전체 후보 아이템셋의 수는 $\sum_{i=1}^{100}\binom{100}{i} = 2^{100} - 1 \approx 10^{30}$ 이상의 후보 아이템셋을 생성해야 한다[12].

(3) 후보집합의 생성으로 인한 잦은 로그 파일의 로딩(loading): 한번 후보 아이템셋 집합을 생성할 때마다 한번의 로딩이 필요하다. 예를 들어, 만약 크기가 10인 패턴을 찾는다면, 매번 후보집합들을 생성할 때 로그 파일의 로딩이 필요하므로 최소한 10번의 로그 파일 로딩이 필요하다.

(4) 온라인(on-line)이 아닌 오프라인(off-line)에서 이루어짐: 데이터 마이닝의 경우 온라인 상태가 아닌 오프라인 상태에서 마이닝을 적용하여 유용한 정보를 찾은 후 웹 사이트에 적용한다. 이는 데이터의 변화가 많이 일어나는 온라인 상태에서 능동적으로 최근 자료를 적용하지 못하며, 또한 온라인 상에서 바로 결과를 돌려주지 못한다. 예를 들어, 쇼핑몰에서 웹 마이닝을 적용한 결과 어떤 상품이 인기가 높다는 결과를 알았다. 이를 쇼핑몰 고객들에게 추천을 하였지만 상품의 생산중단, 계절상품 등의 문제로 인하여 상품판매가 중지된 경우에는 마이닝 결과가 아무런 도움을 주지 못한다.

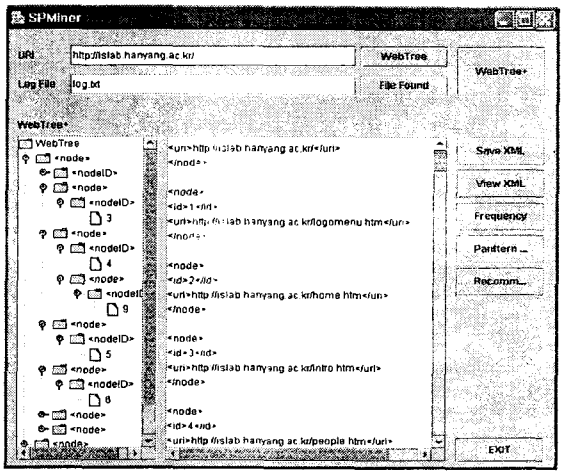
3. SPMiner

본 논문에서 제안하는 SPMiner는 웹의 사용 기록인 웹 로그 파일과 하이퍼링크로 연결된 웹 구조를 이용한다. 기존 시스템들은 웹 사용정보인 사용자 로그 파일의 형식을 미리 정하여 입력 데이터로 사용했다. 고정된 형식을 가지는 로그파일은 전처리 과정에서 트랜잭션 별로 웹 페이지를 분류하는 과정에서 로그 파일을 이루고 있는 여러 필드(field)의 정보를 손쉽게 이용하기 위해서이다. 일반적으로 로그 파일마다 다른 형태를 가지므로, 여러 형태의 사용자 로그 파일 형식을 받아 수행하기에는 문제점이 있다. 본 시스템에서는 사용자 로그 파일에서 단지 참조 페이지만 받아 수행함으로 별도의 전처리 단계가 필요 하지 않는다.

[그림 2]는 시스템의 구조를 설명하고 있으며, 전처리(preprocessing), 패턴 발견(pattern discovery), 추천(recommendation) 세 부분으로 이루어져 있다. 입력 데이터는 웹 사이트의 URL과 웹 서버에 저장된 로그 파일이다. URL을 입력 받아 하이퍼링크를 이용하여 WebTree Agent가 WebTree를 만든다. 그리고 서버 로그 파일은 데이터 정제 과정을 수행한 후 각 참조

문서에 대하여 사용 빈도를 계산하게 된다. 다음으로는 WebTree+는 앞의 두 과정을 병합하는 과정으로 WebTree에 문서의 빈도수를 매핑하는 과정이다. 마지막으로 이렇게 만들어진 WebTree+를 이용하여 탐색 패턴 발견을 하며 사용자에게 흥미 있는 패턴을 추천하는 단계이다. [그림 3]은 SPMiner의 인터페이스이다.

[그림 3] SPMiner 인터페이스



4. WebTree 에이전트

웹 사이트의 구조는 페이지 사이의 하이퍼링크(hyperlink)에 의해 나타난다. 이 링크 구조는 웹 문서들간의 연결관계를 알려주며, 잠재적으로 웹 사용 패턴을 파악하는데 유용한 자료가 된다.

