MORPHEUS: 확장성이 있는 비교 쇼핑 에이전트
(MORPHEUS: A More Scalable Comparison-Shopping Agent)

양 재 영 † 김 태 형 †† 최 중 민 †††
(Jaeyoung Yang) (Taehyung Kim) (Joongmin Choi)

요 약 비교 쇼핑은 웹 상에 존재하는 웹 상점으로부터 구매를 원하는 상품에 대해 저렴한 가격을 찾아주는 일종의 판매자 중개 방법이다. 보다 쉽게 확장 가능한 비교 쇼핑 시스템을 생성하기 위해서 에이전트는 각각의 준 구조화된 상점으로부터 필요한 정보만을 추출할 수 있는 wrapper를 자동으로 생성해 낼 수 있어야 한다. 웹 문서를 작성하기 위한 HTML은 포함하고 있는 정보의 의미가 아닌 브라우저를 통한 정보의 표현에 대해서만 정의하고있다. 또한 각 웹 상점들은 사용자의 다양한 상품 검색 요구를 수용하기 위해 다양한 상품 검색 방법과 검색 결과의 출력 형태를 가진다. 따라서 자동으로 필요한 정보만을 추출하는 wrapper의 생성은 어려운 작업이다. wrapper의 귀납적인 생성은 이러한 이질적인 환경을 극복하기 위한 기술이다. 그러나 Shopbot과 같은 기존의 확장 가능한 비교 쇼핑 에이전트는 원하는 상품 정보를 추출하기 위해 강한 바이어스에 의존한다. 따라서 Shopbot은 바이어스를 따르지 않는 많은 웹 상점으로부터 wrapper를 생성할 수 없다.

본 논문에서는 강한 바이어스를 사용하지 않고 wrapper를 생성해 낼 수 있는 비교 쇼핑 에이전트 시스템인 모피우스를 제안하다. 모피우스는 간단하면서 견고한 학습 알고리즘을 바탕으로 wrapper를 생성한다. 제안하는 학습 알고리즘의 핵심은 상품 검색 결과를 논리적 라인으로 나누고 여기서 나타나는 상품 설명 단위의 패턴으로 wrapper를 생성하는 것이다. 모피우스는 대부분의 웹 상점에 대한 wrapper를 정확하게 생성해 낸다. 또한 학습하려는 검색 결과에 노이즈가 존재하는 경우에도 wrapper를 정확하게 추출할 수 있다. 모피우스는 헤더나 광고와 같은 불필요한 정보들을 제거하는 별도의 단계를 거치지 않으므로 wrapper를 빠르게 생성한다. 궁극적으로 모피우스는 새로운 웹 상점을 사용자가 자유롭게 추가, 삭제할 수 있는 환경을 제공한다.

Abstract Comparison shopping is a merchant brokering process that finds the best price for the desired product from several Web-based online stores. To get a scalable comparison shopper, we need an agent that automatically constructs a simple information extraction procedure, called a wrapper, for each semi-structured store. Automatic construction of wrappers for HTML-based Web stores is difficult because HTML only defines how information is to be displayed, not what it means, and different stores employ different ways of manipulating customer queries and different presentation formats for displaying product descriptions. Wrapper induction has been suggested as a promising strategy for overcoming this heterogeneity. However, previous scalable comparison-shoppers such as ShopBot rely on a strong bias in the product descriptions, and as a result, many stores that do not confirm to this bias were unable to be recognized.

This paper proposes a more scalable comparison-shopping agent named MORPHEUS. MORPHEUS presents a simple but robust inductive learning algorithm that automatically constructs wrappers. The main idea of the proposed algorithm is to recognize the position and the structure of a product description unit by finding the most frequent pattern from the sequence of logical line information in output HTML pages. MORPHEUS successfully constructs correct wrappers for most stores by weakening a bias

· 이 논문은 1998년 한국학술진흥재단의 학술연구비에 의하여 지원되었음
† 학생회원 : 한양대학교 컴퓨터공학과
jyyang@cse.hanyang.ac.kr
†† 정 회 원 : 한양대학교 컴퓨터공학과 교수
tkim@cse.hanyang.ac.kr
††† 종신회원 : 한양대학교 컴퓨터공학과 교수
jmchoi@cse.hanyang.ac.kr
논문접수 : 2000년 9월 25일
심사완료 : 2001년 1월 12일

assumed in previous systems. It also tolerates some noises that might be present in production descriptions such as missing attributes. MORPHEUS generates the wrappers rapidly by excluding the pre-processing phase of removing redundant fragments in a page such as a header, a tailer, and advertisements. Eventually, MORPHEUS provides a framework from which a customized comparison-shopping agent can be organized for a user by facilitating the dynamic addition of new stores.

1. 서 론

유용한 정보를 담고있는 웹의 팽창으로 사용자는 필요한 정보를 다양한 정보 소스에서 검색하고 수집하는데 어려움을 느끼게 되었다. 특히 온라인 쇼핑의 경우, 원하는 상품에 대한 최저가격을 찾는 소비자는 많은 웹 상점에 대한 검색 때문에 이를 포기하게 될 것이다. 더욱이 웹 상점들은 같은 사용자 인터페이스를 제공하지 않는다. 즉, 웹 상점들은 다양한 상품 검색 형태를 가지고 있으며 각기 다른 형태의 검색 결과를 출력한다. 결과적으로 웹 상점을 이용하려는 사용자는 각 웹 상점들의 입력 방식에 대해 숙지하고 있어야하며 상품 비교를 위해 각 상점으로부터의 상품 정보를 직접 수집하여야 한다.

비교 쇼핑 시스템은 사용자 대신에 웹 상점들로부터 자동으로 상품 정보를 수집하고 분석하는 중개자이다. 웹 상점들의 구조가 각기 다르기 때문에 wrapper는 각 웹 상점에 대해서 하나씩 만들어져야 한다. wrapper는 특정 정보 소스로부터 필요한 정보만을 추출하는 프로시져(procedure)이다. 따라서, 정보 소스에서 필요한 정보만을 추출하기 위한 코드나 규칙으로 구성되어 있다. 비교 쇼핑에 적용되는 wrapper는 각 웹 상점에서 제공하는 하나 이상의 상품정보 위치와 이를 추출하기 위한 규칙을 담고 있다.

Wrapper는 사람에 의해 수동 생성되거나 소프트웨어 에이전트에 의해 자동으로 생성될 수 있다. 사람에 의해 wrapper를 수동 생성하는 것은 단순하고 시간 소비적인 지루한 작업이다. 또한 비교 쇼핑 시스템의 확장이 어려울 수 있다. 새로운 웹 상점이 등장한 경우 비교 쇼핑 시스템의 관리자는 새로운 웹 상점을 방문하고 이에 맞는 상품정보 추출 규칙을 만들어 주어야 한다. 에이전트에 의해 자동 생성되는 wrapper는 에이전트가 HTML로 작성된 정보를 분석해야 한다. HTML은 정보를 어떻게 사용자에게 출력해줄 것인지만을 정의하고 있기 때문에, HTML에 포함된 정보의 의미는 사용자가 판단해야 한다. 그러나 다행히도 온라인 웹 상점들은 준구조화(semi-structured)되어 있어 자연어 처리와 같은 의미 기반 분석 방법을 사용하지 않고도 필요한 정보를 추출할 수 있다.

Wrapper 자동 생성의 또 다른 문제점으로는 앞에서 언급한 웹 상점들의 이형질(Heterogeneity)성에 있다. 각기 다른 웹 상점은 사용자의 상품 검색 질의를 처리하기 위해 다양한 방법을 사용하고 있으며 검색 완료된 상품정보를 다양한 방법으로 표현한다. 예를 들면, 몇몇 상점들은 상품의 키워드만을 입력받기 위해 하나의 입력 창을 가지는 반면에 웹 서점의 경우 "서적명", "ISBN" 등과 같이 다중의 입력을 사용자로부터 받아 처리한다. 이러한 입력의 결과로서 웹 상점들은 3절에서 보다 자세하게 설명하게될 테이블형이나 리스트형 검색 결과를 사용자에게 제공한다.

