■ 2002년 정보과학 논문경진대회 수상작

유전체 서열의 정렬 기법을 이용한 소스 코드 표절 검사

(Applying Genomic Sequence Alignment Methodology for Source Codes Plagiarism Detection)

강 은 미 † 황 미 녕 †† 조 환 규 †††
(Eun-Mi Kang) (Mi-Nyeong Hwang) (Hwan-Gue Cho)

요 약 일반적인 컴퓨터 프로그램의 구성적, 구문적 특징은 소스 코드로부터 추출한 키워드들의 서열로 나타낼 수 있다. 따라서 추출한 키워드의 서열을 비교하면 두 프로그램의 유사성과 상이점에 대해서 잘 파악할 수 있다. 서열의 유사성을 측정하는 여러 가지 방법은 생물학적 유전자 서열을 다루는 생물정보학에서 활발한 연구가 이루어져왔다. 본 논문에서 우리는 두 프로그램간의 유사성을 측정하고 서열 정렬 방법을 이용하여 부분 표절 검출을 하는 새로운 방법을 제안한다. 제시한 방법의 성능을 평가하기 위해서, 2001년 자료구조 수업에 참석한 수강생들이 제출한 프로그램을 실험 데이타로 사용하여 표절을 검사하였다. 실험결과는 제안된 기법이 표절 검사에 있어 가장 널리 사용되는 지문법(fingerprint)보다 더 효과적임을 보여 주었다.

키워드 : 서열 정렬, 표절 검사

Abstract The syntactic and semantic characteristics of a computer program can be represented by the keywords sequence extracted from the source code. Therefore the similarity and the difference between two programs can be clearly figured out by comparing the keyword sequences obtained from the given programs. Various methods for measuring the similarity of two different sequences have been intensively studied already in bioinformatics on biological genetic sequence manipulation. In this paper, we propose a new method for measuring the similarity of two different programs and detecting the partial plagiarism by exploiting the sequence alignment techniques. In order to evaluate the performance of the proposed method, we experimented with the actual program codes submitted by 70 students attending a Data Structure course year 2001. The experimental results show that the proposed method is more effective and powerful than the fingerprint method which is the most commonly used for the plagiarism detection.

Key words : sequence alignment, plagiarism detection

1. 서 론

글을 쓸 때 원본 자료의 출처를 분명히 밝히지 않고 자기 것인 양 쓰는 것을 표절(plagiarism)이라고 한다. 책, 잡지, 인터넷 매체 등을 통해 얻은 자료를 이용할 때 참고 문헌을 달지 않고 임의로 도용함으로써 발생하는 문서(document)에 대한 표절 행위뿐만 아니라, 프로그램 소스 코드 역시 기존의 프로그램을 간단한 편집과정을 거쳐 제출하거나 이미 만들어진 소스 코드를 원저자의 참조 사항을 달지 않고 무단으로 재사용하는 것도 표절이라고 본다[1]. 문서나 프로그램 소스 코드의 표절은 인터넷 매체의 발달과 문서 편집기 사용의 확대로 인해 더욱더 만연해지고 있으나, 이를 탐지하거나 방지

· 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2002-042-D00433).
† 비 회 원 : 부산대학교 전자계산학과
emkang@pearl.cs.pusan.ac.kr
†† 비 회 원 : 한국과학기술정보연구원 연구원
mnhwang@pearl.cs.pusan.ac.kr
††† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
hgcho@pearl.cs.pusan.ac.kr
논문접수 : 2002년 6월 17일
심사완료 : 2003년 2월 18일

하기 위한 대책 마련은 미흡한 실정이다. 외국의 일부 대학에서는 학생들의 표절 행위를 검사하기 위한 시스템적인 접근이 이루어지고 있었지만[2], 국내에서의 표절 검사에 대한 연구는 초기 단계에 있다[3].

표절 여부를 검사하는 과정은 단순히 표절 행위를 금지하는 것만을 목적으로 하는 것이 아니라, 기존에 나와 있는 정보들의 유용성을 확인해주는 과정이기도 하다. 표절된 문서는 거의 유사한 내용을 담고 있기 때문에, 검색 사이트의 결과에서도 링크는 다르지만 동일한 내용을 가진 문서들이 걸러지지 않고 반복적으로 나타나게 된다. 검색 사이트에서 찾은 문서가 원본인지, 유사한 내용의 사본인지 확인하는 작업이 필요할 수도 있다[4]. 프로그램 교육 수업의 경우, 학생들이 제출하는 프로그램 과제의 표절 여부를 확인할 필요성이 있는데, 학생 수가 일정 수를 넘게 되면 인력으로는 검사하기가 어려우므로 자동으로 이를 탐지해주는 시스템이 필요하게 된다[3].

표절 행위의 대상이 되는 일반 문서나 프로그램 소스 코드의 공통점은 선형 구조(Linear structure)로 나타낼 수 있다는 것이다. 그림이나 표를 삭제한 문서나 불필요한 주석을 제거한 소스 코드는 생물정보학에서 연구되는 유전자 서열과 같이, 특정한 객체를 일렬로 늘여 놓을 수 있는 구조를 가진다. 문서의 구조는 단락의 순서를 일부 바꾸거나, 삽입, 삭제를 해도 전체적인 구성에 크게 영향을 미치지 않는 반면 프로그램 소스 코드의 경우 순서의 변경이나 일부 코드의 삭제는 전체의 논리적인 흐름을 바꾸어 놓거나, 실제로 정상적인 실행이 되지 않게 만들기도 한다. 프로그램 표절의 경우 표절을 하는 학생이 원본 소스 코드를 완벽하게 이해하지 못하기 때문에 간단한 편집 과정만을 거치게 되고, 이는 전체적인 구조의 흐름을 크게 바꾸어놓지 못하게 된다. 본 논문에서는 선형적인 구조를 가지는 프로그램 소스 코드의 구조적인 특성을 바탕으로 하여 표절 여부를 탐지하는 방법론에 대해 소개하고자 한다. 이 방법론은 프로그램의 논리적인 흐름을 나타내는 서열의 비교를 위해 생물정보학에서 사용하는 유전체 정렬 기법을 응용하였으며 대응하는 키워드의 일치, 유사성을 키워드 유사 행렬(keyword similarity matrix)을 통해 점층적으로 계산한다.

2. 표절과 검사 방법

표절 여부를 검사하는 방법에 있어 두 가지 다른 접근 방법이 제시되고 있다. 특징적인 단어에 초점을 두는 지문(fingerprint) 검사 방법과 전체 구성의 흐름에 초점을 두는 구조 기반 검사 기법이 있다. 두 가지 검사 방법을 설명하기에 앞서 일반 문서 표절 기법과 프로그램 소스 코드의 표절 기법에 대해 살펴보고, 이 방법들에 대응하는 표절 여부 검사 방법을 제시하도록 하겠다.

2.1 문서와 프로그램 소스 코드의 표절 수법

대체로 표절을 할 때에는 원본의 내용을 완벽히 이해하여 새로운 문서로 만들어내는 것이 아니라 짧은 시간 내에 일부분을 편집하여 사본을 만들어내게 된다. 그렇기 때문에 문서의 원본과 사본을 비교해 보면 다음과 같은 특징을 가진다.

1) 일부분의 단락이 삭제되거나 순서가 재배치되는 경우, 또는 원본에는 없는 단락이 삽입되어 있다.
2) 일부 문장을 편집하여 새로이 작성하였지만 주제어는 그대로 사용한다.
3) 문서의 구조나 단락의 구조가 동일하다.
4) 틀린 철자를 그대로 사용한다.
5) 문서에서 일부 단어를 의미가 동일한 단어로 교체하였다.