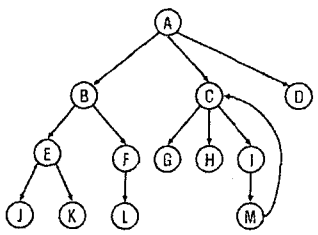
4.1 웹 사이트 & 그래프

웹 사이트 구조를 나타내기 위해서 그래프를 많이 사용한다. 아래와 같이 웹 사이트의 문서와 문서내의 하이퍼링크는 그래프의 노드(node)와 에지(edge)에 해당한다.

$$Web(documents, links) \rightarrow Graph(nodes, edges)$$

웹 사이트는 방향성과 사이클을 가지며, 가중치가 없는 그래프로 나타낼 수 있다. 아래 그림은 웹 사이트 구조를 그래프로 나타낸 한 예이다.

[그림 4] 웹 사이트 구조의 예



d, C를 문서와 문서내의 자식 링크 집합이라 할 때 웹 사이트 $WT = [<(d_1, C_1), (d_2, C_2), ..., (d_n, C_n)>]$로 나타낼 수 있다. 여기에서 C는 문서의 자식 링크들의 집합인 $C = \{c_1, ..., c_k\}$을 나타낸다. 다음은 [그림 4]를 표현한 것이다.

$$WT = [<(A,\{B,C,D\}),$$
$$(B,\{E,F\}),(C,\{G,H,I\}),(D,\{\})$$
$$(E,\{J,K\}),(F,\{L\}),(G,\{\}),(H,\{\}),(I,\{M\})$$
$$(J,\{\}),(K,\{\}),(L,\{\}),M\{C\}>]$$

4.2 WebTree 에이전트

에이전트(Agent)란 사전적 의미로 '대리인'이란 뜻을 가지며 사용자를 대신하여 특정 작업을 자동으로 수행하는 프로그램을 말한다. 웹 상에서 에이전트 기술을 많이 사용하고 있으며, 사용자에게 필요한 정보를 대신하여 자동으로 찾아주는 역할을 한다.

본 논문에서의 WebTree 에이전트는 주어진 도메인에 대하여 하위 하이퍼링크 구조를 생성하는 에이전트 이다. WebTree 에이전트가 가져야 할 요구사항 정리해 보면 다음과 같다.

- 주어진 도메인외의 연결된 하이퍼링크들은 탐색을 하지 않는다. 예를 들어 작업도메인이 http://cse.hanyang.ac.kr/일때 에이전트가 페이지를 검색 중에 http://eecs.hanyang.ac.kr/로 연결된 링크가 있다면 이 도메인에 대해서는 탐색하지 않는다. 이는 도메인 이외의 링크를 제외하고, 에이전트가 무한정 웹을 떠돌아 다니는 것을 방지한다.
- 한번 탐색된 페이지는 다음 탐색에서 제외된다. 위 [그림 4]의 그래프를 보면 C, I, M 페이지가 사이클로 형성되어 있다. 에이전트가 자식 페이지를 탐색할 때 C→I, I→M을 탐색하게 되고 또 M은 C를 탐색하므로 무한루프에 빠진다. 이런 이유로 에이전트가 도메인 내에서 무한 루프에 빠지는 것을 방지한다.

4.3 알고리즘

[그림 5]는 WebTree 에이전트 알고리즘이다. WT는 (d, C)의 집합이며 여기에서 d는 문서의 URL, C는 자식 페이지 링크들의 집합을 의미한다.

처음 WT는 초기 도메인의 URL만 가지지만 이후 페이지의 자식 링크들을 추가한다. WT의 문서들을 하나씩 로딩 후 문서내의 하이퍼링크를 C에 추가한다. 만약 하이퍼링크가 없다면, C와 WT에 추가되지 않으며 탐색할 문서의 수가 줄어들게 된다. A는 이미 검색된 문서를 가지는 집합으로 문서를 로딩 후 A에 속한 문서들은 C와 WT에 추가할 때 제거된다. 함수 makeAccumulatedTree()는 현재 문서 d_k와 자식 링크 집합인 C_k를 입력 받아 트리를 만든다. 최종적으로 WebTree 에이전트는 완성된 트리 T를 돌려주게 된다.