Wrapper induction[1]은 이러한 이형질성을 극복하기 위한 방법을 제공한다. wrapper induction은 정보 소스의 예제 페이지로부터 추론을 통해 각 정보 소스에 맞는 wrapper를 생성한다. 이것은 귀납적 학습 방법이며 하나의 웹 상점의 인스턴스로부터 개념을 학습한다. wrapper induction을 통해 wrapper를 자동 생성하는 ShopBot[2]과 같은 시스템들이 등장하였다. 그러나 Shopbot을 포함한 대부분의 기존 연구에서의 학습 알고리즘들은 정보 소스에서 발생할 수 있는 노이즈에 약하며 또한 강한 바이어스를 사용한다. ShopBot은 한 상품의 정보들, 즉 본 논문에서 제시하는 PDU의 요소들이 한 논리적 라인에 모두 존재해야 한다는 바이어스를 사용한다. PDU가 다중의 논리적 라인에 존재하는 경우 그 상품에 대한 정보를 추출할 수 없다. ShoptBot에서 PDU는 공백라인을 이용하여 인식하는 바이어스를 사용한다. 하나의 PDU 다음에는 공백라인이 존재해야 하며 그 다음에 다른 PDU가 나타나야 한다. 또한 ShopBot은 PDU가 모두 같은 형태로 나타나야 한다는 바이어스를 사용한다. 이것은 한 상품 검색결과 페이지에 나타나는 모든 PDU의 형태가 같을 때에만 상품 정보를 추출할 수 있음을 의미한다. 이러한 강한 바이어스의 사용으로 ShopBot은 실세계의 웹 상점들에 학습 알고리즘을 적용하는 경우 상품 정보를 추출하지 못하는 경우가 많이 발생한다.

본 논문에서는 확장성을 고려한 비교 쇼핑 에이전트 시스템인 모피우스를 제안한다. 모피우스의 학습 알고리즘은 간단하지만 견고성이 뛰어나다. 새로운 웹 상점을

학습하는 단계에서 모피우스는 사용자가 제공하는 웹 상점의 URL을 바탕으로 질의 템플릿을 생성해내고 상품설명단위(Product Description Unit: PDU)의 패턴을 추출한다. 학습 알고리즘의 핵심은 HTML를 논리적인 단위로 나눈 후 발생 빈도에 따라 PDU를 선택하고, 이 PDU를 바탕으로 추출을 시작해야하는 첫 번째 위치와 마지막 위치를 파악해 내는 것이다. 이 PDU의 패턴이 wrapper의 정보 추출 규칙이 된다.

모피우스 시스템의 최종 목적은 개인 사용자가 쉽고 빠르게 자신만을 위한 비교 쇼핑 에이전트 시스템을 생성하도록 도와주는 것이다. 따라서 사용자마다 다른 웹 상점들에 대한 비교 쇼핑을 수행하게 된다. 이러한 시스템에서는 효율성과 안정성, 그리고 유연성을 바탕으로하는 wrapper induction 시스템이 필요하다. 모피우스는 이러한 요구사항을 모두 수용하고 있으며 이를 검색 결과의 만족도를 통해서 알 수 있다. 모피우스에 새로운 웹 상점을 추가하기 위해서는 단지 분석을 하려는 웹 상점의 URL을 입력해 주기만 하면 된다. 이 기술은 개인 메타 검색 엔진과 같이 개인화된 정보 통합 시스템에 적용할 수 있다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 모피우스 시스템의 전체적인 시스템 구조와 기능에 대해 설명한다. 3장에서는 wrapper 학습 알고리즘에 대해서 설명한다. 학습 알고리즘에는 PDU의 학습뿐만 아니라 웹 상점의 질의 템플릿에 대한 학습 방법에 대해서도 언급한다. 또한 wrapper의 실행부분에 대해서 설명한다. 4장에서는 모피우스의 실험결과를 제시하고, 5장에서는 다른 wrapper induction 시스템들과 비교를 하며, 6장에서는 결론 및 모피우스 시스템의 개선 방향을 제시하고자 한다.

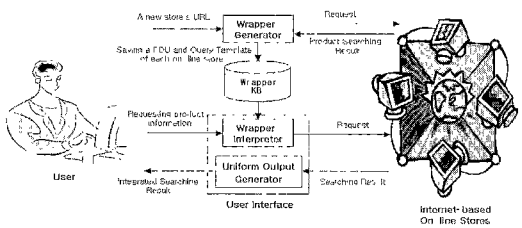

그림 1 모피우스의 구조

2. 모피우스 시스템 구조

개인화된 비교 쇼핑 에이전트 시스템인 모피우스는 그림 1과 같이 wrapper 생성기(Wrapper Generator), wrapper 실행기(Wrapper Interpreter), 상품 검색 결과 통합기(Uniform Output Generator), wrapper 지식 베이스로 이루어져 있다.

다양한 웹 상점들로부터 필요한 상품 정보를 추출하여 사용자가 원하는 상품 비교를 하기 위해서는 에이전트는 세 가지 기능을 가지고 있어야 한다. 첫째, 에이전트는 각 웹 상점들이 사용하는 상품 검색 질의문의 형태를 알아야 한다. 웹 상점들은 사용자의 상품 검색 요청을 처리하기 위해 CGI를 통해서 데이타베이스에 SQL문을 전달한다. 에이전트는 데이타베이스에 전달하게 되는 질의문의 형태를 미리 알고 있어야 한다. 둘째, 에이전트는 각 웹 상점들로부터 PDU 추출을 위해 PDU 패턴을 학습해야 한다. PDU는 하나의 상품을 설명하기 위해 웹 상점에서 사용하는 단위로 정의할 수 있으며 각 웹 상점마다 상품 설명을 위해 사용하는 PDU의 구조는 다르다. 셋째, 에이전트는 패턴을 이용하여 추출한 상품 정보들을 하나의 단일한 형태로 통합해야 한다. 각 웹 상점들은 사용자의 상품 질의에 대해서 각기 다른 검색 결과를 출력한다. 예를 들어, 상점 A에서는 상품명, 제조사, 가격에 대해 출력하는 반면에 상점 B에서는 상품명, 규격, 가격을 출력하는 경우 상점 A와 B의 검색 결과를 어떻게 통합할 것인지를 파악해야 한다.

그림 1에서 에이전트는 두 가지 입력에 대해서 동작한다. 첫째는 상품 질의어 및 PDU 학습을 위해 새로운 웹 상점의 URL을 입력받아 학습하는 경우이다. 사용자가 새로운 웹 상점의 URL을 입력하면 에이전트는 URL에서 연결된 페이지들을 탐색하면서 검색창이 있는 페이지를 찾아낸다. 만약 검색창이 하나이상 존재하면 가장 좋은 검색 결과를 출력하는 검색창을 찾아낸다. 이 검색창을 분석하여 질의어 템플릿을 만든다. 질의어 템플릿을 이용하여 웹 상점에 샘플 질의어를 만들어 검색 요청을 한다. 에이전트는 검색 결과로부터 PDU의 패턴을 학습하게 된다. 질의어 템플릿과 PDU 학습이 완료되면 wrapper 지식베이스에 웹 상점을 추가하게 된다. wrapper 지식베이스는 wrapper 실행기에 의해 참조된다. 둘째는 기존의 학습한 웹 상점들을 이용하여 비교 쇼핑을 행하는 경우이다. 사용자는 wrapper 실행기에 자신이 원하는 상품 검색어를 입력한다. wrapper 실행기는 wrapper 지식베이스에 명시되어 있는 질의어 템플릿을 이용하여 상품 질의를 한다. wrapper 실행기는 각 웹 상점들의 검색 결과를 수집하여 이를 검색 결과 통합기에 제공한다. 검색 결과 통합기는 wrapper 지식베이스에 명시된 각 웹 상점들의 PDU 패턴을 바탕으로

상품 정보를 모으게 된다.