문장이나 문단의 주제와 내용은 동일한데 문장의 편집이나, 단어를 유사한 다른 단어로 교체하는 단순한 편집 과정을 거쳐 표절이 이루어진다. 또 두 종류 이상의 문서를 섞어 하나의 문서로 만드는 경우는 앞 뒤 단락의 흐름이 자연스럽지 못하다.

특히 전자 매체가 발달된 현재에는 사람이 직접 문서의 표절의 여부를 확인하는 것은 불가능한 일이다. 그렇기 때문에 컴퓨터를 이용해 문서의 특징을 파악하고 이를 토대로 표절 검사를 확인하는 작업이 필요하다. 문서는 선형 구조를 이루고 있기는 하나 문서의 크기에 따라 선형 구조의 길이가 가변적이기 때문에 구조적인 특징보다 통계적인 특징을 추출하여 이를 토대로 표절 검사한다. 즐겨 사용하는 단어나 문서 내에서 사용된 단어의 빈도 수, 단어, 문장, 문단의 평균길이 그리고 느낌표나 물음표등의 부호 사용 횟수 등을 비교하여 표절 유무를 검사하는데 이러한 방법을 지문법(fingerprint)라고 한다.

Parker와 Hamblen은 프로그램 소스 코드의 표절(plagiarism in software)을 "한 프로그램에 약간의 루틴의 변형을 가해서 만들어내는 프로그램"이라고 정의하였다[5]. 여기서 변형이라는 의미는 원본 소스 코드의 주석문을 바꾸거나 변수의 데이타 타입, 식별자(identifier)를 수정하는 것과 중복된 구문이나 불필요한 변수를 삽입하는 경우를 말한다. 또한 원본 소스 코드를 논리적인 흐름에 영향을 주지 않도록 단순하게 구조를 뒤섞는 일도 포함한다. 프로그램 소스 코드의 주요 표절 기법은 다음의 대략 9가지로 분류된다.

1) 원본 소스 그대로 제출한다.

2) 주석문을 삽입, 삭제, 편집한다.
3) 변수 이름이나 형식을 바꾼다.
4) 실제로 사용되지 않는 코드(garbage code)를 끼워 넣는다.
5) 정의된 함수의 위치를 바꾼다.
6) 부분적으로 소스 코드를 고쳐 제출한다.
7) 라이브러리 코드를 끼워 넣는다.
8) 다른 코드들을 조합하여 합친다.
9) 하나의 함수를 두 개 이상의 함수로 나누거나 둘 이상의 함수를 하나의 함수로 합쳐서 다시 작성한다.

위의 9가지 기법 중 앞의 5가지 방법은 간단한 표절 과정에서 사용되는 경우이고 뒤의 4가지 방법은 소스 코드에 대한 어느 정도의 이해를 전제로 하고 있는 경우로써 실제 이런 경우는 드물다고 볼 수 있다. 프로그램 언어에서 사용되는 문법이 이미 정의되어 있고 표절을 하는 학생은 한정된 시간 내에 프로그램의 전체적인 흐름을 이해하기 힘들기 때문에 문서 표절에 비해 편집 가능한 기법이 한정되어 있다고 볼 수 있다. 따라서 표절을 해도 원본 소스 코드를 전혀 다른 소스 코드로 작성해 내는 것이 아니라 일부분만을 편집하기 때문에 제어구조는 거의 동일한 결과물이 나오게 된다. 가장 단순한 소스 코드의 표절 확인은 스트링 비교 방법이 있다. 소스 코드에서 주석문을 제거하고 경계 기호(delimiter) 등을 제거한 후, 유닉스 명령어인 diff, grep를 사용하여 두 파일의 스트링을 비교하여 공통적으로 사용된 단어의 개수를 얻어내어 이를 토대로 유사성을 비교하는 방법이다. 초기에는 문서 표절 확인 방법으로 많이 사용되는 지문법을 사용하다가, 프로그램 구조의 특성을 이용해서 검사하는 방법론이 대두되었다.

2.2 여러 가지 기반 표절 검사 방법

문서에서 표절을 확인하는 간단한 방법으로 두 개의 문서에서 사용된 단어들의 유사성을 살피거나 사용된 단어의 빈도수 등을 비교하는 지문법을 들 수 있다. 문서의 표절 검사에서 통계학적인 방법을 사용하는 이유는 문서 길이에 영향을 받지 않고, 가장 쉽게 문서의 지문(fingerprint)을 얻어낼 수 있기 때문이다. 예를 들어, 하나의 문서에서 단락의 순서를 변경하더라도 사용된 단어의 개수, 주제어의 사용 횟수, 단어의 빈도수 등은 모두 동일하게 나타나게 된다. 이 방법을 프로그램 소스 코드의 표절 검사에도 적용을 하면, 사용된 토큰의 개수나 특정 키워드의 사용 횟수 등을 기준으로 하여 표절 유무를 평가하게 된다. 그러나 문서나 프로그램에서 일부분을 추가로 삽입한 경우에는 사용한 단어의 빈도수가 달라지기 때문에 표절 여부 파악이 힘들다.

표 1에서는 문서 표절 검사 시스템과 지문법을 사용하는 표절 검사 시스템을 소개하고 있다. 온라인에서 표절 검사를 해주는 Plagiarism.org[6], IntergirGuard[7]와 EVE2[8] 세 곳은 기본적으로 지문법을 사용하여 특정 그룹을 관리해주거나 특정 문서와 유사한 문서를 검색해 주는 유료 사이트들이다. 이 세 사이트는 정확한 검색 방법론을 공개하고 있지는 않지만 문서의 특징적인 부분을 추출하여 이를 토대로 표절 검사를 한다는 점은 동일하다. Plagiarism.org에서는 대용량의 데이타베이스에 저장되어 있는 기존의 문서들과 비교를 하거나, EVE2는 검색을 통해 결과로 나오는 유사한 문서들과 비교한다. EVE2는 특정 문서와 유사한 문서를 찾아

표 1 문서 표절 검사 시스템과 지문법을 사용하는 표절 검사 시스템

시스템	검사 대상	검사 방법	기타
Plagiarism.org	일반 문서	자세한 방법 공개치 않음	온라인 유료사이트 대용량의 데이타베이스 운영
IntergirGuard	일반 문서	자세한 방법 공개치 않음	온라인 유료 사이트
EVE2	일반 문서	자세한 방법 공개치 않음	온라인 유료 사이트 인터넷으로 유사한 문서 검색
CopyCatch	일반 문서	문서내의 공통 어휘 빈도수 검사	비온라인 시스템
WordCheck	일반 문서	문서내의 단어 사용 횟수 검사	비온라인 시스템
COPS	일반 문서	문장 일치 여부 검사	비온라인 시스템 문장을 데이타베이스에 저장
SCAM	일반 문서	단어 일치 여부 검사	비온라인 시스템 단어를 데이타베이스에 저장
교수 클럽	일반 문서	단어, 특수문자, 공식 등의 빈도수 검사	온라인 유료 사이트
SIM	프로그램 소스 코드	토큰들의 참조 횟수 비교	-
Siff	프로그램 소스 코드	50개의 대표 문자를 추출하여 비교	-

주는 검색 사이트이고, Plagiarism.org와 IntergirGuard는 학급 단위로 등록하여 과제의 표절 검사를 대행해 준다.

