메타 검색엔진을 위한 HTML 문서 변경 탐지기의 설계 및 구현

박 상 위[†] · 오 정 석[††] · 이 상 호[†††]

요 약

검색엔진의 HTML 문서는 수시로 변경되고 있으며, 이는 각 검색엔진의 결과 문서를 통합하여 사용자에게 제공하는 메타 검색엔진의 기능을 저하시키는 요인이 된다. 이에 대한 해결방법으로 본 논문에서는 HTML 문서의 변경을 탐지하는 HTML 문서 변경 탐지기를 설계하고 구현한다. 문서 변경 탐지기는 문서 구조를 추출하기 위해 위치 정보 알고리즘과 수정된 Jaak Vilo 알고리즘을 사용하고, 그 결과로 패턴을 추출한다. 문서 변경 탐지기는 HTML 문서에서 반복적으로 출현하는 구조를 표현하는 패턴을 사용한다. 또한, 문서 변경 탐지기의 정확성을 측정하기 위하여 문서 변경에 대한 전략을 세우고 이를 기반으로 실험을 수행한다.

Design and Implementation of an HTML Pages Modification Detector for Meta-search Engines

Sang-Wee Park[†] · Jeong-Seok Oh[††] · Sang Ho Lee[†††]

ABSTRACT

HTML pages in the web change at any time. It could cause to decrease the functionality of meta-search engines which provide users with integrated results of search engines. To solve this problem, we propose an HTML pages modification detector. It utilizes information of element positions in HTML pages and the modified Jaak Vilo algorithm. The HTML page modification detector uses patterns that represent the structure of HTML expressions occurring repeatedly in HTML pages. An experiment is carried out to verify the correctness of the modification detector.

키워드 : 메타 검색엔진(Meta-search engines), HTML 문서 변경 탐지기(HTML pages modification detector), 문서 구조(Document structure), HTML

1. 서 론

사용자들은 웹에서 효율적인 정보검색을 위해 검색엔진을 이용하고 있다[1]. 일반적으로 검색엔진이 대상으로 하는 웹 사이트 영역은 상이하며, 또한 각 검색엔진이 전체 웹 페이지에서 검색할 수 있는 범위는 매우 낮다[2]. 이러한 문제점을 해결하는 방법으로 다수의 메타 검색 엔진이 개발되어 운영되고 있다. 메타 검색엔진은 각 외부 검색엔진들의 검색결과를 병합하여 단일한 형태로 사용자에게 제공하여, 다수의 검색엔진을 검색한 효과를 얻을 수 있다[3-6].

메타 검색엔진은 목적 검색엔진들의 결과를 통합하는 웹 문서 통합기의 역할을 한다. 따라서, 메타 검색엔진에서 질의하는 목적 검색엔진들은 항상 활성화되어 있어, 검색 결과를 반환해야 한다. 목적 검색엔진의 사용자 입출력(interface) 페이지에서는 일반적으로 <FORM> 태그를 사용하여 질의를 입력받고 이를 통해 검색 질의 형식이 결정된다. 목적 검색엔진의 결과 HTML 페이지는 메타 검색엔진에 불필요한 데이터를 다수 포함하고 있으므로 검색 결과 HTML 페이지 내에서 유효한 데이터만 추출하기 위해 래퍼(Wrapper)를 사용한다. 특히, 래퍼는 HTML 문서의 내용과는 무관하게 문서의 구조를 이용하여 데이터를 추출하기 때문에, 목적 검색엔진의 HTML 문서 변경은 메타 검색엔진이 목적 검색엔진으로 접속이 불가능하게 하거나, 결과 HTML 페이지 내에서 유효한 데이터를 추출하지 못하게 할 수 있다. 원칙적으로 검색 엔진 결과 페이지의 변경은 수시로 발생할 수 있으며 이는 유효한 데이터 추출을 불가능하게 하는 등의 메타 검색엔진 기능을 저하시키는 요인이 된다.

검색엔진이나 쇼핑몰 등과 같이 많은 정보를 표현하는

※ 본 논문은 한국과학재단 목적기초연구(2000-2-51200-002-3) 지원으로 수행되었음.
† 준 회 원 : (주)지택 인터내셔널
†† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
††† 정 회 원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2001년 7월 23일, 심사완료 : 2002년 3월 7일

사이트들은 HTML 문서에 반복 구조를 가지는 특징이 있다. 이는, 반 구조적 특징을 내포하는 HTML 문서에서 구조를 추출하여 HTML 문서 변경의 여부를 탐지할 수 있는 방법을 제시하므로 메타 검색엔진의 유용성을 향상시킬 수 있다.

본 논문은 메타 검색엔진의 유용성을 증가시키기 위해, HTML 문서의 변경을 자동으로 탐지하는 HTML 문서 변경 탐지기를 설계하고 구현하였다. 변경 탐지기는 문서의 변경을 탐지하기 위해 문서의 내용을 분석하고 비교하는 것이 아니라 문서의 구조를 추출하여 분석하고 비교한다. 즉, 의미적(semantic) 분석/비교가 아닌 구조적(structural) 분석/비교를 수행한다. 문서의 구조를 추출하기 위해 위치정보 알고리즘과 수정된 Jaak Vilo 알고리즘을 적용하였으며, 결과로서 패턴을 생성한다. 패턴은 HTML 문서에서 반복적으로 출현하는 구조를 표현한다. HTML 문서 변경은 문서에서 발견되는 패턴을 기반으로 하여 탐지된다. 또한, 변경 탐지기의 정확성을 입증하기 위해 110개의 실제 웹 문서들을 대상으로 가능한 모든 변경 경우를 포함하는 실험을 수행하였으며, 수행 결과도 제시한다.

본 논문의 구성은 다음과 같다. 2장은 문서 변경 탐지의 관련연구와 본 논문의 패턴 발견 알고리즘의 기반인 Jaak Vilo 알고리즘을 설명하고, 3장은 HTML 문서의 구조와 반복 구조를 이용한 패턴과 문서 변경 탐지기의 구조에 관하여 설명하며, 4장에서는 문서 패턴 탐지기의 구조와 탐지기에서 사용하는 위치 정보 알고리즘과 수정된 Jaak Vilo 알고리즘을 기술한다. 5장에서는 구현된 문서 변경 탐지기의 실험을 위한 실험 전략과 실험 결과에 대해 상술하고, 6장은 결론 및 향후계획을 제시한다.