[그림 5] WebTree 에이전트 알고리즘

```
// URL : Domain Address
// WT = [<(d1, C1), (d2, C2), ..., (dn, Cn)>]
// di = 문서 페이지의 URL
// Ci = { cij | cij는 문서 di에 대한 자식 페이지 링크}
// A ={ am | am는 이미 검색된 상위 문서 }

T = ∅

WebTreeAgent{
    WT.d1 ← URL
    k = 1
    While(WT.dk ≠ ∅){
        A ← dk를 추가
        L ← dk에 해당하는 문서를 로딩 후 하이퍼텍스트
            링크만 추출
        Ck = {c | c ∈ L, c ∉ A}
        makeAccumulatedTree(T, dk, Ck )
            For(Ck의 각 c에 대하여){
                WT ← c를 WT의 새로운 d로 추가
            }
            k++
        }
    return T
}
```

4.4 WebTree 에이전트에 의하여 생성된 데이터 표현

WebTree 에이전트에 의해서 생성된 데이터는 XML로 표현한다. WebTree는 확장성과 유연성을 가져야 하며 XML은 이러한 요구사항에 적합한 메타언어이다. WebTree 에이전트는 두 종류의 XML파일을 생성한다. 첫번째 파일은 웹 사이트의 고유번호를 유지하기 위해 시스템에 부여한 ID와 URL을 담고 있으며 두번째 파일은 웹 사이트 구조가 표현되어 있다. 이렇게 두 파일로 분리한 이유는 웹 트리를 URL 자체로 검색하는 것 보다 각 문서에 ID를 부여하여 검색하므로 프로그램의 계산량을 줄일 수 있기 때문이다. 프로그램 수행이 끝난 후 결과를 보여줄 때 각 ID에 해당하는 URL를 보여주면 된다.

[그림 6] ID와 URL을 표현

```
<URL_ID>
    <node>
      <nodeID>ID Number</nodeID>
      <nodeName>URL</nodeName>
    </node>
    ...
</URL_ID>
```

[그림 7] WebTree의 계층적 표현

```
<WebTree>
  <node>
    <nodeID>ID number</nodeID>
      <node>
        <nodeID>ID number</nodeID>
        </node>
          ...
  </node>
  ...
</WebTree>
```

[그림 8] WebTree+의 계층적 표현

```
<WebTree>
  <node>
    <nodeID>ID number</nodeID>
      <node>
        <nodeID>ID number</nodeID>
        <frequency>integer number</frequency>
        <support>real number</support>
      </node>
          ...
  </node>
  ...
</WebTree>
```

[그림 6]과 [그림 7]은 두 XML 파일의 구조를 보여준다. [그림 8]은 다음 절의 WebTree+ 적용 후 생성되는 XML 파일 구조이다. [그림 7]에서 각 문서(노드)에 빈도수와 지지도를 추가한 형태이다.

5. WebTree+

WebTree와 로그 파일에서 추출된 웹 페이지에 대한 정보를 이용하여 패턴 분석과 추천이 가능한 WebTree+를 생성하게 된다. 앞 절에서 WebTree+에 필요한 WebTree를 생성하였다. 이번 절에서는 사용자 로그 파일을 분석하여 각 웹 페이지의 사용빈도에 대한 정보를 추출하여 WebTree와 병합하는 과정을 알아 본다.

5.1 로그 데이터 정제

웹 서버 로그 데이터는 사용자가 웹을 서핑 할 때 사용자의 행위를 기록하는 데이터이며, 가장 일반적인 형태는 CLF, ECLF 형태를 따르는 웹 서버 로그 파일이다. 로그 데이터는 사용자에게 필요한 정보가 아닌 단지 서버측면에서 서버의 모든 행동들을 시간을 기준으로 저장하게 된다. 이런 로그 데이터 자체로는 우리에게 큰 의미를 주지 못하며, 사용자의 탐색 패턴 정보를 찾기란 힘들다. 따라서 마이닝 기술을 적용하여 패턴을 찾기 전에 전처리 과정을 통해서 웹 로그 파일을 변환해야 한다.

Cooley[5]는 전처리 과정을 크게 데이터 정제(data cleaning), 사용자/세션 식별(user/session identification), 페이지 뷰 식별(page view identification), 경로 완성(path completion)의 네단계로 나누어 수행하였다. 여기서 경로 완성은 부가적으로 수행할 수 있는 단계이다. 이 전처리 과정을 수행하면 시간을 기준으로 하여 저장된 기존 로그 데이터는 사용자의 트랜잭션을 기준으로 변환된다. 하지만 본 논문에서는 전처리 과정의 데이터 정제과정만을 수행한다. 이는 기존 트랜잭션 형태의 데이터를 필요로 하는 것이 아니며, 웹 사이트의 구조와 사용자의 웹 탐색 페이지만을 이용하기 때문이다.