3. Wrapper의 학습 및 실행

일반적으로 웹 상점으로부터 상품 정보를 추출하기 위해서는 풍부한 온톨로지가 필요하다. 풍부한 온톨로지는 에이전트로 하여금 쉽고 정확하게 필요한 상품 정보를 추출하게 만든다. 그러나 현실적으로 풍부한 온톨로지의 구축은 어려운 문제이며 또한 풍부한 온톨로지라 할지라도 새로운 웹 상점이 현재의 온톨로지를 따른다는 보장을 할 수 없다. 따라서 온톨로지를 되도록 적게 사용하는 학습 알고리즘을 설계하는 것이 유리하며, 본 논문에서 제시하는 학습 방법도 같은 맥락에서 연구되었다.

3.1 검색 결과의 종류

웹 상점의 학습이란 웹 상점에서 제공하는 상품 검색 결과에서 필요한 정보만을 추출할 수 있는 규칙의 학습, 다른 말로 말하면 PDU 패턴을 학습하는 일련의 과정이라고 정의할 수 있다. 따라서 학습 알고리즘을 설계하기 전에 웹 상점들이 제공하는 검색 결과의 형태를 파악해야 한다.

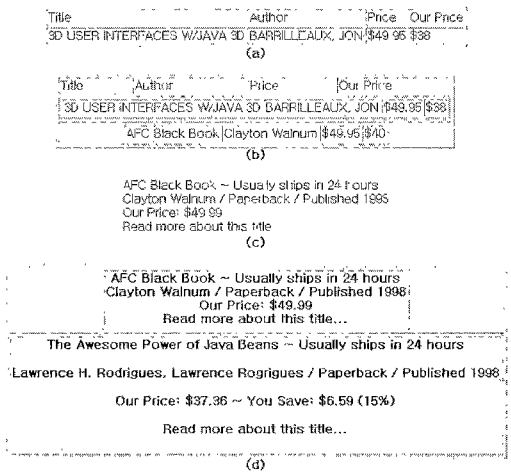

그림 2 검색 결과의 종류

웹 상점에서 제공하는 검색 결과의 형태는 나열할 수 없을 정도로 매우 다양한 형태를 가진다. 그 중 가장 많이 나타나는 형태를 살펴보면 다음과 같이 분류할 수 있다.

- 테이블형 검색 결과: 테이블형 검색 결과의 형태는 그림 2의 a)와 같이 구조화된 형태를 가진다. 즉, 각 문자열이 어떤 의미를 가지는지를 나타내는 온톨로지가 테이블 헤더에 포함되는 형태를 말한다. 대부분의 웹 상점들이 테이블 형태로 검색 결과를 사용자에게 제공한다.
- 중첩 테이블형 검색 결과: 그림 2의 b)와 같은 형태를 가지는 검색 결과이다. PDU 전체 집합이 HTML의 테이블 태그에 포함되어 있으면서 각각의 PDU들이 또 한번 HTML의 테이블로 분리되는 경우이다. 이 경우 온톨로지가 있어도 크게 도움이 되지 않는다.
- 리스트형 검색 결과: 테이블형 검색 결과를 제외한 나머지 형태 중 HTML의 테이블 태그로 PDU가 분리되지 않는 검색 결과를 말한다. 리스트형이란 이름을 붙인 이유는 테이블 형태의 검색 결과를 제외한 나머지 검색 결과 형태가 대부분 상품 속성들을 나열하는 식으로 되어 있기 때문이다. 리스트형인 경우 각 상품 속성들이 가지는 문자열의 의미가 무엇인지를 나타내는 온톨로지가 존재하지 않는다. 그림 2의 c)는 리스트형 검색 결과이다.
- 테이블을 포함하는 리스트형 검색 결과: 그림 2의 d)와 같이 전체 출력은 리스트형이지만 각 PDU가 HTML의 테이블 태그로 구별되는 경우이다.

위의 분류에서 알 수 있듯이 리스트형과 테이블형을 분류해내는 작업 또한 많은 계산을 요한다. 다양한 검색 결과로부터 가장 일반적인 검색 결과 형태를 도출하면 크게 테이블형과 리스트형으로 나누어 볼 수 있다.

웹 상점의 검색 결과가 구조화된 검색 결과를 사용자에게 제공하는 경우 에이전트는 검색 결과에 나타나는 구조적인 특성을 이용하여 필요한 정보를 쉽게 추출할 수 있다. 예를 들어 그림 3과 같이 구조화된 도서 검색 결과가 있다고 가정하자.

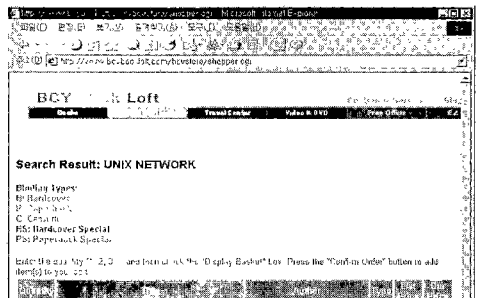

그림 3 웹 상점의 검색 결과

에이전트는 UNIX Network MGMT TOOLS 라는 책의 정보(Title, Author, Type, List Price, Our Price)를 알아내기 위해 각 속성들이 어떤 의미를 나타내는지를 파악해야 한다. 그림 3은 매우 구조적인 검색 결과를 가지고 있어서 각 상품 속성들에 대한 서술자(descriptor)가 붙어 있다. 따라서 에이전트는 Steve 라는 문자열이 가지는 의미가 서술자 Author 에 의해 저자임을 알 수 있게 된다. 이러한 서술자의 집합을 온톨로지라고 부른다. 에이전트는 검색 결과에서 인식한 의미들을 규칙화하여 상점에 대한 학습을 끝내게 된다.

하지만 이 추출방법은 도메인 지식에 웹 상점에서 사용하는 온톨로지에 대한 정보가 존재하지 않으면 그 상점에서 원하는 상품 정보를 추출할 수 없게 된다. 본 논문에서는 이러한 온톨로지 정보 없이도 테이블형 검색 결과와 리스트형 검색 결과로부터 PDU 패턴을 학습할 수 있는 generic algorithm을 설계한다.

3.2 사용자 질의를 위한 입력 창의 분석

에이전트는 웹 상점이 제공하는 입력 폼을 분석하여 자동적으로 질의어 템플릿을 생성해 낼 수 있어야 한다. 입력 폼 분석을 위해 먼저 해결해야 하는 문제가 있다. 첫째, HTML 태그 중에서 어떤 태그를 분석 기준으로 할 것인지를 결정해야 한다. 일반적으로 HTML에서 사용자로부터 키워드 입력을 받아 검색을 수행하기 위해서는 입력 유형이 TEXT인 입력 태그를 사용한다. 따라서 에이전트는 HTML의 TEXT 입력 유형을 기준으로 입력 폼을 분석하게 된다. 둘째, 입력 유형이 TEXT인 입력 태그가 다중으로 존재하는 경우에 어떤 태그를 중심으로 입력 폼을 분석할 것인지를 결정할 수 있어야 한다. 다중의 입력 폼 중에서 에이전트는 각 입력 폼에 테스트 질의를 하고 그 결과를 평가하여 가장 좋은 검색 결과가 나오는 입력 폼을 선택하게 된다. 이 방법은 기존의 온톨로지에 의존하여 입력 폼을 분석하는 것보다 계산시간이나 정확도면에서 약간 성능이 떨어질 수 있지만 온톨로지가 구축되어 있지 않은 웹 상점을 분석할 수 있는 장점이 있다. 셋째, 입력 태그 중 선택사항에 대한 처리를 어떻게 할 것인지를 결정해야 한다. 몇몇 웹 상점들은 HTML의 <OPTION> 태그를 이용하여 상품 분류를 선택하게 한다. 예를 들어 사용자가 컴퓨터에 대한 상품 검색을 하려는 경우 먼저 선택사항의 전자제품군을 선택한 후 검색 키워드를 입력 받아 처리하는 경우이다. 이는 보다 상세한 제품 검색을 위해 사용된다. 선택사항에 대한 처리는 온톨로지를 이용해서만 처리될 수 있다. 일부 웹 상점들은 <OPTION>에서 제공하는 SELECTED를 이용하여 검색을 할 수 있게 지원하지만 그렇지 않은 웹 상점도 존재하기 때문이다.