2. 관련 연구

2.1 문서 변경 탐지

문서 변경 탐지는 동일한 문서의 현재 상태를 이전 상태와 비교하여 문서 변경 여부를 알아낼 수 있다[7]. 문서 비교는 비구조적인(unstructured) 문서와 반구조적인(semi- structured) 문서로 분류하여 비교 방법을 다르게 사용할 수 있다. 비구조적인 문서는 구조가 없는 일반적인 텍스트(flat-file type)로 구성된 문서이다. 이러한 유형의 문서를 대상으로 하는 문서 변경 탐지는 두 문서를 비교하기 위해 대체적으로 LCS (Longest Common Subsequence) 같은 알고리즘 등을 적용하였다. 반구조적인 문서는 SGML이나 HTML과 같은 마크업 언어로 작성된 문서로서 구조 정보가 표현 정보와 섞여 있으므로 문서의 구조 이용이 어려운 문서를 나타낸다. 반구조적인 문서를 대상으로 LCS 같은 알고리즘을 적용하면 두 문서의 일치되는 총 단어수와 총 마크업정보의 개수를 알아낼 수 있으며, 이를 사용하여 문서 내의 데이터 변경은 탐지가 가능하지만 구조의 변경을 탐지할 수 없다[8]. 특히, 데이타 웨어하우징, 문서의 버전 관리, 디지털 라이브러리, 인터넷 데이타베이스 분야에서 SGML이나 HTML로 작성된 반구조적인 문서의 구조적 일치(structural matching)를 발견하는 것은 중요한 작업이다. 또한, 많은 데이타베이스 연구에서도 SGML, HTML과 같은 구조적인 문서 관리를 다루고 있으며[9-11], 이러한 시스템들은 문서를 비교하거나 패턴을 발견하기 위해 문서의 구조를 사용한다. TreeDiff[12]는 문서를 표시된 순서화 트리(labeled ordered tree)로 변환하여 레이텍(Latex)으로 작성된 두 문서를 비교한다. TreeDiff를 확장한 LaDiff[13, 14]는 구조적 일치를 제공하기 위해 계층 구조의 문서를 트리 구조로 변환하고, 그 트리의 변경을 이용하여 문서 구조의 변경을 알아낸다. 반구조적인 문서를 처리하기 위해 문서를 트리 형태나 그래프 형태로 변환하여 노드를 비교하는 방법이 제시되었다. 그러나, 트리 형태의 문서 비교는 모든 노드를 비교하므로 비싼 비용이 요구되며, 그래프 형태의 문서 비교는 객체 순서(object ordering)와 같은 정보를 손실할 수 있다[8]. 본 논문에서는 검색엔진의 검색 결과 HTML 문서 전체가 아닌 문서 내의 반복되는 구조를 찾아 문서의 변경을 탐지하며, 문서의 반복 구조를 찾기 위해 수정된 Jaak Vilo 알고리즘을 사용한다.

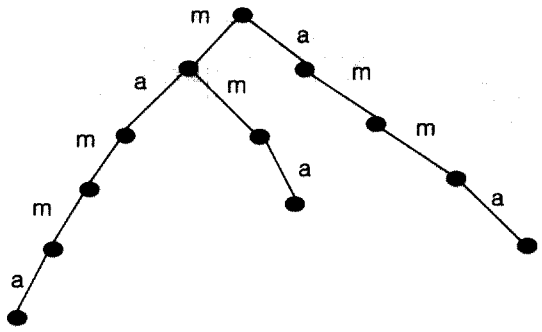
2.2 Jaak Vilo 알고리즘

Jaak Vilo 알고리즘[15]은 생명공학 분야에서 유전자의 반복 구조를 발견하는데 사용되며[16, 17], 접미사 트라이(Suffix Trie) 자료 구조를 기반으로 한다. 접미사 트라이[18-21]는 트라이(Trie)나 패트리시아(Patricia)와 유사한 자료구조로서, 입력 문자열의 모든 접미사(Suffix)를 사용하여 트리를 생성함으로써 반복 문자열을 찾게 된다.

입력 문자열 "mamma"에 대한 접미사를 구해 보면, "mamma", "amma", "mma", "ma", "a" 이다. 접미사 트라이를 생성하는 방법은 현재 접미사의 첫 문자가 트리에 존재하면, 다음 문자를 위해 자식 노드를 만든다. 예를 들면, 접미사 "mma"가 입력될 때, 이미 "mamma"와 "amma"가 트리에 존재하고 있으며, "mma"의 첫 문자 "m"이 노드에 존재하는 가를 검사한 후 존재하면, 다음 문자 "m"을 노드 "m" 하위에 자식노드로 생성시킨다. 그러므로, 문자열 "mamma"의 접미사 트라이는 (그림 1)과 같다.

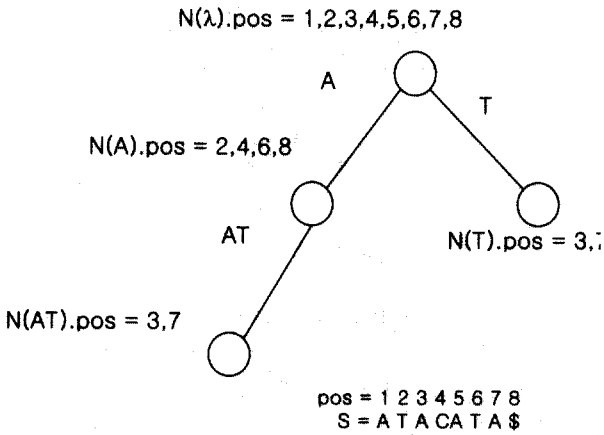
Jaak Vilo 알고리즘은 길이가 N인 입력 문자열 내에서 K회 이상 반복적으로 발생하는 문자열을 발견하는 알고리즘이다. 기존의 접미사 트라이는 문자열에서 발생하는 모든 접미사를 트리 형태로 표현하기 때문에 메모리 사용량이 많았다. 그러나, Jaak Vilo 알고리즘에서는 문자열에서 K회 이상 나타나는 문자만 사용해 트리를 생성함으로서 접미사 트라이의 단점인 메모리 사용량을 줄였다. Vilo 트리를 생성하는 방법은 입력 문자열에서 나타나는 반복횟수를 결정하고, 입

력 문자열을 구성하고 있는 모든 문자에 대해 위치 정보를 구한다. 이 때, 출현 횟수가 반복 횟수 이상이 되는 문자로 트리를 구성한다.



(그림 1) 문자열 "mamma"에 대한 접미사 트라이

(그림 2)는 문자열 "ATACATA$"에 대해 2회 이상 출현하는 문자열을 구하는 Jaak Vilo 트리를 생성한 것이다. 예제에서 초기에 반복횟수를 만족하는 문자는 "A", "T"이다. 먼저, 루트로부터 자식 노드를 생성하여 "A"와 "T"를 삽입한다. "A"의 위치 값은 "A" 다음에 나타나는 문자를 구하기 위해 "A"가 출현하는 위치 (1, 3, 5, 7)에 (+1)을 한 결과인 (2, 4, 6, 8)을 구한다. "T"의 위치 정보도 원래 출현 위치 (2, 6)에서 (+1)을 한 (3, 7)로 정한다. 따라서, "A" 다음에 나타나는 문자는 (2, 4, 6, 8)에 출현하는 문자 중 반복 출현 횟수 2를 만족하는 문자가 선택된다. 따라서, (2, 6)에 해당하는 "T"를 "A" 다음에 추가한다. 위치 (4)에는 "C"가 나타나며, 위치 (8)에는 문자열의 끝을 표시하는 "$"이므로, 이 두 문자는 트리 생성 요소에서 제외된다. 그리고, "A"다음에 나타나는 "T"의 위치는 (3, 7)인데, 위치 3에 해당하는 문자는 "A"이고, 위치 7에 해당하는 문자도 "A"이지만, "A"는 "T"에 선행하므로 트리에 삽입하지 않는다. 이러한 방식으로 생성된 Vilo 트리에서 반복 문자열을 추출하면 "AT" 와 "T"가 해당한다.