<표 1>은 그래픽 및 영상등과 관련된 데이터는 제외한 CLF형태의 웹 서버 로그 데이터의 한 예이다. 표와 같이 로그 데이터에는 많은 정보들이 포함되어 있다. 이러한 정보에는 사용자를 구분할 수 있는 IP 주소, 접속시간 및 날짜, 각 문서의 URL, 프

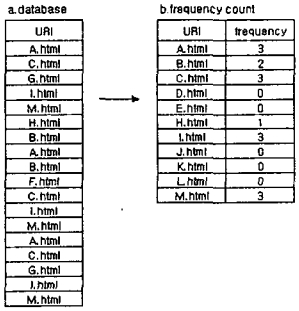
로토콜(protocol), 문서의 상태 및 크기 등이 있다. 일반적으로, 그래픽, 영상 파일 요청에 대해서는 제외한다. 이것은 단지 파일이름의 확장자를 검색하여 제거할 수 있다. 이렇게 불필요한 부분을 제거하면 결과적으로 URL 부분만 남게 된다.

<표 1> 웹 서버 로그 데이터의 예

IP Address	Auth ID	Date / Time	Method/URI/Protocol	Status	Size
61.97.17.223	- -	[19/Sep/2002:00:27:43 +0900]	"GET /A.html HTTP/1.1"	200	3205
61.97.17.223	- -	[19/Sep/2002:00:27:43 +0900]	"GET /C.html HTTP/1.1"	200	12154
61.97.17.223	- -	[19/Sep/2002:00:27:50 +0900]	"GET /G.html HTTP/1.1"	200	13458
61.97.17.223	- -	[19/Sep/2002:00:27:53 +0900]	"GET /I.html HTTP/1.1"	200	24857
61.97.17.223	- -	[19/Sep/2002:00:27:54 +0900]	"GET /M.html HTTP/1.1"	200	10808
61.97.17.223	- -	[19/Sep/2002:00:32:29 +0900]	"GET /H.html HTTP/1.1"	200	20171
61.97.17.223	- -	[19/Sep/2002:00:32:34 +0900]	"GET /B.html HTTP/1.1"	200	38207
203.200.5.189	- -	[19/Sep/2002:01:09:57 +0900]	"GET /A.html HTTP/1.1"	200	3205
203.200.5.189	- -	[19/Sep/2002:01:09:57 +0900]	"GET /B.html HTTP/1.1"	200	12154
203.200.5.189	- -	[19/Sep/2002:01:10:40 +0900]	"GET /F.html HTTP/1.1"	200	13458
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /C.html HTTP/1.1"	200	3205
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /I.html HTTP/1.1"	200	12154
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /M.html HTTP/1.1"	200	24857
203.252.115.231	- -	[19/Sep/2002:22:06:17 +0900]	"GET /A.html HTTP/1.1"	200	3205
203.252.115.231	- -	[19/Sep/2002:22:06:18 +0900]	"GET /C.html HTTP/1.1"	200	12154
203.252.115.231	- -	[19/Sep/2002:22:06:26 +0900]	"GET /G.html HTTP/1.1"	200	13458
203.252.115.231	- -	[19/Sep/2002:22:06:35 +0900]	"GET /I.html HTTP/1.1"	200	16948
203.252.115.231	- -	[19/Sep/2002:22:06:41 +0900]	"GET /M.html HTTP/1.1"	200	71897

다음으로 로그 데이터 정제 과정을 통하여 얻어진 데이터는 각 문서에 대하여 빈도수(frequency)를 계산한다. 빈도수는 문서의 지지도 값을 측정하기 위한 기초 데이터가 된다.

[그림 9] 로그 데이터 정제후 각 문서의 빈도수



[그림 9]은 로그 파일로부터 우리가 필요한 URL 부분을 추출하고 각 URL에 대한 빈도수를 보여주고 있다.

5.2 문서의 빈도수 수정 및 지지도 측정

앞 절에서 각 페이지에 대하여 빈도수를 구하였다. 이 값에 빈도수를 다시 수정하는 것은 이전 방법들의 전처리 과정에서의 잡음 제거를 대신하기 위해서이다. 웹 사이트에서 각 문서를 그래프의 노드라고 생각하자. 루트(root) 노드와, 리프(leaf) 노드를 제외한 모든 노드들은 부모(parent)노드와 자식(child) 노드를 가진다. [그림 4]에서 현재 문서가 C라고 할 때 C의 부모 문서는 A이고, 자식 문서는 G, H, I이다.