이외에도 오프라인 검사 시스템으로 문서내의 공통 어휘의 빈도수를 세어서 검사하는 Birmingham 대학의 CopyCatch[9], 문서에 나타나는 단어들의 사용 횟수를 체크하여 이를 비교하는 WordCheck Keyword Software[10]가 있다. Copy Protection System(COPS)은 전자 도서관에서 사용하는 복사물 탐지 시스템으로 Stanford Digital Library Project의 일환으로 제작되었다[11]. 여러 가지 형식의 문서들을 ASCII로 변환하고 문서를 문장으로 나눈 후 이들 문장을 그룹 지어서 데이타베이스에 저장을 한다. 비교하고자 하는 문서가 들어오면 이들 데이타베이스에 있는 문장과 중복되는 것이 있는지를 비교한다. COPS를 만든 Narayanan Shivakumar와 Hector Garcia-Molina가 COPS를 개선하여 SCAM(Stanford Copy Analysis Mechanism)을 개발했다. 차이점은 COPS는 문장을 해시 테이블에 저장하는 반면 SCAM은 문장 대신 문서에서 사용된 단어의 빈도수를 벡터로 나타내고 이들 벡터들 간의 dot-product를 통해 유사성을 검사한다. 국내에서는 과제 내용상에 나오는 단어, 공식, 특수문자 등의 출연 빈도를 추출하여 과제의 유사도를 분류하는 교수 클럽 사이트가 나와 있다[3].

문서와 마찬가지로 프로그램 소스 코드 역시 지문법 적용이 가능하다. Vrije 대학의 Dick Grune이 개발한 SIM(Software Similarity Tester)는 소스 코드에 있는 토큰들이 다른 곳에서 참조되는 횟수를 계산해서 히스토그램을 그리고 이 히스토그램을 비교해서 표절 여부를 확인한다[12]. 대용량 파일 시스템에서 유사한 파일을 찾아내는 것을 목적으로 만들어진 응용 프로그램인 Siff는 파일에서 50개의 문자를 지문으로 뽑아내어 이를 비교한다[13].

표 2에서는 구조 기반 표절 검사 방법을 사용하는 시스템을 소개하고 있다. 지문법을 이용하면 문서나 프로그램의 지문은 추출해낼 수 있지만 구조적인 분석은 어려워진다. 문서와는 달리 프로그램은 제어 흐름의 변경이 어려우며, 프로그래밍 언어의 문법이 정해져 있기 때문에 구조적인 특성이 잘 나타난다. 구조기반(Structure-based) 표절 검사 방법은 문서의 표절 검사보다 제어흐름을 가지고 있는 프로그램 소스 코드의 표절 검사에 많이 사용된다. CHECK 시스템은 다른 문서 표절 시스템과는 달리 문서의 구조를 먼저 분석하고 중요도가 높은 키워드를 추출하여 이를 지문 벡터(fingerprint vector)로 비교한다. 문서 내에 문서의 구조를 포함하는 LATEX 문서에서 구조적인 트리를 구성하고, 이 트리를 토대로 중요도가 높은 키워드 분포도를 알아내어 이를 비교한다[14]. 그러나 이 시스템은 LATEX 문서만 검사가능하고, 문서 내부에 문서의 구조를 포함하지 않는 일반적인 ASCII 문서들은 검사할 수 없다.

프로그램 코드에서 토큰을 추출하여 나열하면 선형 구조를 이루는 서열이 만들어지게 된다. 이 토큰 서열들에서 Longest Common Subsequence를 찾아서 그 길이를 유사성의 기준으로 삼은 것이 Plague이다[15]. 이 Plague에서 토큰을 추출하는 부분을 좀 더 보강한 시스템이 YAP(Yet Another Plague)이다. YAP은 호출되는 함수의 순서를 파악하여 토큰을 재배치하고 비슷한 기능을 하는 라이브러리 함수들은 동일하게 처리한다[16]. YAP3는 YAP시스템을 확장, 서브 스트링 매칭 방법 중 Karp-Rabin Greedy-String-Tiling 알고리즘을 사용하며 영어로 된 과제의 표절 검사 기능까지 갖추고 있다[17]. 이들 시스템과 유사하며, 온라인 검사가 가능하도록 지원해주는 시스템이 MOSS와 JPlag이다[18, 19]. MOSS는 다양한 프로그래밍 언어를 지원해주고, JPlag는 Java 언어로 작성된 프로그램 코드를 비교하여 시각적으로 표시해준다. 이 시스템들의 공통점은 프로그램 코드를 토큰으로 분리하여 선형 구조를 이루는 서열로 만들고 이들 서열들 간의 비교를 스트링 매

표 2 구조 기반 표절 검사 방법을 사용하는 시스템

시스템	검사 대상	검사 방법	기타
CHECK	LATEX 문서	문서의 특성을 트리로 구성	키워드 분포도(지문법)를 일부 적용
Plague	프로그램 코드	Longest Common Subsequence	
YAP	프로그램 코드	스트링 매칭 방법 적용	
YAP3	프로그램 코드	Karp-Rabin Greedy-String-Tiling	영어로 된 과제의 표절 검사
MOSS	프로그램 코드	스트링 매칭 방법 적용	온라인 시스템
JPlag	프로그램 코드	Karp-Rabin Greedy-String-Tiling	온라인 시스템

칭 기법으로 접근한다는 점이다. 또 다른 구조적인 접근 방법으로, KAIST 전산학과 이광근 교수님 연구실에서 제시한 방법이 있다. 여기에서는 프로그램을 파싱하여 키워드로 이루어진 트리를 만들고 트리의 Tree sub-isormorphism matching을 통하여 유사성을 비교하는 방법이다[20].

문서의 표절 검사를 위해 많이 사용되는 지문법은 문서의 길이에 영향을 받지 않는다는 장점을 가진다. 또한 문서에서 지문을 뽑아내기만 하면 이 지문들끼리 비교를 통해 표절 여부를 판단하기 때문에 시간 복잡도도 낮다. 그러나 이 지문법은 부분적인 표절 여부를 찾아내기가 어렵고, 구조적인 특성을 가지는 프로그램 코드의 검사에 적용하기에는 신뢰성이 부족하다. 지문법과 구조 기반 검사방법에 대한 비교는 표 3과 같다.

표 3 지문법과 구조 기반 검사 방법의 비교

설 명	지문법	구조 기반 검사 방법
문서에서 얻어내는 정보	유동적	고정적
문서의 길이	영향을 받지 않음	길이에 따라 얻어내는 정보가 비례함
부분적인 표절 탐지	어려움	쉬움

3. 생물체 유전자 정렬 기법과 응용

생물정보학에서 많은 연구가 되어 있는 분야중 하나가 서열들을 비교하여 유사한 영역을 찾아내는 서열 분석이다. 생물정보학에서 서열 분석 연구는 유전자 서열이나 단백질 서열들을 비교해서 유사한 영역을 찾아내는 것인데, 이것은 동일한 서열을 가지는 유전자는 같은 기능을 가진다는 가정 하에 동종이나 타종의 유전체 서열의 기능 분석에 사용된다. 예를 들어 어떤 유전자의 기능을 추측하기 위해 가장 흔히 사용하는 방법은 우리가 알아내고자 하는 유전정보를 가진 서열을 유전체 데이타베이스(Genome Database)에서 찾아서 일치하는 것이나 유사성이 높은 것을 찾아내게 된다. 두 서열에서 부분적으로 동일하게 또는 유사한 부분이 있는지를 판단하는 것은 유전자의 서열 분석과 기능 유추를 위해 매우 중요한 일이다. 그리고 여러 유사한 서열들의 진화적인 계통도나 종별 연구를 위해서는 다중 정렬(multi alignment) 방법까지 사용된다. 이 분야는 이미 생물정보학에서 많은 연구가 되어 있으며, 정렬 방법들은 유전자나 단백질 서열의 비교뿐 아니라 프로그램 소스 코드 같은 선형 구조를 이루는 모든 객체들의 유사성 비교에 효과적으로 적용할 수 있다.