(그림 2) 문자열 "ATACATA"에 대한 Vilo 트리

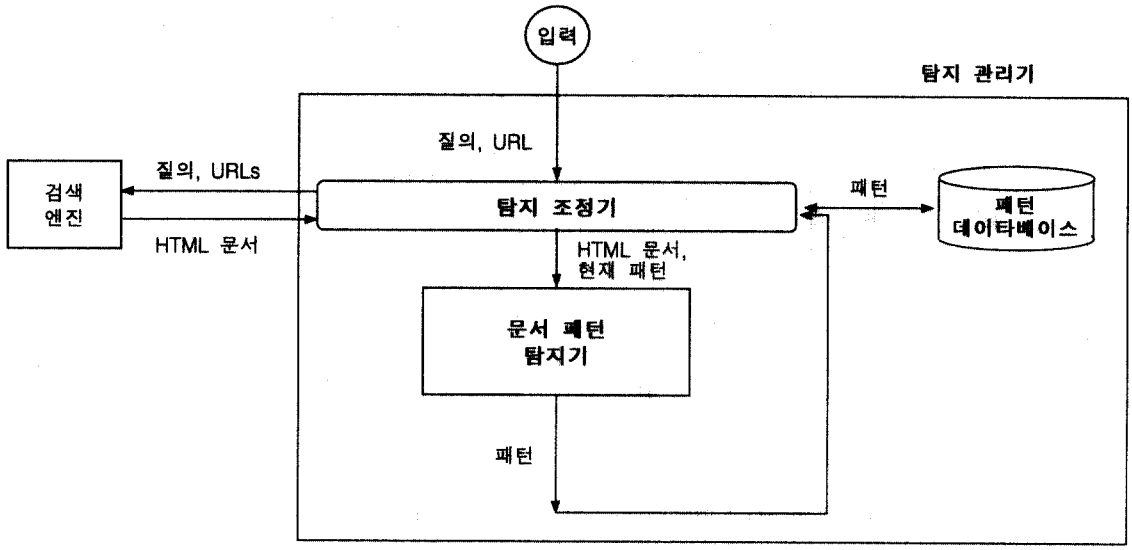
3. 패턴 및 HTML 문서 변경 탐지기

HTML은 마크업(Markup) 정보로 구성되며, 마크업 정보는 형식적(Stylistic) 정보, 구조적(Structural) 정보, 의미적(Semantic) 정보로 구성된다. 형식적 정보는 문서의 형식을 표현하는 태그들의 집합으로 "<FONT>, <B>, <U>"등과 같은 태그들이 있다. 구조적 정보는 문서의 구성을 조작하는 태그들로 구성되며 "<Hn>, <P>, <DIV>"와 같은 태그들이 있다. 의미적 정보는 문서의 의미를 표현하는 태그이며, "<Title>, <Code>"와 같은 태그들이 존재한다. 그리고, HTML 문서는 이 세 정보를 복합하여 사용하기 때문에, 문서를 구조적으로 나타내기보다는 문서의 표현을 나타내기에 적합하다. 이는, HTML 문서가 기존의 데이터베이스 시스템에서 사용되었던 데이터와 동등하게 취급될 수 없기 때문에, 웹 상에서 데이터의 통합 및 추출에 대한 대상으로 부적당한 것임을 보이고 있다[22]. 그러나, 본 논문에서 대상으로 하는 HTML 문서는 일반적으로 쇼핑몰, 검색엔진, 부동산 등의 많은 정보를 표현하고 있으며 이들은 사이트의 특성상, 문서 내에 반복적인 구조를 가지고 있다. 예를 들면, 쇼핑몰 문서에서 각각의 상품 소개와 가격에 대한 내용은 다르지만 문서의 구조는 동일한 태그와 문자열의 순서로서 반복된다. 검색엔진의 결과 문서의 반복 구조는 (그림 3)과 같다. HTML 문서 변경은 검색 결과 문서의 구조 변경을 의미하며, 문서의 변경을 탐지할 수 있는 기반이 된다.

(그림 3) 검색엔진 문서에 나타나는 반복 구조

HTML 문서의 변경 탐지를 위해 본 논문에서는 HTML 문서 특성을 표현하는 '패턴'을 이용한다. 패턴은 문서에 반복적으로 출현되는 구문을 표현하며, 다음과 같은 특성을 갖는다.

① 패턴은 지정된 반복 횟수 이상으로 HTML 문서 구조에 출현되어야 한다. 반복 횟수는 사용자에 의해 임의의 수로 지정될 수 있다. 특히, 반복 횟수에 따라 문서의 패턴이 다르게 생성되므로, 반복 횟수의 결정은 문서에서 패턴의 발견과 변경 탐지 시에 중요한 사항이다.



(그림 4) HTML 문서 변경 탐지기의 구조

② 패턴의 구성요소는 문서 구조를 기반으로 하기 때문에 태그와 "TEXT"의 조합이어야 한다. 태그는 일반적으로 태그 이름과 속성으로 이루어져 있지만, 패턴에서는 속성을 제외한 태그 이름만을 사용한다. "TEXT"는 HTML 태그를 제외한 내용을 표시한다.

③ 패턴을 구성하는 구성요소는 세 가지 이상의 다른 요소로 이루어진다. 예를 들면, "TEXT"나 "<A>TEXT<A> TEXT<A>"는 패턴에서 제외된다. "TEXT"는 단일 요소이고, "<A>TEXT<A>TEXT<A>"는 단지 두 개의 요소로만 이루어졌기 때문이다.

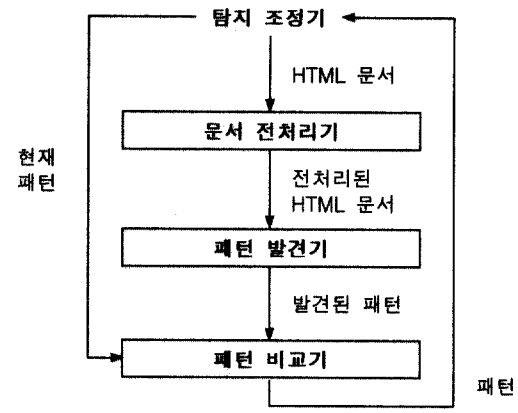
④ 패턴의 구성요소에서 동일한 태그의 시작 태그(start tag)와 종료 태그(end tag)는 다르게 취급한다. 즉, "<A>"와 "</A>"는 다른 요소이다.