웹 사이트를 방문시 정상적인 경우 가장 상위 문서부터 접속을 하게 되고 이후 하위 문서들을 탐색해 나간다. 따라서 상위 문서는 하위 문서 보다 빈도수가 높게 나온다. $freq_{Node}$ 를 Node의 빈도수, *CurrentNode* 를 현재 노드, *ParentNode* 를 부모 노드, *ChildNode* 를 자식 노드라 할 때 다음과 같은 식이 성립한다.

$$freq_{ParentNode} \geq freq_{CurrentNode} \geq \sum freq_{ChildNode}$$

현재 문서는 부모 문서(상위 문서)의 빈도수 이하이고 자식 문서의 빈도수 합 이상이다. 하지만 로그 데이터의 특성상 다음과 같이 여러 가지가 나타난다.

(1) $freq_{ParentNode} \geq freq_{CurrentNode} \geq \sum freq_{ChildNode}$

(2) $freq_{ParentNode} \geq freq_{CurrentNode} \leq \sum freq_{ChildNode}$

(3) $freq_{ParentNode} \leq freq_{CurrentNode} \geq \sum freq_{ChildNode}$

(4) $freq_{ParentNode} \leq freq_{CurrentNode} \leq \sum freq_{ChildNode}$

(1)은 가장 일반적인 형태를 나타낸다. (2)는 현재 문서의 빈도수가 부모 문서의 빈도수 보다는 작고, 또한 자식 문서들의 빈도수의 합보다도 작은 경우이다. (3)은 (2)와 반대되는 경우이며, (4)는 (1)과 반대되는 경우이다. (1)을 제외한 나머지의 경우는 로그 데이터의 잡음으로 나타나는 경우이다. 웹 로그 데이터에 잡음을 유발하는 경우를 정리해 보면 다음과 같다.

- 네트워크의 오류: 서버와 사용자는 네트워크로 연결되어 원하는 정보를 주고 받는다. 이때 네트워크의 불안정으로 데이터가 사라지는 경우가 발생하며, 로그 데이터에 기록을 못한 경우이다.
- 웹 브라우저의 캐시: 웹 브라우저의 캐시는 사용자들이 탐색한 문서들을 임시 저장한다 보통 사용자들은 이전 문서로 돌아가기 위하여 웹 브라우저의 백(back) 버턴을 누른다 이때 캐시는 이전에 본 문서에 대하여 웹 서버의 접근 없이 바로 보여줌으로 웹 페이지의 로딩 시간을 줄여준다. 그리고 탐색 정보는 서버의 로그 데이터에 기록되지 않는다.
- 관심 있는 문서의 즐겨 찾기: 사용자는 관심 있는 문서를 편리하게 찾기 위하여 웹 브라우저의 즐겨 찾기 기능을 많이 사용한다. 따라서 사용자는 특정 웹 문서를 바로 탐색할 수 있으며 이러한 경우 상위 문서들은 로그 데이터에 기록되지 않는다.

위의 여러가지 경우로 인해 상위 문서가 하위 문서의 빈도수보다 낮은 경우가 발생한다. 이 문제를 해결하기 다음의 규칙을 이용하여 빈도수를 업데이트 한다.

(1) 만약 $freq_{CurrentNode} \geq \sum freq_{ChildNode}$ 이면 $freq_{CurrentNode} = freq_{CurrentNode}$ 이다. 이는 현재 문서가 자식 문서들의 빈도수의 합보다 큰 경우이며 현재 문서의 빈도수는 그대로 유지한다.

(2) 만약 $freq_{CurrentNode} < \sum freq_{ChildNode}$ 이면 $freq_{CurrentNode} = \sum freq_{ChildNode}$ 이다. 현재 문서가 자식

문서들의 빈도수의 합보다 작은 경우이며 현재 문서는 하위 문서들의 빈도수의 합으로 수정한다.

(2)의 경우 현재 문서는 단지 이동을 목적으로 하는 문서에 가깝다. 그러므로 현재문서의 빈도수를 자식 문서들의 합과 가장 가까운 값으로 수정한다. 이것은 미리 설정해둔 최소 지지도 이상의 탐색 패턴 들을 찾을 때, 이 문서들의 선택을 최소한으로 줄일 수 있다.

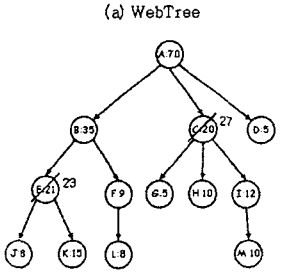
빈도수는 웹 사이트나 로그 파일 크기에 따라 달라진다. 이것은 최소 빈도수를 정하기 어렵게 만든다. 이 문제점을 해결하기 위하여 각 페이지의 빈도수를 표준화(normalization) 한다. 웹 사이트의 URL에 해당하는 루트 페이지는 다른 하위 문서보다 큰 빈도수를 가진다. 이를 기준으로 각 문서의 빈도수를 표준화된 값으로 바꿀 수 있다. 다음 식은 각 문서의 빈도수를 표준화하는 식이다.

$$Support(Node) = freq_{Node} / freq_{Root}$$

지지도는 가장 상위 문서인 루트 페이지의 빈도수로 나누어 줌으로 표준화된 값을 가진다.