유전체 서열의 정렬에 있어 정렬이란 서열에 공백을 넣어 길이가 같은 서열로 만드는 것을 말한다. 전체 서열을 비교하는 전체 정렬(global alignment)과 부분열(substring)의 일치에 초점을 두는 지역정렬(local alignment) 방법이 있고, 길이의 차이가 나는 서열의 비교에 많이 사용되는 semi-global 정렬 방법이 있다. 이들 정렬 방법은 동적 프로그래밍 방법을 적용하면 매우 효과적으로 계산할 수 있다.

전체 정렬방법은 서열 원소가 일치하는 경우의 점수와 불일치하는 경우, 공백을 넣어주는 경우 각각에 해당하는 점수를 주고, 가능한 여러 가지 정렬 중 점수 합계가 높은 서열 정렬 결과를 찾는다. 일치하는 경우, 불일치하는 경우, 공백을 주는 경우의 점수는 응용 방법에 따라 다를 수 있으며, 아래의 예에서는 서열 원소가 일치하는 경우에는 +1점, 불일치하면 -1점, 공백을 넣을 때는 -2라는 점수를 주었다. ACGT의 염기 서열로 이루어진 DNA 서열인 GACGGATTAG 와 GATCGGAATAG를 정렬하면 여러 경우의 결과를 얻을 수 있다.

```
GA - CGGATTAG
GATCGGAATAG
```

정렬 (1)

```
- GACGGATTAG
GATCGGAATAG
```

정렬 (2)

정렬 결과에서 정렬 (1)의 점수를 계산하면 6점이 나오고 정렬 (2)는 2점이 나와서 정렬 (1)의 결과가 더 좋은 정렬이라고 평가한다. 동적 프로그래밍을 적용하여 전체 정렬을 구할 때에는 서열의 비교 과정인 점수를 저장하는 행렬을 사용한다.

$$a[i,j] = \max \begin{cases} a[i,j]-1+gap \\ a[i-1,j-1]+p(i,j) \\ a[i-1,j]+gap \\ 0 \end{cases} \quad (1)$$

$p(i,j)$는 서열1의 i번째 원소와 j번째 원소가 일치 여부에 따른 점수이고, gap은 공백 삽입에 대한 감점 점수로서 이를 이용하여 행렬 a를 채운다. 이 행렬 a의 오른쪽 하단의 마지막 값이 전체 정렬의 최대 값이 된다.

전체 정렬이 서열에 적절한 공백을 넣어서 같은 길이의 서열로 만드는 것에 비해 지역정렬은 서열의 부분열이 가장 길게 일치하도록 비교하는데 초점을 둔다.

DNA 서열 AGGTATTGA와 CCTATGGC의 지역정렬의 결과는 다음과 같다.

AGGTATTGA
- CCTATGGC

정렬 (3)

전체 정렬과 마찬가지로 점수를 저장하는 행렬 a를 동적 프로그래밍 방법을 통해 구하지만 공백을 넣어 강제적으로 정렬시키지는 않는다. 행렬 a의 원소 중 가장 최대 값이 두 서열의 지역정렬의 결과가 된다.

$$a[a,j]=\max\begin{cases} a[i,j-1]+gap \\ a[i-1,j-1]+p(i,j) \\ a[i-1,j]+gap \\ 0 \end{cases} \quad (2)$$

전체 정렬 방법은 공백을 넣어 두 서열을 정렬시키기 때문에 비교하는 서열의 길이가 다르면 결과가 좋지 않게 나온다. CAGCACTTGGATTCTCGG와 CAGCGTGG를 전체 정렬시킨 결과는 다음과 같다.

CAGCACTTGGATTCTCGG
CAGC - - - - - G - T - - - - GG

정렬 (4)

CAGCA - CTTGGATTCTCGG
- - - CAGCGTGG - - - - - - - - -

정렬 (5)

정렬 결과 정렬 (4)는 전체 정렬의 결과로 지나친 공백의 삽입으로 인해 결과가 좋지 않게 나온다. 이처럼 서열의 길이가 상이한 경우는 짧은 서열의 앞 뒤 공백은 무시하는 것이 정렬 (5)와 같이 더 나은 결과를 가져다준다. 전체 정렬과 같이 일치는 되지만 서열의 앞, 뒤에 붙어있는 공백을 무시하면서 정렬시키는 방법이 semi-global 정렬이다.

3.1 선형 구조체(Linear Object) 유사도 계산을 위한 방법론

생물정보학에서 연구되는 여러 가지 정렬 방법은 프로그램 서열이나, 악보, 일반 문서 또는 시간 순서(time series)로 구성되는 객체를 선형화한 후 이를 정렬하는 데 적용할 수 있다. 프로그램은 각 프로그래밍 언어마다 다른 특징이 있지만, 공통적으로 체계화된 문법이 있고 사용할 수 있는 예약어는 정해져 있다. 프로그램 소스 코드에서 변형하기 어려운 예약어들을 뽑아내어 서열을 만들면 소스 코드의 구조와 특성을 간단하게 추출할 수 있게 된다. 이 서열을 유전자 서열 또는 아미노산 서열에 대응시키면, 기존의 생물정보학에서 사용하는 여러 가지 정렬 시스템[21]을 사용하여 두 개의 서열을 비교할 수 있다. 소스 코드를 아미노산 서열로 대응시키는 과정이 표 4에 나타나 있다. 소스 코드의 토큰 중에서 미리 정의해놓은 키워드들와 블록의 시작과 끝을 의미하는 중괄호('{', '}') 등을 뽑아내고 이를 단백질을 구성하는 20개의 아미노산에 대응시킨다. 그러면 키워드 서열은 아미노산으로 구성되는 서열이 되어 이미 개발되어 있는 아미노산 서열 정렬 시스템을 사용해서 유사성 검사를 할 수 있다. 아미노산의 개수가 20개이기 때문에 , 아미노산 서열 정렬 시스템을 사용하기 위해서는 소스 코드에서 뽑아낼 수 있는 예약어의 종류가 20개로 제한된다. 하지만 이 과정을 통하여, 생물정보학에서 개발되어 있는 정렬 시스템을 이용하여 소스 코드에서 추출하여 아미노산에 대응시킨 서열들 또한 정렬 가능하며, 그 결과도 맞게 나온다는 것을 확인할 수 있다.