문서 변경 탐지기는 HTML 문서 내의 구조에 대한 패턴을 생성하고 기존 HTML 문서의 패턴과 현재 문서 패턴을 비교하여 HTML 문서 변경을 탐지한다. 문서 변경 탐지기의 전체 구조는 (그림 4)와 같은 구조로 탐지 조정기, 문서 패턴 탐지기와 패턴 데이타베이스의 세 부분으로 구성된다.

탐지 조정기는 사용자 질의(User query)와 검색 사이트의 웹 주소로 구성된 질의를 입력으로 받는다. 목적 검색 엔진의 질의에 대한 응답은 HTTP 상태 코드[23]와 결과 HTML 문서가 반환된다. 오류가 탐지되지 않으면 탐지 조정기는 질의 결과 HTML 문서와 패턴 데이터베이스에 저장되어 있는 기존 패턴에 대한 정보를 문서 패턴 탐지기로 전송한다. 문서 패턴 탐지기는 탐지 조정기에서 HTML 문서를 입력으로 받아, 입력 문서에 전 처리를 하고, 수정된 Jaak Vilo 알고리즘을 사용하여 전 처리된 문서로부터 패턴을 생성한다. 생성된 패턴은 기존 문서 패턴과 비교되어 결과를 탐지 조정기로 반환한다. 패턴 데이터베이스는 HTML 문서의 현재 패턴에 대한 정보를 저장하고 있는 데이터베이스이다.

4. 문서 패턴 탐지기

문서 패턴 탐지기는 HTML 문서를 전 처리하는 문서 전처리기, 전 처리된 HTML 문서에서 패턴을 발견하는 패턴 발견기와 기존의 패턴과 발견된 패턴을 비교하는 패턴 비교기로 구성된다. (그림 5)는 문서 패턴 탐지기의 구조를 보이고 있으며, 본 장 전반적으로 설명된다.

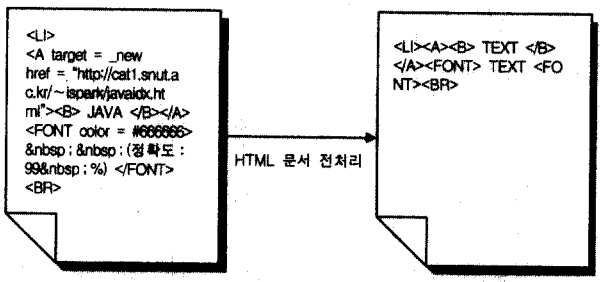


(그림 5) 문서 패턴 탐지기의 구조

4.1 문서 전 처리기(Pre-processor)

HTML 문서는 태그마다 각 태그의 속성(attribute)을 함께 표현하고 있고, 문서에서 표현하는 내용도 제각기 다르다. 이를 효율적으로 처리하기 위하여 문서 전처리 과정은 HTML 문서의 형태를 정형화한다. HTML 문서에 대한 전처리는

두 가지로 나누어서 처리된다. 첫 번째, 문서 내에 있는 태그와 문자열의 형태를 변환한다. 태그의 변환은 태그 이름만 남기기위해 문서 내의 태그에 대한 모든 속성들을 삭제하고, 문서에서 태그가 아닌 문서의 내용의 형태는 문자열 "TEXT"로 치환한다. 첫 번째 문서 전처리가 끝나게 되면, HTML 문서는 "<", 태그 이름, "/", ">", "TEXT"의 형태로 변환된다. (그림 6)은 HTML 문서 전처리의 첫 번째 과정을 예제로 보이고 있다.



(그림 6) HTML 문서 전처리의 예

HTML 문서에서 문서의 내용을 기술하고 문서의 구조를 구성하는 부분은 "<Body>" 태그 내의 부분이므로, 문서 전처리의 두 번째는, HTML 문서에서 "<Head>"에서 "</Head>" 부분을 삭제하고 "<Body>"에서 "<Body>" 부분만 남기도록 하고, 문서의 끝을 나타내는 문자 "$"를 전 처리된 문서 마지막에 삽입한다.

4.2 패턴 발견기(Pattern Finder)

HTML 문서는 문서의 형식적 정보를 표현하는 태그를 사용할 수 있다. 형식적 태그는 문서에서 불규칙적으로 나타나기 때문에, 동일하게 반복되는 문서의 구조를 추출하는 것이 어렵다. 또한, Vilo 알고리즘은 패턴의 부(Sub) 패턴도 생성하여 많은 수의 패턴이 나타나며, Vilo 알고리즘은 문서 구성 요소 중에서 반복 횟수를 만족하는 구성 요소를 선택할 때, 구성 요소 출현 위치의 개수를 이용한다. 이때, 초기 구성 요소 선택 시에 모든 구성 요소를 비교한다. 즉, 위치 개수가 반복 횟수를 만족하지 않는 구성 요소도 비교를 수행한다. 따라서, 비교 횟수를 줄이기 위해 초기에 반복횟수를 만족하지 않는 문서 구성 요소를 입력 문자열의 끝을 의미하는 "$"로 변환하여 처리하도록 수정하였다. 수정된 Vilo 알고리즘에서 문서의 형식에 관련된 태그들은 일반 문자열과 동일하게 취급하여 "TEXT"로 치환한다. "TEXT"로 치환되는 형식적 태그는 "<FONT>, <TT>, <I>, <B>, <BIG>, <SMALL>, <STRIKE>, <S>, <U>"이다[24]. 그리고, 수정된 Vilo 알고리즘은 패턴의 비교횟수를 줄이기 위해 최장 패턴만 패턴으로 나타나도록 패턴의 개수를 줄였다. 예를 들면, 반복 구조 "<A>TEXT</A>"가 있으면, Vilo 알고리즘은 "<A>TEXT</A>" 뿐만 아니라, "TEXT</A>"와 "</A>"를 패턴으로 출력한다. 이때, "<A>TEXT</A>"를 최장 패턴이라 한다.

패턴 발견기는 전처리 된 HTML 문서에 수정된 Vilo 알고리즘을 적용하여 문서에서 반복적으로 발생하는 구조 패턴을 발견한다. 수정된 Vilo 알고리즘은 문서를 하나의 문자열로 다루며, 문자열 내에 존재하는 문자들의 위치 정보를 이용하여 패턴을 찾는다. 위치 정보를 생성하는 절차는 초기에 입력 문자열의 모든 원소에 대한 집합을 구한다. 그리고, 원소 집합에 속하는 각 원소에 대해 입력 문자열에서의 위치 정보를 찾아낸다. <표 1>에 기술되어 있는 위치 정보를 생성하는 알고리즘을 이용하여 문서의 구성요소인 HTML 태그들과 "TEXT"에 대해 위치 정보를 구한다.