[그림 10]는 로그 데이터로부터 얻어진 각 문서의 빈도수를 업데이트하고 지지도를 구하는 과정을 보여주고 있다. C와 E문서의 빈도수가 자식 문서들의 빈도수의 합 보다 작은 경우로 각각 27과 23으로 수정되었다.

[그림 10] 빈도수 업데이트 및 지지도 계산



(b) 빈도수 업데이트 & 지지도 측정

Node	Frequency	Update	Support
A	70	70	1.00000
B	35	35	0.50000
C	20	27	0.38571
D	5	5	0.07143
E	21	23	0.32857
F	9	9	0.12857
G	5	5	0.07143
H	10	10	0.14286
I	12	12	0.17143
J	8	8	0.11429
K	15	15	0.21429
L	8	8	0.11429
M	10	10	0.14286

[Update]

$C = \sum freq_{Children} = 27$

$F = \sum freq_{Children} = 23$

[Support]

$Support(Node) = freq_{Node} / freq_{Root}$

5.3 WebTree⁺ 알고리즘

WebTree⁺는 WebTree에 로그 데이터를 매핑하는 과정이다. 5.2절에서 각 문서의 빈도수를 수정하는 과정을 보여주었다. 다음은 WebTree⁺ 알고리즘이다. WT는 WebTree를 나타내며, 각 문서마다 (d, C, f, s)를 가진다. d는 문서, C는 d의 자식 문서들의 집합, f는 d의 빈도수, s는 지지도를 나타낸다. DF는 문서 빈도수(document frequency)로써 로그 데이터에서 얻어진 각 문서에 대한 빈도수의 집합이다. N은 DF에서 WT에 매칭이 되지 않은 문서들의 집합이다.

[그림 11] WebTree⁺ 알고리즘

```
// WT = {<(d1, C1, f1, s1), (d2, C2, f2, s2)
         , ..., (dn, Cn, fn, sn)>}
// Ci = { cij | cij는 문서 di에 대한 자식 문서}
// DF = {(dm, freq) | 로그 데이터를 정제 후,
         문서 dm 에 대한 빈도수(freq)}
// N : WebTree에 매핑이 안된 문서들의 집합

WebTreePlus{
    For(DF의 각 d에 대하여) {
        If d가 WT내에 존재하면 WT의 해당 문서의 f에
           freq를 입력
        Else d를 N에 추가시킨다.
    }
    FrequencyUpdate(WT, d1)
}

FrequencyUpdate(WT,d){
    sum=0
    If p의 Child가 있으면
        For(C의 각 c에 대하여){
            FrequencyUpdate(WT, c)
            sum = sum + r
        }
        If d.f < sum 이면 d.f = sum
    Else return r = d.f
}
```

[그림 12] 표준화를 위한 지지도 측정

```
Support(WT){
    max_frequency : WT에서 루트(root) 페이지의 f값
    For(WT의 각 d에 대하여 ){
        s = d의 f값 / max_frequency
    }
}
```

[그림 11]의 WebTree⁺ 알고리즘은 세 부분으로 나누어진다. 먼저 로그 데이터에서 구한 각 문서에 대한 빈도수를 WebTree에 매핑시킨다. 이때 매핑이 되지 않은 문서들은 N에 남는다. 이 자료는 웹 사이트에서 하이퍼링크로 연결되어 있지 않은 문서들이다.

두 번째는 함수 FrequencyUpdate()를 실행시킨다. 이 함수는 WebTree의 빈도수를 수정하는 부분이며, 귀납적(recursive) 방법을 사용한다. 가장 하위의 리프(leaf) 문서부터 탐색하며 수행한다. 자식 문서들의 합을 구한 후, 현재 문서의 빈도수보다 높으면 빈도수를 자식 문서들의 합으로 수정한다. WebTree⁺ 알고리즘의 결과로는 빈도수가 추가된 WT와 매핑이 되지 않은 문서의 집합 N이다. WT는 다음장의 탐색 패턴의 발견과 추천에 필요한 자료가 된다. N은 부가적인 결과물로 웹 사이트 관리에 유용한 자료가 된다. 현재 웹 사이트에 나타나지 않는 문서들이므로 웹 사이트에 새롭게 추가 한다거나 폴더내의 불필요한 자료를 제거할 수 있다.