프로그램 소스 코드에서 추출된 키워드를 대응되는 아미노산 서열로 변환한 데이타를 ClustalW라는 다중 정렬 시스템에 입력 파일로 넣는다(그림 1). 이 ClustalW는 European Bioinformatics Institute에서 만든 유전자의 염기 서열, 단백질의 아미노산 서열을 다중 정렬시켜 주는 시스템이다[22, 23]. 이 시스템에 변환된 아미노산 서열들을 입력으로 넣고, 유사성 키워드 행렬을 주면 그 비교결과가 출력 파일로 나오게 된다(그림 2). 이 결과는 수치적인 결과이며, 시각적으로 확인하기 위해서 계통도 분석 프로그램인 PhyloDraw[24]를 이용하여 정렬된 정도에 따라 트리 모양의 그림으로 나타낼 수 있다. PhyloDraw를 이용하여 나타낸 트리 모양의 결과는 그림 3에서와 같이 확인 할 수 있다. 트리에서

표 4 간단한 소스 코드에서 키워드를 추출하고 이를 아미노산 서열로 대응시키는 과정

소스 코드의 예	추출되는 키워드	대응되는 아미노산
void main( ) {	{	D
int i;	int	N
for(i = 0; i<100; i++){	for (= < ++ {	R D S M P D
.....		
if( ) x = y;	if =	Y S
else break;	else break	V
while(1) { }	while { }	K D W
y = 10;	=	S
}	}	W
}	}	W

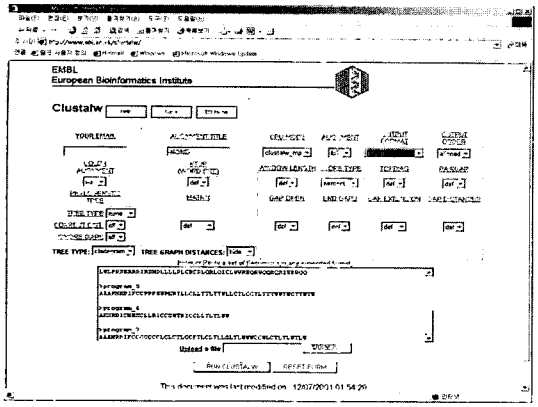

그림 1 프로그램 시퀀스를 아미노산으로 대응하여 ClustalW에 입력

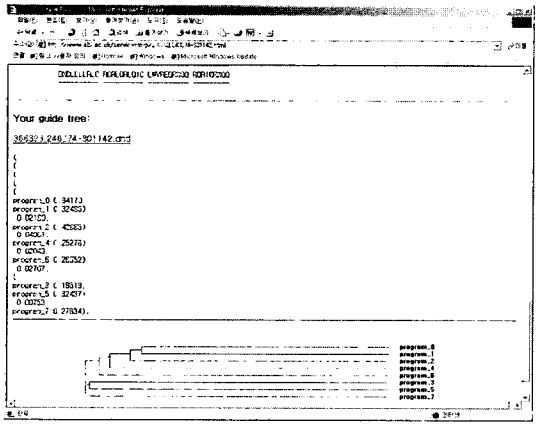

그림 2 ClustalW에서 나온 유사도 트리 결과

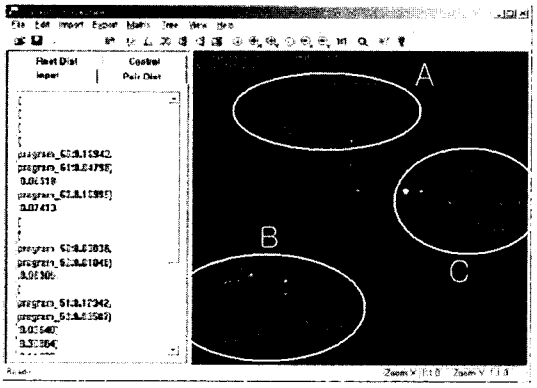

그림 3 ClustalW에서 얻은 유사도 트리를 PhyloDraw로 그린 결과 : A, B, C 개의 유사한 그룹으로 나누어진 것을 확인할 수 있다.

가까이 위치할수록 유사하게 정렬서열로서, 그림 3에서는 11개의 프로그램이 유사한 3개의 그룹 A, B, C로 나뉘는 것을 확인할 수 있다.

4. 함수 호출에 대한 선형화(Linear Ordering for Function Call)

표절 검사에 이용할 수 있는 방법들을 요약해보면, 지문법과 순차적인 키워드 시퀀스를 추출, 그리고 구조적인 면을 고려한 시퀀스 추출 기법을 들 수 있다. 지문법은 사용하기에 간단하고 쉬우며, 프로그램의 크기에 제한이 없다. 하지만, 지문법의 결과가 같다고 해서 표절이라고 단언하기에는 매우 부족하다. 예를 들어보면, 음악에서 '도'음 10개와 '레'음 10개로 이루어진 모든 곡이 같다는 결과가 나온다.

순차적인 키워드 시퀀스를 추출하여 표절검사를 하는 경우, 앞서 언급하였듯이 이미 생물학적 알고리즘 분야에서 개발되어있는 ClustalW[22] 및 PhyloDraw[24] 툴을 사용하여 표절을 검사할 수 있다. 표절 검사를 할 시퀀스를 추출하여 ClustalW에서 사용할 수 있도록 아미노산 서열로 대응시켜 주면된다. 대응되는 아미노산 서열끼리의 유사도는 ClustalW에서 확인할 수 있다. ClustalW에서 나온 결과를 PhyloDraw를 이용해서 그려보면 그 유사도 관계가 더욱 명확하게 나타난다. 하지만 이 경우에 순차적으로 프로그램 시퀀스를 뽑아내기 때문에 표절시에 불필요한 코드를 넣거나, 실제 수행에 사용되지 않는 함수를 추가하는 경우 추출되는 시퀀스가 달라진다. 또 프로그램의 특징을 나타내는 키워드 개수에 비해 아미노산의 개수가 20개로 한정되어 있기 때문에, 프로그램에서 추출한 키워드 시퀀스 중에서 어떤 것은 아미노산 시퀀스로 대응되지 못한다. 참고로 자바의 경우에 예약어의 개수만 40여 개다.

프로그램 소스로부터 특징적인 키워드들로 이루어진 키워드 시퀀스인 codeDNA를 추출하는 방법으로 기존의 프로그램 표절 검사 방법에서는 프로그램 소스 코드를 순차적으로 읽어서 소스 코드 키워드 시퀀스를 추출해 내어 비교하는 방법을 사용하였다. 이 방법을 사용하면 프로그램을 앞에서 뒤로 읽는 순서대로 키워드 시퀀스를 추출하기 때문에, 실제 프로그램 수행 순서를 고려한 구조적인 면을 반영하기 어렵다. 또 실제 수행에서 여러 번 호출되는 함수에 속하는 키워드들이나, 한번만 호출되는 함수에 속하는 키워드들이 키워드 시퀀스에서 나타나는 횟수가 같으며, 실제 수행에서는 호출되지 않는 함수에 속하는 키워드들도 키워드 시퀀스에 포함된

표 5 순차적으로 추출한 프로그램 키워드 시퀀스

소스 코드의 예	순차적으로 추출되는 프로그램 시퀀스
void main( ) {	**main_Start**
int i;	int
for(i = 0; i<100; i++) {	for, =, <, ++, **For_Start**
func_call();	
if( ) x = y;	if, =
else break;	else, break
while(1) { }	while **While_Start While_End**
y = 10;	=
}	**For_End**
}	**main_End**
void func_call() {	**Func_Start**
float c;	float
...	...
return;	return
}	**Func_End**

다. 표 6에서는 간단한 소스 프로그램과 이를 순차적으로 읽어나가면서 추출한 키워드 시퀀스를 나타내고 있다. 순차적인 방법으로 키워드 시퀀스를 추출하는 경우에, 함수 내부에서의 구조적인 특징이라고 할 수 있는 for, while 등의 제어구조는 키워드 시퀀스에 나타나지만, 프로그램이 수행 시에 불려지는 구조, 즉 함수 호출 구조를 파악하기는 힘들다. 순차적인 방법으로 키워드 시퀀스를 추출한다고 가정하고, 표 5의 프로그램 소스 코드에서 main 함수 내에서 func_call()이 있는 경우와 없는 경우를 생각해 보자. func_call()이 있는 경우와 없는 경우 실제로 동작하는 프로그램 수행 과정은 다르지만, 최종 키워드 시퀀스는 차이가 없음을 알 수 있다. 따라서 실제 수행 순서에 따른 함수 호출 구조를 고려한 키워드 시퀀스 추출이 필요하다. 구조적인 면을 고려한 키워드 시퀀스 추출 기법은 함수 호출 순서대로 프로그램의 키워드 시퀀스를 추출하는 것이다. 이는 프로그램의 수행 순서대로 키워드 시퀀스를 추출하기 때문에, 수행과정 중에 호출되는 함수들에 속하는 키워드들이 시퀀스에 포함하게 된다. 실제 수행에서 한번도 호출되지 않는 함수에 속하는 키워드들은 키워드 시퀀스에 포함되지 않는다. 따라서 함수 호출 순서대로 선형화 하여 시퀀스를 추출하면 프로그램의 특징을 가장 잘 나타내는 키워드 시퀀스가 추출된다. 프로그램의 키워드 시퀀스를 추출해낼 때 고려해야 할 또 다른 사항은, 여러 가지 프로그램 예약어들과 연산자들 중에서 어떤 것들을 프로그램 키워드 시퀀스에 포함시키는 것이 효과적인지를 알아내는 것이다. 되도록 프로그램의 특성을 잘 반영할 수 있는 키워드와 연산자를 포함시켜야 하며, 그 적정 수준을 찾아내는 것도 중요하다.