<표 1> 위치 정보를 만드는 알고리즘의 의사코드

```
Input : String S
Output : Set(S), ie., set of input string S have position
         information

I ← 0 ;
J ← 0 ;
Set(S) ← null ;
while (S.length) {
  data[I] ← Tokenized value in S by "<, >" ;
  Set(S) ← add data[I] ;
  Increase I ;
} // end while

while (Set(S).size) {
  for (J = 0, J < size of data, J++) {
    if (Set(c) is equals to data[J])
      Set(c).pos ← Set(c).pos + J ;
  } // end for
  } // end while
```

입력 문자열 S에 대해서, Set(S)는 S를 이루는 모든 구성 원소에 대한 집합(Set)을 의미하며, Set(c)는 Set(S)의 원소 "c"를 의미한다. 위치 정보 알고리즘의 예제는 다음과 같다.

[예제 1] 입력 문자열 S에 대한 위치 정보를 생성한 후에 Set(S)를 출력한다. Set(S)는 입력 문자열의 모든 구성 원소와 그의 위치 정보도 함께 반환한다.

```
입력 문자열 S = "TEXT<BR>TEXT<BR><Img>$" ;
출력 : Set(S) = { TEXT, <BR>, <Img>, $}
data[i] :  { TEXT  <BR>  TEXT  <BR>  <Img>  $ }
   i =       0     1      2     3      4     5
Set(TEXT).pos = {0, 2} ;
Set(<BR>).pos = {1, 3} ;
Set(<Img>).pos = {4} ;
Set($).pos = {5} ;
```

■

〈표 2〉 수정된 Jaak Vilo 알고리즘의 의사코드

```
/* Set(S), Set(c), and character c are caused by position
information algorithms */
while (S.length) {
  if (charcater c ∈ Ω)
    convert c to "TEXT" in S ;
  if (Set(c) < K)
    c ← $ ;
} // end while
  new node ← Root ; Root.char ← λ ;
  Root.pos ← (1,2,···,|S|) ;
  enqueue (Q, Root) ;
  while (Node ← dequeue(Q)) {
    foreach character c ∈ Σ
      Set(c) ← null ;
    foreach position ∈ Node.pos
      add position+1 to Set(S[position]) ; // Set(S[position] =
      Set(c) in each Node.pos
    foreach character c ∈ Σ such that |Set(c)| ≥ K {
      NP ← Root pointer
      while (NP.child.size) {
        if (value of NP is equals to c in Q) {
          delete NP in tree ;
        } // end if
        NP ← Root.child ;
      } // end while
      while (character in Q) {
        if (c is equals to character)
          remove character ;
      } // end while
      Node.child(c) ← newNode with lavel newNode.char = c ;
      newNode.pos ← Set(c) ;
      enqueue(Q, newNode) ;
    } // end foreach
  } // end while
  return Root ;
```

<표 2>는 수정된 Vilo 알고리즘의 의사코드를 보이고 있다. 진하게 표시된 부분은 Vilo 알고리즘에서 수정된 부분이다. <표 2>의 알고리즘에 나타나는 기호는 다음과 같다.

- Ω = {<TT>, <I>, <B>, <BIG>, <SMALL>, <STRIKE>, <S>, <U>}
- 문자(character) c는 HTML 문서를 구성하는 요소, HTML 태그 혹은, "TEXT"
- |S|는 입력 문자열 S의 길이
- Σ는 S에 나타나는 문자 c
- "$"는 입력 문자열 S에서 마지막 문자 표시
- λ는 루트(root)에 대한 표시
- Q는 자료구조 큐(Queue)를 표현

수정된 Jaak Vilo 알고리즘은 초기에 입력 문자열 내에 존재하는 형식적 태그들을 "TEXT"로 변환한다. 수정된 Vilo 알고리즘은 최장 패턴을 유지하기 위해 두 가지 방법을 사용한다. 첫 번째는 현재 큐에 입력될 값 문자 c와 트리 내의 노드 값을 비교하여 동일한 값이 있으면 트리에서 그 노드에 대한 포인트를 삭제한다. 두 번째는 문자 c와 큐에 들어있는 값 문자가 동일한 값이면, 문자를 삭제한다. 수정된 Vilo 트리에서 패턴을 추출하기 위한 트리의 검색은 루트로부터 DFS (Depth-First-Search) 방식으로 노드를 검색한다.

다음은 HTML 문서에서 전 처리 과정을 거친 후 수정된 Vilo 알고리즘을 이용하여 패턴을 찾는 예제이다.

[예제 2] 입력 HTML 문서를 전처리하고, 전 처리된 HTML 문서에서 수정된 Jaak Vilo 알고리즘을 사용하여 (그림 7)에 나타나는 단계를 거쳐 패턴을 생성한다.

HTML 문서 :
"<HTML> <HEAD> <TITLE> Example </TITLE> </HEAD> <BODY> This Document is example of patterns. <H1> First </H1>
 <H1> Second </H1>
 <H1> Third </H1>
 </BODY> </HTML>"

전 처리된 HTML 문서 :
"<HTML> <BODY> TEXT <H1> TEXT </H1>
 <H1> TEXT </H1>
 <H1> TEXT </H1>
 </BODY> </HTML>"

전 처리된 문서를 다음 (그림 7)에 나타나는 단계로 자료구조 큐와 트리를 이용해 패턴을 발견한다. 예제의 반복 횟수는 3으로 한다. 문자열 S를 입력으로 하여 출력 문자 집합 Set(S)를 생성해 낸 후 각 문자의 위치 정보를 구한다.

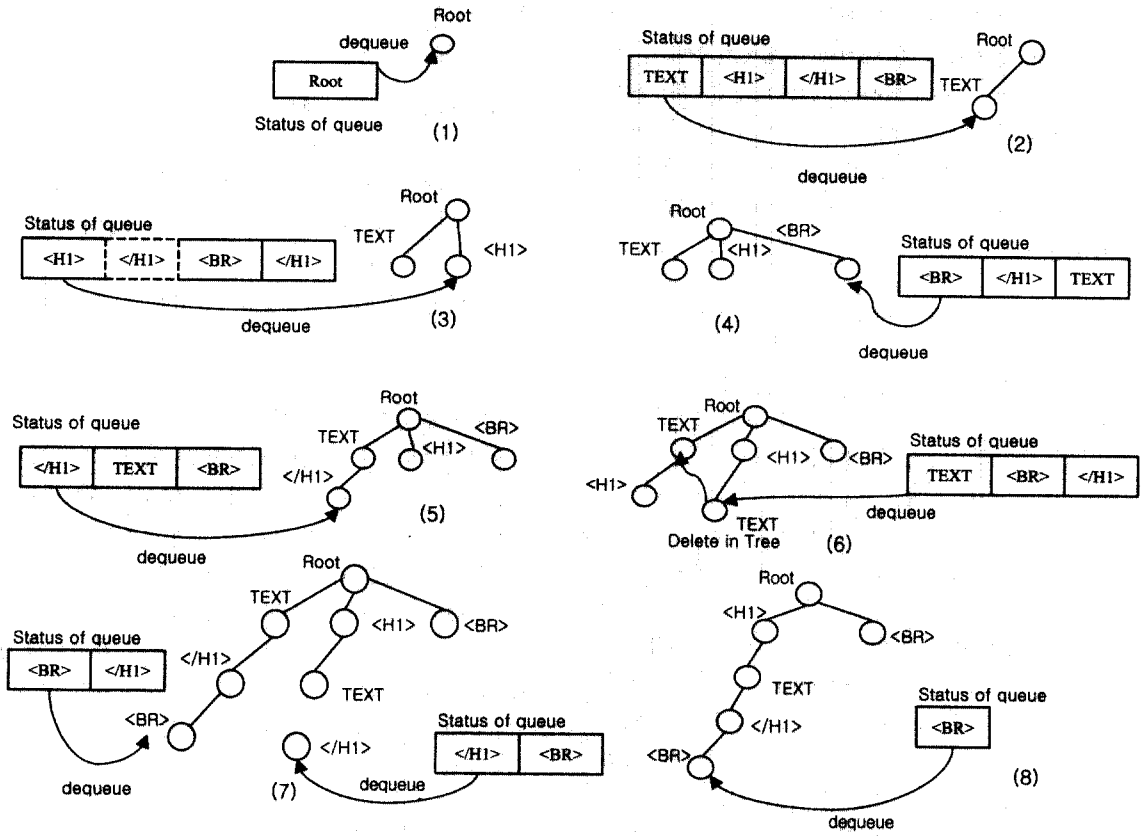
입력 문자열 S = "<HTML> <BODY> TEXT <H1> TEXT </H1>
 <H1> TEXT </H1>
 <H1> TEXT </H1>
 </BODY> </HTML>$" ;