웹 상점의 입력 폼이 다중으로 존재하는 경우 입력 폼을 평가해야 한다. 입력 폼이 하나 이상이면 여러 상품 속성으로 검색을 할 수 있다. 만약 비교 쇼핑 에이전트가 사용자로부터 상품명 만을 입력받아 상품 검색을 한다고 가정하자. 에이전트는 다중의 입력 폼 중에서 입력 유형이 TEXT인 폼을 찾아낸다. 각각의 입력 폼에 결과를 미리 알 수 있는 검색어로 질의를 한다. 결과를 미리 알 수 있다는 것은 검색 결과에 PDU가 포함되어 있는 정상적인 검색 결과를 의미한다. 검색 결과 중에서 가장 많은 PDU를 포함하고 있는 페이지를 찾고 이러한 검색 결과가 나오게 만든 입력 폼을 기준으로 질의어 템플릿을 만든다.

```
-post
http://www.amazon.com/exec/obidos/search-handle-form/
002-7739636-2540219 -form "index=books"
"field-keywords=INPUT_TEXT" "Go=Go"
```

그림 4 아마존의 질의 템플릿

그림 4는 실제 아마존 웹 페이지에서 추출한 질의 템플릿이다. 아마존의 경우 HTTP의 REQUEST 방법 중 "POST"방법만을 제공하게 되며 실제 사용자가 책을 찾기 위해 사용하는 질의어는 "INPUT_TEXT"로 나타나 있는 부분과 대체된다.

3.3 상품 속성 단위(PDU)의 인식

리스트형 학습 알고리즘은 다음과 같은 규칙성(regularity)를 가정한다.

- 표현의 동일성(uniformity): 각 상점들의 데이타베이스에서 상품 정보를 사용자에게 보여주는 방법은 다양하나 한 상점이 각각의 검색어로 상품 정보를 사용자에게 출력하는 경우 표현 방법은 같다.
- 상품 정보 단위(PDU: Product Description Unit): 상품에 대한 정보는 상품이 가지고 있는 속성(모델명, 가격, 규격, 제조사 등)을 바탕으로 상품 정보 단위로 나눌 수 있다.
- 상품 정보 단위의 가격 포함: 각 상품 정보의 단위에는 그 상품의 가격 정보가 반드시 포함되어 있어야 한다.

리스트형 검색 결과는 테이블형 검색 결과를 제외한 모든 상품 정보 단위로 나눌 수 있는 검색 결과 형태를 말한다. 이것은 일반화하면 다음과 같다.

α와 β는 각각 헤더(header)와 테일(tail)로 불리며 현재 사용자가 원하는 검색 결과와 무관한 여분의 정보이

그림 5 서로 다른 검색어를 통한 검색 결과

다. 그리고 ε 은 실제 추출해야 하는 정보를 담고 있는 정보 단위이며 이를 상품 설명 단위(PDU)라고 부른다. 이 정보 단위는 일정한 형태의 반복적인 구조를 가진다. ε 은 상품 정보를 설명하는 속성(attribute)의 집합이다. 각 속성은 μ로 표현한다. μ는 도서인 경우 도서명, 저자, 출판사, 가격 의 정보를 담고있는 정보 속성이다. 따라서 리스트형 학습 알고리즘은 하나의 상품에 대해 그 상품을 설명할 수 있는 상품 정보 단위 ε을 찾아내는 것을 목적으로 한다. 예를 들면 그림 5는 같은 상점에 대해 "UNIX"라는 검색어와 "MFC"라는 검색어를 통해 검색한 결과이다.

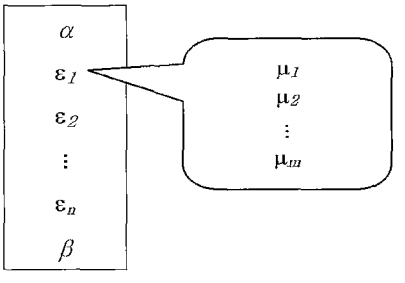

그림 5에서 α에 해당하는 부분이 "1"전까지이다. 이 부분은 어떤 검색어를 입력하더라도 똑같이 나타나는 결과이므로 사용자가 원하는 상품에 대한 검색 결과가 아니다. ε에 해당하는 부분은 "1"부터 시작해서 박스안에 나타나는 정보를 말한다. ε은 속성들로 이루어져 있는데 이 속성들 중에도 반복되는 요소들이 있다. 즉 검색 결과의 순서를 알기 위한 "1"과 주문을 위한 버튼 등은 반복된다. 그러나 이것은 α나 β가 될 수 없고 한 정보의 단위로 취급해야 한다. β는 검색 결과 후미에 붙는 여분의 정보를 말한다. 리스트형 학습 알고리즘에서 가장 중요한 부분은 바로 정확한 PDU 즉, ε의 인식이다. ε은 앞에서 언급했듯이 추출한 정보들을 담고 있는 하나의 정보 단위이다. 따라서 ε을 잘못 인식하게 되면 잘못된 정보를 추출하게 된다. ε은 속성들의 반복적인 패턴으로 구별된다. 이것은 다음과 같은 전처리를 거친다.

● 단위 인식을 위한 논리적 라인 처리: 상점에서 패턴은 상품에 대한 정보를 어떤 내용과 순서로 보여 주는지를 나타낸다. 그러므로 똑같은 내용이라도 어떻게 파일의 내용을 분리하는가에 따라 패턴은 달라지게 된다. 그러므로 결과파일을 분리하는 기준이 있어야 하는데 이것의 기준은 브라우저를 통해서 사용자에게 보여지는 형태를 기준으로 한다. 즉 HTML의 공백(blank) 태그나 이에 준하는 태그들을 이용하여 태그가 발생하기 이전까지를 하나의 의미단위로 판단하게 된다.

● 패턴 생성을 위한 논리적 라인의 의미파악: 단위 인식을 위한 라인 처리가 된 파일은 초기 HTML 파일을 재구성한 것이므로 그 자체만으로 패턴을 찾아내는 것은 불가능하다. 패턴을 찾기 위해서는 각 라인이 무엇을 의미하는지 파악해야 한다. 여기서 각 라인은 공백 태그를 기준으로 다음 공백 태그가 나올 때까지 즉 화면상에서 사용자가 볼 수 있는 라인을 의미한다. 패턴생성의 효율을 높이기 위해 각 라인의 의미를 파악한 후 이것을 숫자를 이용해서 패턴을 생성하게 된다. 각 숫자의 의미는 추출하고자 하는 정보의 종류에 따라 다르게 나타난다. 도서인 경우 라인의 의미를 도서명, 가격, 공백, 출판사의 네 가지로 나눌 수 있다. 도서명인 경우 검색결과를 논리적인 라인단위로 나누고 이 라인에 도서명에 해당하는 키워드가 발생하면 그 라인을 도서명으로 인식한다. 가격의 경우 정규 규칙을 통해서 인식한다. 공백의 경우 공백 태그를 이용해서 인식하며 출판사의 경우 반복되는 문자열의 인식을 통해서 간접적으로 인식할 수 있다.