함수를 쪼개거나 합하는 경우에 함수 호출순서에 변화가 생기는데 , 유전체 서열 정렬방법 중에서 지역정렬 방법을 사용함으로써 보완된다. 지역정렬 방법을 사용하면 시퀀스의 부분열 일치를 이용해서, 전체 표절 및 부분 표절을 파악할 수 있다. 대부분의 경우에, 같은 기능을 수행하는 함수를 쪼개거나 합하더라도, 함수 내부의 코드는 크게 변하지 않는다. 따라서 구조를 제외한 나머지 키워드 시퀀스는 유사하게 추출되므로 지역정렬에서 일치되는 결과로 나타나게 된다. 이 세 가지 방법을 간단히 요약한 것이 표 6이다. 또 지역정렬 시에 정렬되는 키워드에 따라 일치할 때 주어지는 점수와 일치하지 않을 때 주어지는 점수를 다르게 적용하는 키워드 유사성 행렬을 사용한다. 유전체 서열에서 각 키워드들인 염기 A, G, C, T는 각각 같은 중요도를 갖는다고 할 수 있다. 반면, 프로그램 시퀀스의 키워드들은 그 성질에 따라서, 프로그램에 미치는 영향이 다르다. 예를 들어서 int 와 for의 두 키워드 중, 프로그램의 수행에는 for가 훨씬 더 많은 영향을 준다는 것을 알 수 있다. 따라서 이러한 프로그램의 키워드들의 성질을 고려하여 다른

표 6 기존의 표절 검사 방법과 함수 호출 선형화 방법 비교

	표절 검사 방법	특 징	단 점
교수 클럽[22]	지문법	방법이 간단하며 프로그램의 크기에 영향받지 않는다.	소스코드에 불필요한 코드를 넣으면 지문법 결과가 달라진다.
ClustalW를 이용[16]	생물학적 정렬	키워드들을 20개의 아미노산으로 대응하여 시퀀스를 만든다. 정렬 알고리즘을 개발하지 않고 ClustalW를 사용하면 된다.	프로그램 소스코드 내에서 함수의 위치를 바꾸면 시퀀스 순서가 달라진다. 프로그램의 특징을 나타내는 키워드들의 개수보다 아미노산의 개수가 훨씬 작다.
함수 호출 선형화	수행 순서를 고려한 구조적인 키워드 시퀀스 추출방법	함수 호출 순서를 고려하여 키워드 시퀀스를 추출하므로 실제 수행되는 흐름을 이용하여 표절 검사를 하게 된다. 프로그램 수행 시에 사용되지 않는 함수는 시퀀스에 포함되지 않는다.	프로그램에서 함수를 쪼개거나 합하는 경우, 함수 호출 순서가 달라진다. 호출구조는 달라지지만 수행 코드들은 유사하게 남아있으므로 지역정렬 기법으로 파악 가능하다.

정렬 점수를 적용하였고, 이 점수 집합을 키워드 유사성 행렬이라고 정의하였다.

본 논문에서는 추출할 키워드와 연산자를 따로 파일로 저장하여 파라미터로 사용하였으며, 이 파일들을 편집함으로써 추출할 키워드와 연산자를 제어 가능하다. 간단한 파일 편집과 다양한 실험을 통해서 추출할 키워드와 연산자의 적정 수준을 찾아낼 수 있다.

4.1 함수 호출 선형화(Linearization for Function Call)

기존의 방식은 프로그램의 함수 경계를 고려하지 않고, 소스 코드에 있는 순서대로 키워드 추출하였다. 이러한 경우, 소스 코드에 있는 함수들의 순서를 재배치하게 되면, 프로그램이 수행되는 순서는 변함이 없지만 소스 코드에 있는 순서대로 키워드를 추출하기 때문에 키워드 순서가 상당히 달라진다. 또한 실제로 프로그램 수행에서 실행되지 않는 코드가 추가되었을 때, 키워드 시퀀스에 포함될 수 있다. 요컨대, 소스 코드에 있는 순서대로 키워드 추출할 경우에는, 표절한 프로그램이라 하더라도 함수 순서의 편집이나 불필요한 코드를 프로그램 사이사이에 끼워 넣음으로서, 결과적으로 다른 키워드 시퀀스를 구성하게 된다. 따라서 이 서열들을 정렬하게 되면 낮은 유사성 값을 가진다. 이에 대한 방안으로 본 논문에서는 함수 호출 선형화 방법을 제시한다. 이는 프로그램 수행과정을 고려한 구조적인 키워드 추출방법이다. 실제로 프로그램의 수행과정 중, 함수의 호출 순서는 컴파일과 실행 시에 결정되어진다. 그러나 프로그램의 목적과 경우에 따라서는 수행 과정 중에 동적으로 호출순서가 결정되는 경우도 있기 때문에 이를 정확히 추적하는 것은 어렵다. 본 논문에는, 소스 프로그램에 나타난 호출 순서대로 프로그램을 재구성하여 CodeDNA를 구성하는 함수 호출 선형화 방법을 사용하였다. 즉, 프로그램 소스 코드에 나타난 함수 호출 순서에 따라, 전체 키워드가 선형으로 재구성되어 CodeDNA를 구성하게 된다. 소스 코드에 나타나는 함수 순서를 재배치하여도, 프로그램에서 함수 호출순서가 바뀌지 않으므로, 최종 추출되는 CodeDNA에는 영향을 미치지 않는다. 또 프로그램 소스 코드에는 있는 함수이지만, 실제로 수행 중에 한번도 호출되지 않는 함수일 경우에는 함수 호출 순서에서 제외되므로, CodeDNA에는 포함되지 않는다. 따라서 프로그램 수행 중에 호출되지 않는 불필요한 함수의 시퀀스가 제거된다. 함수 호출 선형화 방법을 사용함으로써 의미 있는 코드들이 의미 있는 순서대로 배치되어서 최종 CodeDNA가 구성된다.

함수 호출 선형화 방법에서 함수 호출 순서를 따라가다 보면 주의해야 할 한 가지 사항은 되불림(recursion)다. 되불림을 고려하지 않고 함수 호출 선형화를 진행하게 되면, 무한 호출에 빠지게 된다. 되불림은 자기 자신을 직접 호출하는 경우인 직접 되불림(direct recursion)과, 함수 A가 함수 B를 호출, 함수 B가 함수 A를 호출하는 간접 되불림(indirect recursion)으로 나누어 볼 수 있다. 함수 호출 순서를 결정할 때에는 모두 스택을 사용하여서, 직접 되불림 또는 간접 되불림 함수 호출 순서를 결정한다. 함수가 호출되는 순서를 탐색할 때 함수가 호출되는 순서만을 저장하는 스택을 사용하여 함수가 호출되는 순서대로 스택에 삽입하고, 그 호출된 함수가 종료되면, 스택에서 삭제한다. 만약 새로 불려지는 함수가 이미 스택에 들어있다면 되불림인 것이다. 되불림인 경우에는 되불림 부분이 종료되었다고 보고, 다음 단계로 함수 호출 순서를 진행한다. 함수 호출 순서를 저장하는 스택의 변화 과정이 바로 함수 호출 순서이다.