마지막으로 함수 Support()는 WebTree의 빈도수를 0~1사이의 표준화된 값으로 바꾸기 위하여 사용된다. [그림 12]에서 업데이트된 빈도수로부터 지지도를 구한 예를 보여준다.

6. 탐색 패턴 발견 및 사용자를 위한 문서 추천

6.1 탐색 패턴 발견

탐색 패턴 발견은 WebTree⁺에서 미리 정해둔 최소 지지도(*min_support*)를 만족하는 패턴을 찾는다. *min_support* =0.2라고 할 때 [그림 10]으로 부터 다음과 같은 패턴을 찾을 수 있다.

<표 2> *min*_support =0.2 이상 탐색 패턴

Pattern	Support
A	1.00000
A→B	0.50000
A→C	0.38571
A→B→E	0.32857
A→B→E→K	0.21429

A는 웹 사이트의 시작 페이지로 항상 1을 가진다. 하지만 이것은 가장 상위 문서이다. 패턴을 분석할 때나 사용자에게 문서를 추천할 때에는 필요로 하지 않는다.

6.2 사용자를 위한 문서 추천

웹 사용 마이닝은 온라인이 오프라인의 데이터에 대하여 흥미 있는 패턴을 발견하고 분석하여 웹에 적용하거나 사용자들에게 정보를 제공하여 준다. 하지만 웹의 빠른 변화와 사용자들의 현재 탐색 정보를 반영하지 못한다. WebTree⁺에서 발견된 패턴들은 사용자가 웹 사이트를 탐색할 때 좀더 편하고 유용한 정보를 추천한다. 사용자가 웹을 서핑하는 동안 현재 문서에서 흥미 있는 하위 패턴들을 추천해 준다. 또한 사용자의 탐색 정보를 온라인상에서 반영하기 위하여 지지도를 새롭게 계산한다. 이렇게 하므로 사용자의 탐색 정보를 능동적으로 이용할 수 있다.

7. 구현 및 실험

이 장에서는 앞 장에서 논의한 알고리즘과 이론들에 대하여 시스템을 구현해 보고 실험을 통해 패턴 분석 결과의 정확성과 시스템의 성능을 알아보려고 한다. 본 논문에서는 한양대학교 컴퓨터공학과에 소속되어 있는 13개 연구실의 웹 사이트와 로그 데이터를 사용하였다.

<표 3>은 WebTree 에이전트 수행하여 얻은 결과들을 정리한 것이다. 각 URL마다 하이퍼링크 문서를 추출하여 WebTree를 생성하게 된다.

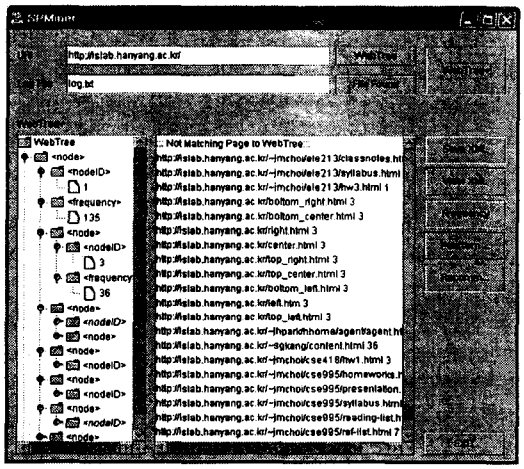
[그림 13]은 WebTree와 로그 데이터를 입력하여 WebTree⁺ 알고리즘을 실행한 결과 화면이다. 왼쪽 WebTree에서 각 문서마다 빈도수가 생성된 것을 알 수있다. 가운데의 대화창에는 WebTree에 매칭이 되지 않은 문서들이다. 이 자료는 관리자에게 유용한 자료가 된다. 웹 사이트를 개선하거나 불필요한 자료를 제거할 수 있다.

[그림 14]는 패턴 분석 결과를 보여주고 있으며 미리 주어진 최소 지지도(*min_support* =0.2)이상의 값을 찾아 URL과 지지도를 함께 보여준다.