출력 문자 집합 Set(S) = {<HTML>, <BODY>, TEXT, <H1>, </H1>,
, </BODY>, </HTML>, $}

data[i] :	<HTML	<BODY>	TEXT	<H1>	<TEXT>	</H1>	 	<H1>	···$}
i =	0	1	2	3	4	5	6	7	···17

Set(<HTML>).pos = {0} ;
Set(<BODY>).pos = {1} ;
</BODY>, </HTML>, $
Set(TEXT).pos ={2, 4, 8, 12} ;
Set(<H1>).pos = {3, 7, 11} ;
Set(</H1>).pos = {5, 9, 13} ;
Set(
).pos = {6, 10, 14} ;
Set(</BODY>).pos = {15} ;
Set(</HTML>).pos = {16} ;
Set($).pos = {17} ;

전 처리된 HTML 문서에 대한 트리 생성 단계가 (그림 7)에 나타나고 있으며, 먼저, (1)과 같이 루트 노드를 트리에 생성한다. 루트의 위치 정보 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17)에 출현하는 구성 요소 중 반복 횟수를 만족하는 요소를 선택하여 큐에 삽입하고, 단계 (2)의 큐의 상태가 된다. 단계 (2)에서 "TEXT"를 큐에서 값을 빼내어, 루트 노드 하위에 삽입하고 "TEXT" 위치 정보 (2, 4, 8, 12)에 (+1)을 한 (3, 5, 9, 13)에 출현하는 요소 중 반복횟수를 만족하는 요소를 선택하면 "</H1>"이다. 이와 같은 방식으로 큐를 이용하여 트리를 생성한다. 특히, 단계 (3)과 (6)이 수정된 알고리즘이 적용된 부분으로, 단계 (3)은 "</H1>"이 큐에 삽입되면서, 이미 큐에 저장되어 있던 "</H1>"을 삭제한다. 단계 (6)은 "TEXT"를 큐에서 빼내어 트리에 삽입할 때 트리에 중복된 노드 값이 존재하므로, 기존의 "TEXT"에 대



(그림 7) 전 처리된 문서에서 패턴을 위한 트리 생성의 단계

한 포인트를 삭제한다. (그림 7)에서 나타나는 단계 (8)에서 패턴을 구하면 "<H1>TEXT</H1>
"과 "
" 이다. "
"은 패턴의 조건 중에서 구성 요소 개수를 만족하지 않기 때문에 패턴에서 제외되므로, 마지막으로 남는 패턴은 "<H1>TEXT</H1>
"이다. ■

4.3 패턴 비교기(Pattern Comparer)

패턴 비교기는 새로 발견된 패턴과 탐지 조정기에서 전달받은 현재 패턴을 비교하는 작업을 수행한다. 이 때, 두 패턴의 비교는 패턴의 개수와 패턴 구성요소의 순서를 모두 고려한다. 패턴 순서 비교는 각 패턴의 전체가 정확하게 일치(exact match)하는 가를 비교하며, 각 패턴 전체에서 부분적으로 일치하는 경우의 패턴은 달라진 것으로 가정한다. HTML 문서의 패턴이 달라졌는가를 탐지한 후에 패턴 비교기의 결과인 HTML 문서의 패턴을 탐지 조정기로 반환한다. 탐지 조정기에서 전달받은 패턴을 패턴 데이타베이스에 전달하여, 패턴을 갱신하는 작업을 수행한다.

5. 실험 전략 및 결과

구현된 문서 변경 탐지기가 올바르게 수행되는지를 검증하기 위해 정확성을 측정할 수 있는 문서 변경 전략을 구축하였으며, 실험에 적용하였다. 실험의 효율을 위하여 네트워크 환경을 배제하였고 이를 위해, 각 문서들을 하드디스크에 저장하고 실험하였다. 실험에 사용되는 데이터는 문서 내에 반복 구조를 가지는 HTML 문서로 총 110개의 실제 웹 문서이다. 문서의 구성은 쇼핑몰 문서 10개, 부동산 문서 10개, 검색엔진 문서 59개, 메타 검색엔진 문서 16개와 15개의 기타 웹 문서로 구성하였다. 실험 데이터 문서들은 원본을 기준으로 변경시킨 후 변경된 문서의 패턴과 변경 전 문서들의 패턴을 비교하여 성공적인 변경 탐지 비율을 측정하도록 한다.

5.1 실험 전략

문서 변경은 문서 구성요소의 삽입, 삭제, 갱신의 변경 동작으로 인한 것이고, 문서 패턴 발견기에서 HTML 문서 구성요소를 형식적 태그와 비 형식적 태그, "TEXT"로 구분하여 문서 변경 동작에 적용하므로, <표 3>과 같이 여섯 가지 기본 변경 전략으로 구분하였다. 여섯 가지 변경 전략은 문서 변경의 모든 경우를 포함하기 위해 문서 구조 패턴의 구성요소 변경 동작을 삽입, 삭제, 갱신으로 분리하였다. 첫 번째로 삽입 동작 시에 문서 구조를 변경시키기 위해 태그의 종류에 따라 삽입 위치를 다르게 하였다. 즉, 형식적 태그는 태그 사이에 삽입하고, 비 형식적 태그는 문서의 "TEXT"

전후에 삽입한다. 실험에서 태그가 삽입되는 위치는 <표 3>의 설명과 같다. 두 번째로 삭제 동작에는 형식적 태그를 삭제하거나, 비 형식적 태그를 삭제하는 경우가 존재하며, 마지막으로 갱신 동작은 형식적 태그를 비 형식적 태그로 갱신하거나 비 형식적 태그를 형식적 태그로 갱신하는 경우가 존재한다. 따라서, 문서 변경 전략에는 이를 적용한 총 여섯 가지 경우가 존재한다.