4.2 키워드 유사성 행렬

유사성을 검사할 2개의 프로그램으로부터 함수 호출 선형화 방법을 이용해서 CodeDNA를 생성한 후에는 이 두 CodeDNA의 유사성을 측정해 내야한다. 이 유사도 측정의 단계에서 유사성을 결정하는 것이 키워드 유사성 행렬(Keyword Similarity Matrix)이다. 두 CodeDNA 중에서 일치하거나 유사한 요소가 있을 때, 그 유사도 점수를 몇 점으로 줄 것인가를 결정하는 행렬이다.

CodeDNA를 구성하는 키워드는 그 성질에 따라, 프로그램의 구조적인 성질을 나타내는 구조적인 키워드와 그렇지 않은 비구조적인 키워드로 나누어 볼 수 있다. 구조적인 키워드에 해당하는 것은 클래스의 시작이나 끝, 함수의 시작이나 끝, 또는 제어와 관련한 for, while 등의 시작이나 끝을 나타내는 '{', '}' 가 있으며, 그 외 다른 키워드들을 비구조적 키워드들로 분류된다. 이 분류를 이용해서 구조적인 면의 유사성에 높은 가중치를 주거나, 비구조적 면의 유사성에 높은 가중치를 두는 것을 수학적 파라미터를 이용해서 적용할 수 있다. 구조적인 키워드와 비구조적인 키워드를 구별하여서 가중치를 두면, 경우에 따라서 구조적이거나 비구조적인 유사성을 중점적으로 측정할 수 있을 뿐 아니라, 적당한 가중치의 값을 적용하면 구조적인 유사성과 비구조적인 유사성이 적절히 반영되어 두 프로그램의 유사성을 가장 잘 나타내게 된다. 프로그램 유사성을 측정할 때, 구조적인 키워드의 일치와 비구조적인 키워드의 일치가 전체 프로그램의 유사성에 미치는 영향은 다르다. 프로그램 표절의 경우에 앞서 언급하였듯이 구조적인 흐름을 바꾸기

는 어렵기 때문에 구조적으로 두 프로그램의 유사성이 높은 경우 표절을 의심해 볼 수 있다. 하지만 구조적인 면이 완전히 일치할 때 반드시 표절이라고 단언할 수는 없다. 구조적인 면, 즉 함수의 호출 순서나 제어구조가 같더라도, 수행하는 작업의 내용이 다를 수 있기 때문에, 비구조적인 키워드들의 시퀀스의 유사성도 어느 정도 고려해야 한다.

키워드 테이블 T_{key} = { if, else, for, while, public, ...} ∪ { '{', '}', ... } ∪ { gap } 로 정의 할 수 있으며, 키워드 유사성 행렬인 score matrix $key\,S_{key}(p, q)$는 그림 4와 같이 나타낼 수 있다.

$key\,S_{key}(p,q)$	구조적 키워드			비구조적 키워드			
	{	}	...	if	else		...
{							
}							
...							
if	$q()$	$q()$	...	$(1-p)\cdot S_{ij}$	$(1-p)\cdot S_{ij}$		...
else	$q()$	...	...	$(1-p)\cdot S_{ij}$	...		...
...	...	...	...	...	...		...

그림 4 키워드 유사성 행렬

p : 구조적인 키워드와 비구조적인 키워드의 가중치를 조절하는 변수이다. 구조적인 키워드 쌍의 일치될 때는 가중치 점수 p 를 주고 비구조적인 키워드 쌍이 일치되었을 때에는 (1- p)의 가중치를 준다.

q : 일치되지 않았을 때의 점수이다.

$S_{i,j}$: 비구조적 키워드 쌍의 일치 점수이다.

gap : 공백과 정렬되었을 점수이다.

실제 프로그램에서는 구조적인 키워드인 '{', '}'를 그 위치에 따라 class start, class end, function start, function end, for start, for end, while start, while end로 세분화하여 사용하였다.

비구조적인 키워드의 일치 점수를 나타내는 $S_{i,j}$ 에, 성질이 비슷한 그룹별로 일정점수를 적용하였다. 예를 들면 for-while이 일치되는 경우에, for-for/ while-while의 일치점수보다는 낮지만, 일치되지 않는 경우나 공백과 일치되는 경우보다는 점수를 높게 주었다. 또한 $key\,S_{key}(p, q)$ 에는 나타나있지 않지만, 공백과 일치되는 경우에는 따로 공백 점수를 주었다.

4.3 프로그램 키워드 정렬

키워드 정렬의 방법으로는 앞서 소개한 전체 정렬, 지역정렬, semi-global 정렬의 세 방법이 있다. 학생들이 프로그램을 표절할 때 소스 코드 전체를 그대로 카피하는 경우보다는, 일부분을 카피하거나 소스 코드의 순서를 편집한다. 그렇기 때문에, 표절 프로그램과 원본 프로그램을 비교해보면 보면, 전체 시퀀스가 같게 나오기 보다는 시퀀스의 부분 부분이 일치함을 볼 수 있다. 따라서 세 가지 정렬 방법 중에서 시퀀스의 부분열 일치에 초점을 두는 지역정렬방법이 효과적이다. 지역 정렬 방법을 사용할 경우에 프로그램 코드 내에서 함수 위치를 바꾼 경우 파악도 쉽다.

가중치를 적용한 구조적 키워드 쌍의 일치 점수 p와 불일치 점수 q로 정의된 키워드 유사성 행렬 $key\,S_{i,j}(p, q)$을 이용하여, 두 프로그램 P_a , P_b 의 지역 정렬 유사성을 나타내는 Local Similarity LoSIM은 식 (3)과 같이 정의된다.

$$LoSIM(\ P_a\ ,\ P_b\ ) = Local\ alignment\ P_a\ and\ P_b\ with\ key\,S_{i,j}(p, q) \tag{3}$$

또한, 최적의 유사성 값을 나타내는 optimal SIM*은 식 (4)와 같이 정의할 수 있다.

$$SIM^*(\ P_a,\ P_b\ ) = \max_{0\le p,q\le 1}\{\ SIM_{p,q}(\ P_a\ ,\ P_b\ )\} \tag{4}$$

점수 p와 불일치 점수 q를 다양한 단계로 적용하였을 때, 가장 높은 유사성이 측정되는 것이 Optimal SIM^* 이다.