이런 전처리 과정을 통해서 패턴들이 표현된 파일을 얻을 수 있게 된다. 패턴을 인식하기 위한 검색(search) 작업에서는 검색 속도를 높이기 위해 가격 정보를 기준으로 패턴을 찾아낸다. 이런 과정을 통해서 얻은 패턴은 그 패턴에 나타나는 정보의 종류에 따라 규칙 실행기를 통해서 추출되게 된다. 이와 같은 방법의 특징은 온톨로지 없이 반드시 필요한 정보를 정확히 추출해 낼 수 있다. 즉, 사용자에게 반드시 제공해야 하는 검색 결과인 제품명, 가격과 같은 기본 정보를 온톨로지 사용없이 준

구조화된 검색 결과 환경에서 정확하게 추출할 수 있다. 또한 패턴의 사용은 상품 정보 단위의 작은 변화에 민감하게 반응하지 않는다. 현재 페이지에서 나타나는 반복 횟수가 일정한 임계치를 넘으면 패턴으로 인식하게 하여 그 정확도를 높인다. 설명한 학습 알고리즘의 의사코드는 그림 6과 같다.

```
seq ← the input sequence of logical line categories,
seqStart ← 1; /* initial position for pattern search */
numCandPDUs ← 0, /* number of distinct PDU pattern */
while (true) {
        /* find the next candidate PDU */
        priceIndex ← findIndex(seq, seqStart, PRICE);
        pnameIndex ← findIndexReverse(seq, priceIndex, PNAME);
        currentPDU ← substring(seq, pnameIndex, priceIndex);
        /* no more PDUs */
        if (currentPDU == NULL) then exit the whole loop;
        /* count the frequencies of distinct PDU patterns */
        if (currentPDU is already stored in candPDUs array)
        then the frequency count of currentPDU is incremented by 1,
        else { /* a new PDU pattern*/
                save currentPDU in candPDUs array,
                increment numCandPDUs by 1, }
        seqStart ← priceIndex+1, /* starting point for searching the next PDU */
}
mostFreqPDU ← the element of candPDUs array with maximum frequency;
return(mostFreqPDU),
```

그림 3 학습 알고리즘의 의사코드

이 알고리즘을 실제 적용해 보면 다음과 같다. 아마존(http://www.amazon.com)에 적용한 경우 검색 결과는 그림 7과 같다. 검색어는 "Machine Learning"이다. 학습 알고리즘에 의해 각 라인은 표 1과 같이 분류된다. 그림 7의 경우 '3202120202'패턴이 반복적으로 일어나 우리가 찾고자 하는 정보 단위인 ε을 정확하게 찾아낸다. 이 학습 알고리즘에서 속성을 파악할 수 있는 함수를 더 정교하게 만들면 많은 정보를 정확하게 분류 및 추출해 낼 수 있다.

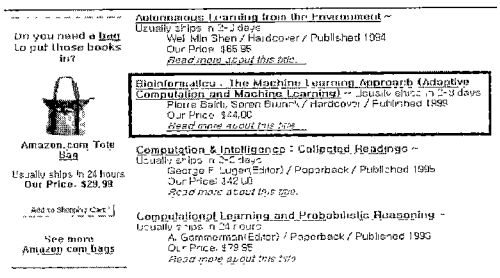

그림 7 아마존 검색 결과

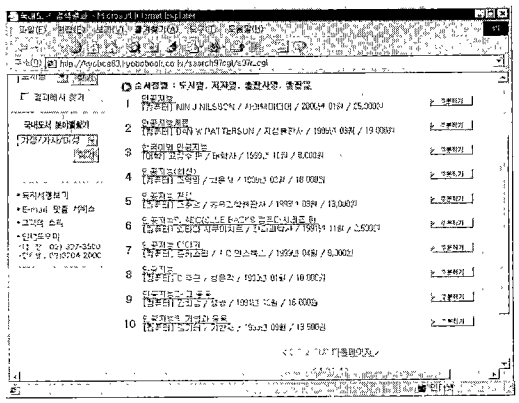

그림 8 테이블형 검색 결과의 PDU들

표 1 아마존 검색 결과의 패턴

<a href=CGI address>Bioinformatics: The ·	3
<dd>	2
Pierre Baldi, Soren Brunak / Hardcover ·	0
 	2
Our Price: $44.00	1
 	2
Average Customer Review ·	0
 	2
<a href=CGI address> Read more about ·	0
 	2

도서인 경우 패턴에서 '3'은 도서명을 의미하며 '2'는 논리적 라인으로 나누는 근거인 HTML의 줄 나눔 태그이며 '0'는 도메인 지식이 있는 경우 의미 파악을 할 수 있지만 현재 상태로는 알 수 없는 정보임을 나타낸다. 그리고 '1'은 가격 정보임을 나타낸다.

3.4 검색 결과로부터의 패턴 추출

리스트형 검색 결과는 테이블형에서 나타나지 않는 문제점을 가지고 있다. 테이블형 검색 결과는 PDU에 노이즈(noise)가 존재하지 않으며, 항상 일정한 형태를 가지게 된다. 그림 8은 테이블형 검색 결과의 PDU들이다. 각각의 PDU가 가지는 상품 속성들은 고정되어 있으며 중간에 없거나 속성 자체가 존재하지 않는 경우가 존재하지 않는다.

그러나 리스트형 검색 결과는 PDU에 노이즈가 존재하는 경우가 발생하게 된다. 그림 9는 노이즈가 존재하는 PDU를 나타낸다. 그림 9의 PDU에는 다음과 같은 노이즈가 존재한다.

- 문자열의 동의어 미처리로 인한 속성 인식 오류 문제
- 속성 중 일부가 PDU에 존재하지 않는 경우
- 속성 중 일부가 PDU에 여분으로 존재하는 경우

● 속성 중 일부가 PDU에 여분으로 존재하는 경우

노이즈가 존재하는 PDU사이에서 실제 정보 추출을 위해 사용되어야 할 PDU의 선택은 쉬운 일이 아니다. 만약 정보 추출에 사용될 적절한 PDU를 찾아낸다면 노이즈가 존재하는 다른 PDU에서도 필요한 정보를 추출할 수 있을 것이다.

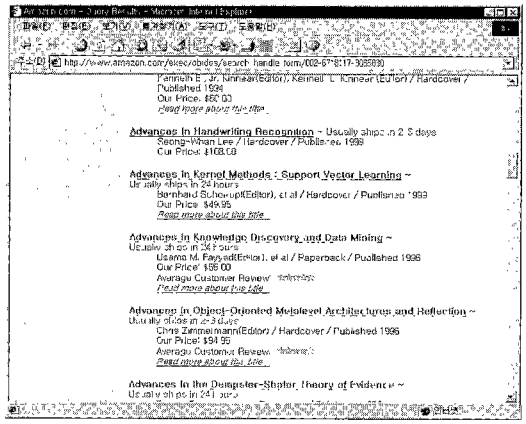

그림 9 PDU에 노이즈가 존재하는 리스트형 검색 결과

그림 10 첫 번째 후보 PDU

그림 9에는 세 가지의 PDU후보가 존재한다. 첫번째 후보는 그림 9와 같은 세 가지 상품 설명 속성으로 이루어진 PDU이다. 두번째 후보는 그림 11과 같이 네가지 상품 설명 속성으로 이루어진 PDU이다. 세번째 후보는 그림 12와 같이 다섯가지 상품 설명 속성으로 이루어진 PDU이다. 만약 검색 페이지에 다양하게 나타날 수 있는 모든 PDU 후보에 대해서 실제 정보 추출에 사용될 PDU가 될 확률이 uniform distribution을 따른다고 가정하면 n개의 PDU 후보는 $\frac{1}{n}$의 PDU가 될 기회를 가지게 된다. 본 논문에서는 $\frac{1}{n}$을 실제 정보 추출을 위한 최소기준으로 보고 이를 넘는 PDU를 실제 정보 추출을 위해 사용할 PDU로 인식하게 하였다. 만약 PDU가 될 확률이 비슷하게 나타나게 되면 이 PDU중 가장 많은 상품 속성을 가진 PDU를 상품 정보 추출 PDU로 인식하게 된다. 따라서 그림 9에서는 전체 PDU 중에서 33%를 넘는 PDU를 실제 정보 추출에 사용할 PDU로 인식한다. 아마존의 경우 세 PDU 후보 중 세번째의 PDU가 실제 정보 추출을 위해 사용되는 PDU가 된다.