〈표 3〉 문서 변경 전략

변경 동작 내용	설 명
삽 입	태그 사이에 형식적 태그 삽입 (Is)
	"TEXT" 사이에 비 형식적 태그 삽입 (I)
삭 제	형식적 태그 삭제 (Ds)
	비형식적 태그 삭제 (D)
갱 신	형식적 태그를 비형식적 태그로 갱신 (Us)
	비형식적 태그를 형식적 태그로 갱신 (U)

기본 문서 변경 전략에 사용되는 형식적 태그는 "<TT>, <I>, <B>, <BIG>, <SMALL>, <STRIKE>, <S>, <U>"이 속하며, 비 형식적 태그는 HTML 태그 중에서 형식적 태그를 제외한 나머지 태그를 지칭한다. 기본 전략과 더불어, 실험 분류를 위해 HTML 태그 중에는 종료 태그가 존재하는 태그와 종료 태그가 존재하지 않는 태그로 분류하여 사용한다. 각 구분된 태그는 형식적 태그 "S", 비 형식적 태그 "NS", 종료 태그가 존재하는 태그 "C", 종료태그가 존재하지 않는 태그 "NC"로 지칭한다. 실험은 총 세 가지로 분류되며, 각 실험 종류에 대한 설명은 다음과 같으며, 실험 분류에 따른 실제 문서 변경 내용은 <표 4>에 설명된 바와 같다.

실험 분류 1 : 형식적 태그 S와 비 형식적 태그 NS를 사용하여 문서를 변경한다.

실험 분류 2 : 형식적 태그 S와 비 형식적 태그 NS에 종료 태그가 없는 태그 NC를 조합하여 문서를 변경한다.

실험 분류 3 : 형식적 태그 S와 비 형식적 태그 NS에 종료 태그가 있는 태그 C를 조합하여 문서를 변경한다.

5.2 실험 결과

변경 탐지기는 자바(Java) 버전 1.3으로 작성하고 번역(compile)하였다. 변경 탐지기는 HTML 파서(parser)가 필요한데, 자바용 HTML 파서는 비 상업용으로 공개되어 있는 "adc" 파서를 HTML 버전 4.0으로 확장하여 사용하였다. 실험 데이터에서 패턴을 발견하기 위해 사용한 패턴의 반복 횟수는 10회로 하였다. 실험의 결과는 각 실험에서 얻어지는 HTML 페이지의 변경 탐지 성공률을 구한다. 변경 탐지 성공률은 변경 탐지에 성공한 횟수를 변경된 문서의 수로 나눈 비율이다.

$$\text{변경 탐지 성공률}(\%) = (\text{변경 탐지 성공 횟수} / \text{변경 문서의 개수}) \times 100$$

실험에서 변경 전략에 따라 대상 HTML 문서의 변경은 110개 문서의 패턴을 변경하기 위한 것이었으므로 모든 문서의 패턴이 변경되었다.

〈표 4〉 실험 분류에 따른 문서 변경 내용

실험 분류 및 사용 태그		설 명
실험 분류 1	기본 문서 변경 전략 적용 *S* = {"<I>, </I>"} *NS* = {"<Table>, </Table>"}	"<I>, </I>" 태그 삽입
		"<Table>, </Table>" 태그삽입
		"<I>, </I>" 태그 삭제
		"<Table>, </Table>" 태그삭제
		"<I>" 태그를 "<Table>"로, "</I>" 태그를 "</Table>" 태그로 변경
		"<Table>"태그를 "<I>"로, "</Table>" 태그를 "</I>" 태그로 변경
실험 분류 2	기본 문서 변경 전략과 NC 태그의 조합 *S* = {"<I>, </I>"} *NS* = {"<Table>, </Table>"} *NC* = {" "}	"<I> , </I>" 태그 삽입
		"<Table> , </Table>" 태그삽입
		"<I> , </I>" 태그 삭제
		"<Table> , </Table>" 태그삭제
		"<I>"와 " "을 "<Table>"로, "</I>"를 "</Table>" 태그로 변경
		"<Table>"과 " "을 "<I>"로, "</Table>" 태그를 "</I>" 태그로 변경
실험 분류 3	기본 문서 변경 전략과 C 태그의 조합 *S* = {"<I>, </I>"} *NS* = {"<Table>, </Table>"} *C* = {"<A>, </A>"}	"<I><A>, </A></I>" 태그 삽입
		"<Table><A>, </A></Table>" 태그삽입
		"<I>", "<A>", "</A>", "</I>" 태그 삭제
		"<Table>", "<A>", "</A>", "</Table>" 태그삭제
		"<I>, <A>" 태그를 "<Table>"로 "</I>, </A>" 태그를 "</Table>" 태그로 변경
		"<Table>, <A>" 태그를 "<I>"로, "</Table>, </A>" 태그를 "</I>"태그로 변경

실험은 세 분류로 나누어 18가지의 변경의 종류를 이용하여 수행되었다. 18 경우 모두 문서를 변경하여 문서의 패턴을 변경하였으며, 총 실험 횟수는 각 문서 당 여섯 가지의 변경 내용이 적용되고, 세 가지 실험 분류로 구분되므로 110(실험 문서 수) × 18가지(각 실험 분류에 대한 변경 내용) = 1,980회이며, 이에 대한, 문서 변경 발생 실험은 잘 수행되는 것을 <표 5>의 실험 결과를 통해 알 수 있다.

〈표 5〉 실험 결과

분 류	변경동작	전체 페이지 수	변경된 페이지 수	변경 탐지된 페이지 수	변경 탐지 성공률(%)
실험 분류 1	Is	110	110	110	100
	I	110	110	110	100
	Ds	110	110	110	100
	D	110	110	110	100
	Us	110	110	110	100
	U	110	110	110	100
실험 분류 2	Is	110	110	110	100
	I	110	110	110	100
	Ds	110	110	110	100
	D	110	110	110	100
	Us	110	110	110	100
	U	110	110	110	100
실험 분류 3	Is	110	110	110	100
	I	110	110	110	100
	Ds	110	110	110	100
	D	110	110	110	100
	Us	110	110	110	100
	U	110	110	110	100

6. 결론 및 향후 과제

본 논문에서는 메타 검색엔진의 목적 검색 사이트들의 특성을 이용하여 문서 변경을 탐지하는 방법을 제안하였고 문서의 패턴을 이용하여 문서 변경 탐지기를 설계하고 구현하였다. 문서 변경 탐지기는 결과 HTML 문서에 위치정보 알고리즘과 수정된 Jaak Vilo 알고리즘을 적용하여 패턴을 추출하고 비교한다. 구현된 문서 변경 탐지기의 기능을 시험하기 위해 문서 변경 전략을 세우고 실제 웹 페이지 110개에 대해 실험을 수행하였다. 실험을 통해 문서의 패턴이 변경되는 경우에 HTML 문서 변경 탐지기가 원활하게 작동됨을 검증하였다.