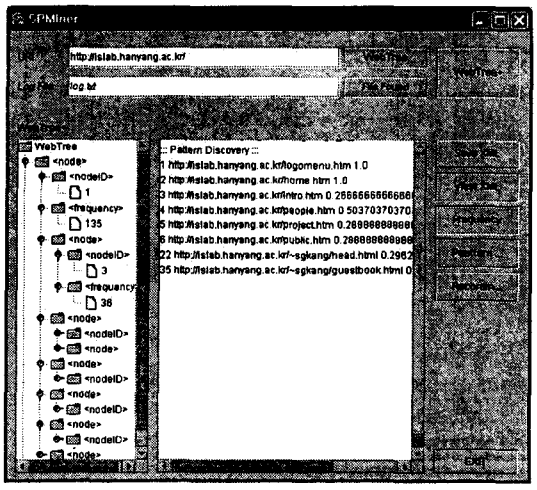
<표 3> WebTree 에이전트에 의하여 생성된 결과

No.	URl	검색문서수
1	http://viplab.hanyang.ac.kr/	14
2	http://ailab.hanyang.ac.kr/	58
3	http://para1.hanyang.ac.kr/	5
4	http://osnn.hanyang.ac.kr/	10
5	http://commlab.hanyang.ac.kr/	50
6	http://visionlab.hanyang.ac.kr/	23
7	http://distcomp.hanyang.ac.kr/	61
8	http://cns.hanyang.ac.kr/	15
9	http://mslab.hanyang.ac.kr/	52
10	http://pllab.hanyang.ac.kr/	11
11	http://islab.hanyang.ac.kr/	63
12	http://aspen.hanyang.ac.kr/	11
13	http://salab.hanyang.ac.kr/	14

[그림 13] WebTree⁺ 실행 결과



[그림 14] WebTree⁺를 이용한 패턴 분석



8. 결론 및 향후 연구 방향

본 논문은 웹 사용 마이닝에서 웹 사이트 구조를 이용하여 사용자 탐색 패턴을 발견하고 추천 가능한 SPMiner을 제안하였다. SPMiner는 로그 데이터의 전처리 과정을 줄여주고 온라인 상황에서 사용자들에게 능동적으로 유용한 패턴들을 추천하게 된다. 웹 사이트 구조를 생성하기 위하여 WebTree 알고리즘을 만들었고, WebTree와 로그 데이터에서 얻은 각 문서의 빈도수를 병합하는 WebTree$^+$를 제안하였다.

향후 연구로는 본 논문에서 사용한 html 페이지 이외의 JSP, ASP, PHP 등 다른 형태의 웹 페이지들을 가지는 웹 사이트에서도 패턴 발견 및 추천이 가능한 시스템 개발을 들 수 있겠다. 이런 페이지들은 웹 브라우저에서 html 형태의 페이지로 변환되어 보인다. 하지만 페이지가 복잡하여 정확한 하이퍼링크를 찾기에 어려운 점이 있다. 또한 사용자에게 추천하는 과정에서 사용자의 탐색 정보를 받아 지지도를 변환할 때, WebTree$^+$의 각각 지지도를 수정해야 한다. 이때 걸리는 수행 시간을 줄이기 위한 연구가 있을 수 있다. 마지막으로 본 논문에서 사용된 로그 파일에서 구한 각 문서의 지지도 이외에 사용자의 로그파일이나 다른 여러 정보를 사용하여 좀더 효율적인 패턴을 발견하기 위한 연구가 있을 수 있다.

참고문헌

[1] A. Y. Levy and D. S. Weld. Intelligent internet systems. *Artificial Intelligence*, 118(1-2), 2000.

[2] R. Agrawal, T. Imielinski and A. Swami. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowledge-Based Databases*, 9(6), pp. 914-925, 1993.

[3] O. Etzioni. The world wide web: Quagmire or gold mine. *Communication of the ACM*, 39(11), pp. 65-68, 1996.

[4] S. K. Madria, S. S. Bhowmick, W. K. Ng and E. P. Lim. Research issues in Web data mining. In *Proceedings of Data Warehousing and Knowledge Discovery, First International Conference, DaWaK'99*, pp. 303-312, 1999.

[5] R. Cooley. Web Usage Mining: Discovery and Application of Interesting Patterns from Web data. *PhD thesis, Dept.of Computer Science, University of Minnesota,* May 2000.

[6] M. Spiliopoulou. Data mining for the Web. In Proceeding of Principles of Data Mining and Knowledge Discovery, *Third European conference, PKDD'99*, pp. 588-589, 1999.

[7] R. Kosala and H. Blockeel. Web Mining Research: A Survey. *SIGKDD Explorations*, 2(1), pp. 1-15, 2000.

[8] M. Spiliopoulou. Web usage mining for web site evaluation. *Communications of the ACM*, 43(8), pp. 127-134, 2000.

[9] R. Agrawal and A. Srikant. Fast algorithms for mining association rules. *Proc. VLDB'94*, pp. 487-499, 1994.

[10] J. Han, J. Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proc. of the ACM SIGMOD, int. conf. on Management of Data*, pp. 1-12, 2000.

[11] A. Srikant and R. Agrawal: Mining Sequential patterns: Generalizations and Performance Improvements. *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, 1996.

[12] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In. *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pp. 215-224, 2001.

[13] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2), 2001.