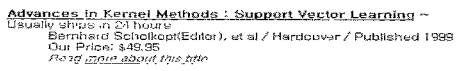

그림 11 두 번째 후보 PDU

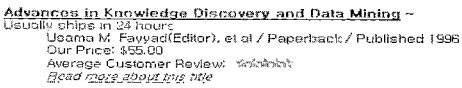

그림 12 세 번째 후보 PDU

이렇게 선택된 PDU를 제외한 나머지 PDU 후보들은 노이즈로 인식된다. 즉, 추출할 정보가 완전하지는 않지만 PDU 후보안에 존재하는 정보만이라도 반드시 추출해야 하기 때문에 선택된 PDU를 이용하여 다른 노이즈가 존재하는 PDU를 보정한다. 정보 추출 PDU보다 많은 상품 속성 노이즈가 가진 PDU에서는 이를 제거하고, 이 보다 적은 상품 속성 노이즈를 가진 PDU에는 시스템이 식별할 수 있는 임의의 메시지를 추가하여 패턴을 수정할 수 있다.

3.5 Wrapper의 실행

웹 상점의 학습이 완료되면 개인 프로파일에는 상품 검색을 시도할 상점의 질의어 템플릿과 PDU가 존재한다. 질의어 템플릿과 PDU 그리고 상품 검색 결과의 통합 규칙이 wrapper가 된다. wrapper 실행기는 각 웹 상점에 상품 검색을 요청하고 웹 상점으로부터의 상품 검색 결과를 파일로 저장하게 된다. 저장된 파일을 논리적 라인으로 분리하고 각 라인이 가지는 의미를 파악하여 기록한다. 검색 결과로부터 상품 정보를 추출하기 위해서 에이전트는 프로파일에 저장된 상점의 PDU와 비

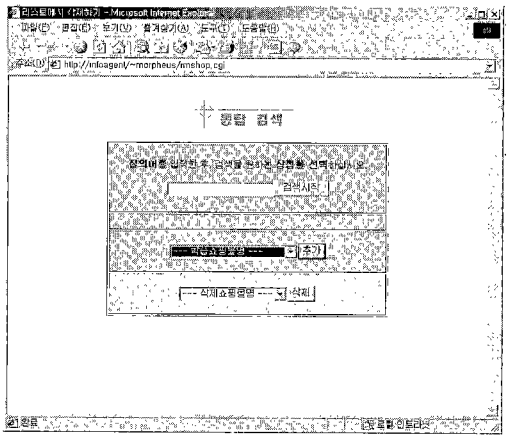

그림 13 모피우스의 초기 상태

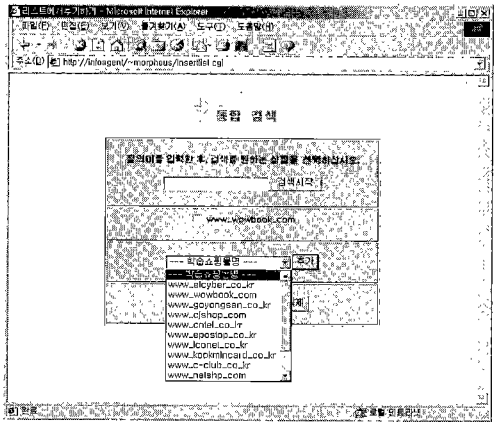

그림 14 웹 상점의 추가

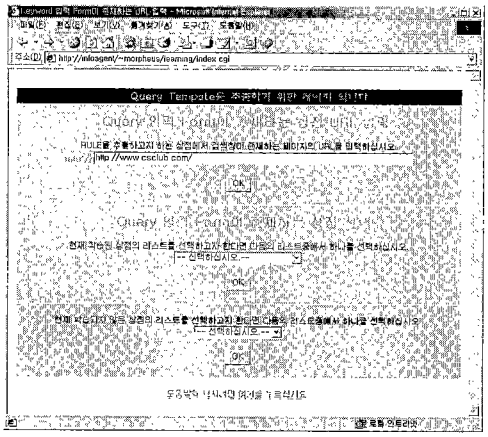

그림 15 웹 상점 학습 과정 : 웹 상점의 질의창의 URL 입력

교하여 PDU와 같은 패턴을 추출한다. 추출된 패턴이 사용자가 검색하기를 원한 상품의 상품 성명 단위인 PDU가 된다. 그림 13은 처음 사용자가 모피우스에 접속할 때의 모습이다. 그림 13의 하단부에 두 개의 리스트 상자("학습쇼핑몰명", "삭제쇼핑몰명")가 있는데, 첫 번째는 모피우스가 가지고 있는 학습된 쇼핑몰들에 대한 질의어 템플릿들이다. 이 리스트에 사용자가 원하는 쇼핑몰이 존재하면 그림 14와 같이 바로 추가 할 수 있다. 두 번째 리스트 상자는 선택한 쇼핑몰을 개인 프로파일에서 삭제한다.

만약 "학습쇼핑몰명" 리스트에 사용자가 원하면 웹 상점이 존재하지 않으면 사용자는 그림 15와 같이 웹상점 학습 에이전트를 이용하여 새로운 웹 상점의 wrapper를 생성하게 된다. 사용자는 웹 상점 학습 에이

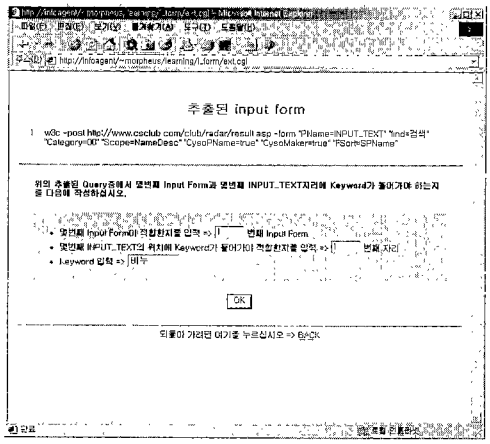

그림 16 웹 상점의 학습 과정 : 샘플 질의어 입력

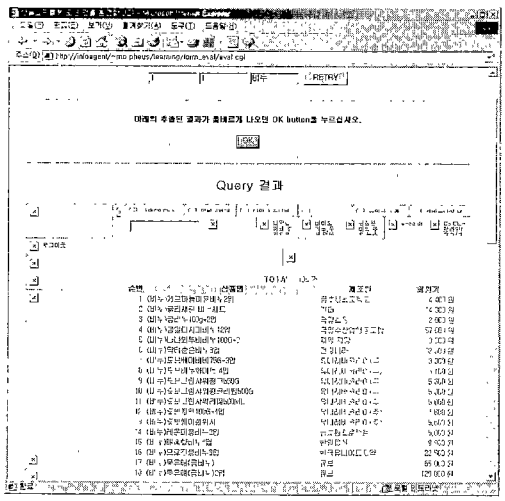

그림 17 웹 상점 학습 과정 : 질의어 템플릿 생성이 성공한 경우

전트에 질의창이 있는 웹 상점의 URL을 입력한다. 학습 에이전트는 입력된 URL을 바탕으로 질의 템플릿을 만들기 위해 사용자로부터 샘플 질의어를 그림 16와 같이 입력받게 된다. 그림 16에서 사용자로부터 세 가지의 입력을 받게 된다. 첫 번째는 HTML 소스에 다중의 FORM 태그가 나타나는 경우 어떤 FORM 태그를 선택해야 하는지에 관한 것이며 두 번째는 입력 창이 다중으로 존재하는 경우 몇 번째 입력 창에 샘플 질의어를 입력할 것인지를 묻는다. 세 번째 입력 창에는 샘플 질의어를 입력받게 된다. 만약 사용자가 입력한 사항이 모두 맞으면 그림 17과 같은 결과를 출력하게 된다. 이

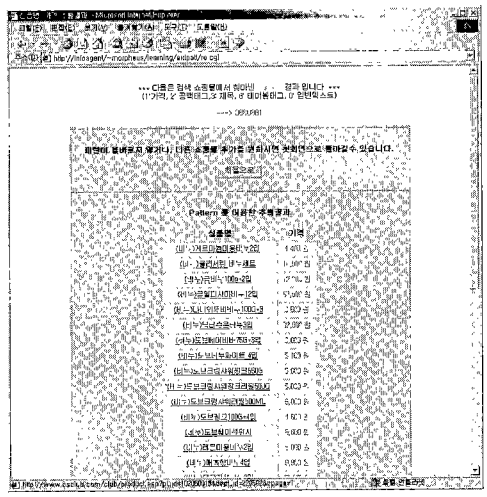

그림 18 웹 상점의 학습 완료

러한 과정을 통해서 특정 웹 상점에 대한 질의어 템플릿을 에이전트는 학습하게 되며 이렇게 학습된 특정 웹 상점의 질의어 템플릿은 공유되어 다른 사용자들도 사용할 수 있게 된다.