향후 과제로는 실제 웹사이트의 문서를 대상으로 문서 변경 탐지기의 기능성에 대한 평가가 요구된다. 이를 위해, 네트워크 환경을 통하여 실시간으로 문서의 변경을 탐지하는 장기적인 실험 계획을 수립하고, 실험 계획에 따라 래퍼(Wrapper)와 변경 탐지기의 결과를 비교하여 문서 변경 탐지기의 기능성 측정과 평가가 수행되어야 한다. 또한, 래퍼의 자동 규칙 변경을 위해 문서 변경 탐지기의 적용이 요구된다. 즉, 문서 변경 탐지기에 의해 생성된 패턴을 이용하여 변경된 HTML 문서의 구조를 메타 검색엔진의 래퍼의 정보 추출 규칙에 적용할 수 있다. 일반적으로 래퍼의 정보 추출 규칙의 수정은 자동화되어 있지 않고 수동으로 이루어지고 있다. 각 검색엔진의 규칙을 수정하기 위해서는 문서의 구조를 분석하고 그 결과를 규칙에 적용하므로, 메타 검색엔진의 효용성을 감소시키는 요인이 된다. 이를 해결하기 위해 문서 변경 탐지기에서 발견된 패턴을 정보 추출 규칙에 적용하게 되면, HTML 문서의 구조 변경 시에 정보 추출 규칙을 수정할 수 있다. 래퍼의 정보 추출 규칙의 자동 수정을 위해서는 패턴 발견기가 생성하고 있는 패턴들 중 한 가지가 규칙에 적용될 수 있으므로, 규칙에 적용되는 패턴을 선택하는 방법론이 제시되어야 한다.

참 고 문 헌

[1] D. Drelinger and A. E. Howe, "An Information Gathering Agent for Querying Web Search Engines," Technical Report CS-96-111, Department of Computer Science, Colorado State University.

[2] S. Lawrence and C. Giles, "Accessibility of Information on the Web," Nature, 400, pp.107-109, 1999.

[3] D. Dreilinger and A. E. Howe, "Experiences with Selecting Search Engines Using Metasearch," ACM Transactions on Information Systems(TOIS) 15(3), pp.195-222, 1997.

[4] E. Selberg and O. Etzioni, "Multi-Service Search and Comparison Using the MetaCrawler," Proceedings of the 4th International World Wide Web Conference, 1995.

[5] S. Lawrence and C. Giles, "Inquirus, the NECI meta search engine," Seventh Internatilnal World Wide Web Conference, pp.95-105, 1998.

[6] L. Gravano, and Y. Papakonstantinou, "Mediating and Metasearching on the Internet," Data Engineering Bulletin 21(2), pp.28-36, 1998.

[7] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change Detection in Hierarchically Strucrued Information," In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pp.493-504.

[8] N. Addam, I. Adiwijaya, T. Critchlow, and R. Musick, "Detecting Data and Schema Changes in Scientific Documents," In Proceedings of IEEE Advances in Digital Libraries 2000(ADL 2000), pp.160-172, 2000.

[9] K. Bohm, K. Aberer : "HyperStorM - Administering Structured Documents Using Object-Oriented Database Technology," (Demonstrator) Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD Record 25(2), pp.547, 1996.

[10] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From structured documents to novel query facilities," In Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, pp.313-324, 1994.
[11] T. Nguyen and V. Srinivasan, "Accessing relational databases from the World Wide Web," In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pp.529-540, 1996.
[12] J. T. L. Wang, K. Zhang, and D. Shasha. "Pattern matching and pattern discovery in scientific, program, and document databases". In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp.487, 1995.
[13] J. T. Wang, D. Shasha, G. J. S. Chang, L. Relihan, K. Zhang, and G. Patel, "Structural Matching and Discovery in Document Databases," Proceedings of SIGMOD '97, pp.560-563, 1997.
[14] J. Wang, K. Zhang, K. Jeong, and D. Shasha, "A System for Approximate Tree Matching," In IEEE Transaction on Knowledge and Data Engineering, volume 6, pp.559-570, 1994.
[15] J. Vilo, "Discovering Frequent Patterns from Strings," Technical Report C-1998-9, Department of Computer Science, University of Helsinki, 1998.
[16] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen, "Predicting Gene Regulatory Elements in Silico on a Genomic Scale," Genome Research 8, pp.1202-1215, 1998.
[17] A. Brazma, I. Jonassen, E. Ukkonen, and J. Vilo, "Discovering patterns and subfamilies in Biosequences," Proceedings of Fourth International Conference on Intelligent Systems for Molecular Biology(ISMB)-96, AAAI Press, pp.34-43, 1996.
[18] M. Crochemore and W. Rytter, "Text Algorithms," Oxford University Press, 1994.
[19] D. Gusfield, "Algorithms on Strings, Trees, and Sequences," Cambridge University Press, 1997.
[20] http : //www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Suffix.html.
[21] http : //www.cs.mcgill.ca/~cs251/OldCourses/1997/topic7/.
[22] D. Florescu, A. Levy, and A. Mendelzon, "Database Techniques for the World-Wide Web : A Survey," SIGMOD Record 27(3), pp.59-74, 1998.
[23] http : //www.ietf.org/rfc/rfc2616.txt.
[24] http : //www.w3.org/TR/1999/REC-html401-19991224/.

박 상 위

e-mail : ggypsy@nate.com
1998년 한남대학교 컴퓨터공학과 졸업(학사)
2001년 숭실대학교 대학원 컴퓨터학과(석사)
2001년~현재 (주)지텍 인터내셔널
관심분야 : 인터넷 데이타베이스, 데이타베이스 시스템 성능평가

오 정 석

e-mail : dbstar@orion.ssu.ac.kr
1996년 서경대학교 정보처리학과 졸업(학사)
1998년 숭실대학교 대학원 컴퓨터학과(석사)
1998년~현재 숭실대학교 컴퓨터학과 대학원 박사과정
관심분야 : 인터넷 데이타베이스, 데이타베이스 시스템 성능평가, 정보검색

이 상 호

e-mail : shlee@computing.soongsil.ac.kr
1984년 서울대학교 컴퓨터공학과(학사)
1986년 미국 노스웨스턴대학교 전산학과 (석사)
1989년 미국 노스웨스턴대학교 전산학과 (박사)
1990년~1992년 한국전자통신연구원 선임연구원
1992년~현재 숭실대학교 컴퓨터학부 부교수
1999년~2000년 미국 George mason 대학교 교환 교수
관심분야 : 인터넷 데이타베이스, 데이타베이스 성능평가, 트랜잭션 처리