질의어 템플릿이 성공한 경우 그림 17의 상단부에 나오는 "OK"버튼을 통해서 실제 현재 웹 상점에 대한 학습이 이루어지게 된다. 학습 데이타는 그림 16에 나오는 "Query 결과"이다. 학습이 제대로 이루어지면 그림 18과 같은 PDU의 패턴과 추출 결과를 얻게 되며 이것은 그림 14와 같은 형태로 웹 상점 리스트에 추가된다.

그림 19는 다중의 웹 상점들로부터의 상품 검색 결과를 단일한 형태로 출력한 결과이다. 에이전트는 각 웹

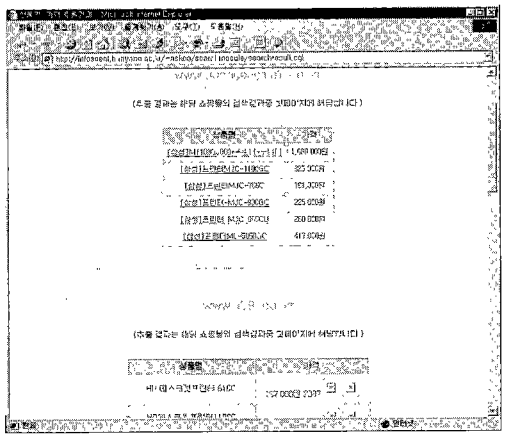

그림 19 모피우스의 비교 검색 결과

상점들의 검색 결과로부터 PDU를 추출하여 단일한 형태로 결과를 통합하게 된다. 에이전트는 통합된 결과를 HTML 파일로 변환하여 사용자에게 제공하여 사용자가 이를 토대로 상품에 대한 비교 쇼핑을 할 수 있게 한다.

4. 실험 결과

학습 알고리즘의 성능을 측정하기 위해 다양한 상품 검색 결과를 가지는 웹 상점을 대상으로 실험하였다. 학습 대상 웹 상점의 개수는 총 62개이며 검색 결과가 테이블인 웹 상점이 28곳이고 검색 결과가 리스트인 웹 상점은 34곳이다.

표 2는 일부 웹 상점에 대한 분석 결과이다. 우리의 실험에서 정확도는 약 93%이다. 실험 결과 알 수 있었던 사실은 검색 결과가 테이블인 경우 즉, 검색 결과가 구조화되어 있는 경우 100%의 정확도를 가진다. 하지만 검색 결과가 리스트인 경우, 즉 준 구조화된 검색 결과나 비 구조화된 검색 결과에서는 정확도가 떨어졌다. 정확도가 떨어진다는 의미는 검색 결과로부터 PDU를 생성하지 못하는 경우도 있지만 PDU가 아닌 정보를 함께 추출하는 경우까지 포함한다. 다음과 같은 검색 결과에서는 학습이 정확하게 이루어지지 않았다.

- PDU에 포함되어 있는 상품 속성에 대한 인식문제: 특히 가격 정보를 판별하는 regular definition사이에 HTML의 태그가 존재하여 판별이 정확하게 이루어지지 않은 경우에 발생한다. 이 경우에는 잘못된 PDU를 학습하는 경우가 발생하였다.
- PDU의 길이가 짧은 경우: PDU의 길이가 너무 짧은 경우 잘못된 정보를 추출한다. 실험에 의하면 PDU의 길이가 적어도 두 줄 이상은 되어야 한다.
- 하나의 논리적 라인에 다중의 상품 속성이 존재하는 경우: 하나의 논리적 라인에 여러 상품 속성 정보가 존재하면 검색 결과를 통합하는 과정에서 속성 정보가 존재하는데도 추출하지 못하는 경우가 발생한다.

개발된 시스템은 계산시간을 줄이기 위해 한 논리적 라인에 대해 여러 상품 속성을 인식하지 않는다. 현재 한 라인의 다중 상품 속성의 분석에 대한 연구가 진행중이다.

5. 관련연구

준 구조화된 문서에서 원하는 정보를 추출하기 위한 연구들이 진행되고 있다. TSIMMIS[3,6,8]와 ARANEUS[4]는 정보 추출을 위해 연구되고 있는 시스템들[5,7]로 사람이 어떤 정보를 어떻게 추출해야 하는지를 기술해 주어야 한다. 처음으로 비교 쇼핑에 대해

표 2 62개 웹 상점 중 일부에 대한 실험 결과 분석

Store URL	Test query	학습된 PDU 패턴	총 PDU 수	추출을 위해 사용된 PDU의 발생 횟수	학습 결과
www.more.com	gift	388088088088121	7	7	OK
www.jewelryweb.com	ring	3202222888088188880882l	18	18	OK
www.softwarebuyLine.com	school	320808020808088081	40	29	OK
www.1cache.com	video	32088021	10	10	OK
www.etronixs.com	video	32088021	10	10	OK
www.bookbay.com	java	3202021	17	17	OK
intertain.com	java	321	133	133	OK
www.more.com	gift	388088088088121	7	7	OK
www.amazon.com	java	32088881	50	26	OK
⋮	⋮	⋮	⋮	⋮	⋮
학습 성공 웹 상점 : 58					
www.dsports.com	ball	320881	9	9	Fail
www.drugstore.com	cream	3880881	14	10	Fail
⋮	⋮	⋮	⋮	⋮	⋮
학습 실패 웹 상점 : 4					

ARANEUS[4]는 정보 추출을 위해 연구되고 있는 시스템들[5,7]로 사람이 어떤 정보를 어떻게 추출해야 하는지를 기술해 주어야 한다. 처음으로 비교 쇼핑에 대해 연구하고 프로토타입 시스템으로 개발된 BargainFinder [15] 또한 사람이 직접 정보 추출 규칙을 작성해 주어야 한다. 현재 비교적 많은 사람들에게 알려져 있는 MySimon[16], PriceWatch[17], BottomDollar[18], 코메로[19]와 같은 시스템 또한 사람이 직접 정보 추출 규칙을 작성해야 한다. 이렇게 정보 추출 규칙을 수동으로 작성해야 하는 시스템들은 공통적으로 새로운 웹 상점에 대한 확장성의 결여 및 기존 웹 상점의 검색 결과의 변화에 대해 지능적으로 대처하지 못하는 문제점을 가지고 있다.

이러한 문제점을 해결하기 위해 준 구조화된 문서에서 자동으로 정보 추출 규칙을 생성하는 시스템[9,10,11,12,13,14]에 대한 연구가 제안되었다. 이러한 시스템에서는 정보 추출 규칙을 귀납적 학습을 통해서 생성한다. 이러한 방법은 문서의 구조에서 나타나는 규칙성을 바탕으로 한다.

이러한 측면에서 모피우스는 ShopBot과 가장 유사한 연구이다. 두 시스템 모두 온라인 웹 상점에 대한 wrapper를 귀납적 학습 방법을 사용하여 생성한다. ShopBot은 HTML 파일에서 불필요한 헤더와 테일 정보를 제거한 후, 논리적인 라인을 생성해낸다. 이 논리적인 라인은 또 다시 보다 추상화된 형태로 변환된다. 보다 추상화된 논리적 라인에 대해서 설명(description)을 붙인다. 논리적 라인의 설명은 빈도 카운터를 통해서 이루어진다. ShopBot이 새로운 웹 상점을 학습하기 위해서 HTML 파일에 대한 많은 변화과정을 거쳐야 하는 반면에 모피우스는 매우 간단한 변환을 통해 보다 빠르고 안정된 학습을 할 수 있다. 모피우스 또한 HTML 파일을 논리적 라인으로 바꾼다. 논리적 라인으로 변환할 때 각 논리적 라인의 의미를 파악하고 이에 알맞은 카테고리를 부여한다. HTML은 이제 카테고리의 연속으로 볼 수 있다. 연속된 카테고리에서 빈번하게 발생하는 패턴을 추출하여 이를 해당 웹 상점에서 필요한 상품 정보의 추출을 위한 추출 규칙으로 사용한다. ShopBot은 하나의 PDU가 하나의 논리적 라인에 존재함을 가정한다. 이러한 강한 바이어스는 실제 시스템의 성능을 저하시킬 수 있다. 그러나 모피우스는 이러한 바이어스를 사용하지 않으며 상품 정보가 하나 이상의 논리적 라인에 걸쳐 설명되어 있는 경우뿐만 아니라 패턴에 노이즈가 존재하는 경우에도 정확하게 상품 정보를 추출할 수 있다.

6. 결론 및 향후 연구

대부분의 비교 쇼핑 시스템들은 서비스 제공자가 제공하는 웹 상점에 대해서만 비교 쇼핑을 할 수 있었다. 이러한 시스템들에는 다음과 같은 문제점을 가지고 있다.

● 사용자 입장에서의 웹 상점에 대한 고려: 대부분의 사용자들은 자신이 즐겨 애용하는 웹 상점들이 정해져 있다. 이 의미는 자신이 애용하는 웹 상점에 대해서는

신뢰를 하고 그 상점으로부터 상품을 구매할 수 있음을 의미한다. 현재의 비교 쇼핑 시스템들은 웹 상점의 신뢰도를 고려하지 않고 있다. 사용자가 애용하는 웹 상점이 비교 리스트에 포함되어 있지 않은 경우에는 사용자에게 의미가 없다.

● 상품 검색 속도: 앞에서도 언급했듯이 사용자는 자신이 애용하는 웹 상점이 정해져 있다. 현재의 비교 쇼핑 시스템들은 자신들이 비교 할 수 있는 모든 웹 상점들을 다 비교하기 위해서 많은 시간을 소비한다.

모피우스는 이러한 문제점들을 해결하여 사용자가 원하는 웹 상점들에 대해 클라이언트에서 웹 상점을 학습하고 이를 토대로 비교 쇼핑을 수행하는 에이전트 시스템을 제안하였다. 모피우스는 다음과 같은 장점을 가지고 있다.

● 적은 온톨로지의 사용: 모피우스는 다른 시스템들과는 다르게 아주 적은 온톨로지만을 사용하자. 즉 hard-coded된 온톨로지만을 사용하며 온톨로지에 대한 유지가 필요없다.

● 빠른 비교 쇼핑: 사용자가 신뢰하는 웹 상점에 대해서만 비교하므로 다른 비교 쇼핑 시스템들보다 빠르게 결과를 출력해 낼 수 있다.

● PDU에 의한 상품 정보 추출: 학습 알고리즘은 PDU에 노이즈가 존재하는 경우에 대해서도 능동적으로 대처할 수 있다. 이는 기존에 추출하지 못하는 상품 정보에 대해서도 이를 인식하고 추출할 수 있게 한다.

모피우스가 모든 웹 상점에 대해서 정상적으로 학습을 수행하지는 못한다. 실험결과에서도 알 수 있듯이 학습 성공률이 95%를 넘지 못한다. 이를 해결하기 위해서는 모피우스에 보다 정교한 문자열 기능이 있어야 하며 논리적 라인을 분석할 수 있는 온톨로지가 필요하다.

참 고 문 헌

[1] Kushmerick, N., Weld, D., Doorenbos, R., *"Wrapper Induction for Information Extraction,"* *International Joint Conference on Artificial Intelligent,* pages 729-735, 1997.

[2] Doorenbos, R., Etzioni, O., Weld, D., *"A Scalable Comparison-Shopping Agent for the World Wide Web,"* *First Internation Conference on Autonomous Agents*, pages 39-48, 1997.

[3] Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., Vassalos, V., *"Template-based wrappers in the TSIMMIS system, "ACM SIGMOD International Conference on Management of Data,* pages 532-535, 1997.

[4] Atzeni, P., Mecca, G., Merialdo, P., *"Semi-structured and Structured Data in the Web: Going Back and Forth.," ACM SIGMOD Workshop on Management of Semi-structured Data,* pages 1-9, 1997.

[5] P. Atzeni, G. Mecca, and P. Merialdo., *"To weave the web,"* In *Proceedings of the PODS'97*, 1997.

[6] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom, *"The tsimis project: Integration of heterogeneous information sources,"* In *IPSJ Conference*, pages 7-18, January 1996.

[7] A. Gupta, V. Harinarayan, and A. Rajaraman, *"Virtual database technology," SIGMOD Record,* 26(4):57-61, December 1997.

[8] J. Hammer, H. Garcai-Molina, J Cho, R. Aranha, and A. Crespo, *"Extracting semistructured information from the web,"* In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[9] B. Adelberg, *"Nodose - a tool for semi-automatically extracting structured and semistructured data frim text document,"* In *Proceedings of SIGMOD'98*, 1998.

[10] N. Ashish and C. Knoblock, *"Wrapper generation for semi-structured internet sources," SIGMOD Record,* 26(4):8-154, December 1997.

[11] S. Soderland, *"Learning to extract text-based information from the world wide web,"* In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining,* pages 251-254, Newport Beach, California, August 1997.

[12] S. Soderland, *"Learning Text Analysis Rules for Domain-specific Natural Language Processing,"* Ph.D. Thesis. Dept. of Computer Science Technical Report.

[13] S. Soderland, D. Aronow, D. Fisher, J. Aseltine, W. Lehnert, *"Machine Learning of Text Analysis Rules for Clinical Records,"* CIIR Technical Report.

[14] Riloff, E., *"Automatically Constructing a Dictionary for Information Extraction Tasks," Proceedings of the Eleventh National Conference on Artificial Intelligence,* 811-816, 1993.

[15] BargainFinder, http://bf.cstar.ac.com/bf

[16] MySimon, http://www.mysimon.com

[17] PriceWatch, http://www.pricewatch.com

[18] BottomDollar, http://www.bottomdollar.com

[19] 코메로, http://www.commero.com

양 재 영

1998년 한양대학교 전자계산학과 졸업(학사). 2000년 한양대학교 대학원 전자계산학과 졸업(석사). 2000년 ~ 현재 한양대학교 대학원 컴퓨터공학과 박사과정. 관심분야는 지능형 에이전트 시스템, 기계학습, 데이타마이닝, 정보검색, 정보추출

김 태 형

1986년 서울대학교 컴퓨터공학과 졸업(B.S) 1988년 서울대학교 컴퓨터공학과 졸업(M.S.). 1988년 ~ 1990년 현대전자산업(주) 응용S/W 개발부. 1996년 University of Maryland, Department of Computer Science (Ph.D.). 1996년 ~ 1999년 현대정보기술(주) 정보기술연구소 책임연구원. 1999년 ~ 2000년 한양대학교 컴퓨터공학과 전임강사.

최 중 민

1984년 서울대학교 컴퓨터공학과 졸업(학사). 1986년 서울대학교 대학원 컴퓨터공학과 졸업(석사). 1993년 State University of New York at Buffalo, Computer Science 졸업(박사). 1993년 ~ 1995년 한국전자통신연구원 인공지능연구실 선임연구원. 1995년 ~ 현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 지능형 에이전트 시스템, 인공지능, 정보검색, 데이타베이스, HCI