본 논문의 실험에서 사용한 방법은 LoSIM 과 Greedy String Tiling방법을 합한 것으로 일정한 LoSIM 이상의 길이를 갖는 모든 정렬에 대해서 재귀적으로 조사한다. 유사성을 측정할 두 프로그램 P_a , P_b 에서 함수 호출 선형화를 고려하여 CodeDNA를 추출하고, 두 CodeDNA의 길이 중 짧은 길이에 일정 비율 x를 적용한 길이 l 이상 지역 정렬되는 경우를 유사성 측정 점수에 포함시킨다.

$$l = min(\ P_a\ CodeDNA\ length,\ P_b\ CodeDNA\ length) \cdot x \tag{5}$$

즉 지역 정렬 값이 큰 정렬 부분을 찾아내어서 길이 l 이상을 만족하면 유사성 측정 점수에 포함시킨다. 그리고 시퀀스에서 정렬된 부분을 잘라낸 후, 나머지를 합한 시퀀스를 이용하여 다시 지역 정렬을 수행하고 지역 정렬 값을 조사하는 과정을 지역 정렬 일치 길이(local alignment match length)가 l 이상을 만족하지 않을 때까지 반복한다. 반복 과정에서 계산되는 일치 점수인 score를 Total Score에 계속 더한다. 여기서 길이 l 은 식 (5)처럼 구하며, 유사성 측정 점수 S 는 식 (6)과 같

이 백분율로 계산된다.

$$Total\ Score = \Sigma(\ if\ match\ length > l,\ score)$$

$$S=(Total\ Score/min(P_a\ CodeDNA\ length, P_b\ CodeDNA\ length\ ) \cdot match) \cdot 100 \quad (6)$$

5. 실험 및 결과

프로그램 표절을 검사하는 시스템을 JAVA로 구현하여, 실제 학생들이 제출한 프로그램을 데이타로 하여 표절 유무 측정을 실험하였다. 표절 검사 시스템은 크게 다음의 4단계로 나누어 볼 수 있다.

단계 1) 추출할 프로그램 키워드들을 설정한다.

추출할 프로그램의 키워드들을 프로그램 예약어, 연산자, 그리고 구조를 나타내는 기타 기호들로 구분하여서 설정한다. 추출할 프로그램 키워드들은 각각의 파일을 편집함으로써 제어할 수 있다.

단계 2) 키워드 유사성 행렬을 생성한다.

프로그램에서 나타날 수 있는 모든 키워드 일치의 경우에 해당하는 점수를 파일로부터 읽어서 행렬로 만든다. 일치 점수와 불일치 점수을 이용해서 전체 행렬을 초기화한 후, 구조적인 키워드의 매치, 비구조적인 키워드 매치의 경우 가중치 점수를 적용하여 행렬의 해당 원소에 점수를 설정한다. 성질이 유사한 키워드 그룹별로 각각 추가의 일치 점수를 설정한다.

단계 3) 함수 호출 선형화 방법을 적용하여 프로그램 P_a, P_b로부터 $P_aCodeDNA$, $P_bCodeDNA$ 추출한다.

단계 4) 부분 정렬을 이용하여 $P_aCodeDNA$, P_b $CodeDNA$의 유사성 정도 측정

$CodeDNA$로 추출할 키워드를 정의한 파일과 키워드 유사성 행렬 값 파일을 기본 정보로 하여, 두 프로그램 P_a, P_b를 입력받아서, 두 프로그램의 유사성을 점수화하여 출력한다.

실험 데이타로 학부 자료구조 과목 강의 중에 학생들이 제출한 프로그램 과제를 수집하였다. 각 학생들은 한 학기동안 12개의 과제를 제출하였으며, 각 과제별로 제출자는 60명 내외이고, 프로그램 작성 언어는 JAVA이다. 각 과제 프로그램은 하나의 파일로 구현되어있으며, 데이타의 입출력은 항상 파일로 이루어진다. 실험 데이타 그룹은 12개의 과제 중 5개를 선정하여, 학생들이 제출한 프로그램을 골라서 이를 바탕으로 하여 여러 가지 표절 기법을 적용, 조작된 실험 데이타를 만들어 실험 데이타 그룹을 구성하였다. 각 실험 데이타 그룹은 10개의 실험 데이타를 가지며, 내부 표절 그룹 구성은 표 7과 같다. 또 각 그룹에서, 실험 데이타 쌍에 대한 표절 여부를 나타내는 테이블은 표 8과 표 9와 같다.

표절 검사 실험은 각 실험 데이타 그룹에 대하여, 각 프로그램에서 순차적으로 시퀀스를 추출한 다음 지역정렬방법으로 유사성을 측정하는 실험과, 본 논문에서 제시한 방법인 함수 호출 선형화를 이용한 구조적 시퀀스 추출과 키워드 유사성 행렬을 적용하여 유사성을 측정하는 실험으로 구성된다. 순차적으로 추출한 시퀀스에서

표 7 실험 데이타 그룹

	data group 1	data group 2	data group 3	data group 4	data group 5
내부 표절그룹	그룹 A : 3개 그룹 B : 3개	그룹 A : 2개	그룹 A: 4개 그룹 B: 3개	그룹 A: 4개	내부 표절 그룹 없음
총 실험 데이타 수	10개	10개	10개	10개	10개

표 8 실험 데이타 그룹 1의 실제 표절 유무를 나타내는 테이블

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
P_1	-	1	1	1*	0	0	0	0	0	0
P_2		-	1	1*	0	0	0	0	0	0
P_3			-	1*	0	0	0	0	0	0
P_4				-	0	0	0	0	0	0
P_5					-	1?	1+	1+	0	0
P_6						-	1?	1?	0	0
P_7							-	1	0	0
P_8								-	0	0
P_9									-	1?
P_{10}										-

의 지역정렬 방법과 본 논문에서 제시한 방법을 비교하여 봄으로써, 본 논문에서 제시한 방법이 표절검사에 있어서 어느 정도의 측정능력이 있는지 확실하게 알 수 있다.

본 논문에서 제시한 지역정렬 방법을 바탕으로 함수 호출 구조를 고려한 키워드 추출, 키워드 유사성 행렬을 적용하여 표 7에서 제시한 데이타 집합을 사용하여 표 10처럼 실험을 하였다. 순차적 키워드 추출 후 지역정렬을 하는 방법과 함수 호출순서대로 키워드 추출 후 키워드 유사성행렬을 적용하여 지역 정렬하는 두 가지 방법으로 실험한 결과는 표 11, 표 12, 표 13, 표 14와 같다.

표 11과 표 13은 함수 호출 선형화 구조를 이용하지 않고 순차적으로 키워드를 추출해서 지역 정렬한 결과이다. 이때 일치 점수는 1점, 불일치 점수는 -1점 그리고 공백 점수는 -2점을 주었다. 표 12와 표 14는 함수 호출 선형화 구조를 이용해서 CodeDNA를 추출하였으며 구조적 키워드에 대해서 가중치 점수를 20점으로 주었다. 그리고 키워드 유사성 행렬을 적용하여, 유사한 키워드가 서로 일치할 때 키워드 유사성 행렬을 적용하여 동일한 키워드로 일치될 때의 점수보다는 낮지만, 불일치보다는 높은 점수를 주었다. 이때 일치 점수는 10점, 불일치 점수는 -10점 그리고 공백 점수는 -20점을 주었다.

유사도 점수를 계산하는 방법은 키워드의 개수와 일치 점수를 고려하여 전체 키워드 시퀀스 길이*일치 점수 값으로 총 유사도 점수를 나누어주는 방법을 사용하였다. 즉, 전체 시퀀스 길이가 길고, 유사도 점수가 높은 경우나 전체 시퀀스 길이가 짧고 유사도 점수가 낮은 경우에 둘 다 시퀀스 길이를 고려하여 점수를 계산하였기 때문에, 같은 유사도 점수가 나올 수 있다(식 (6)).

단순한 기법의 표절인 경우에는 순차적 시퀀스 유사도가 높으며, 또한 함수 호출 선형화와 구조적 가중치를 준 결과 값도 모두 높게 나옴을 볼 수 있다. 특히 함수의 위치적인 순서를 조작하여 표절한 경우에 본 논문에서 제시한 함수 호출 선형화 구조를 고려한 방법을 사용하여 검사한 결과에서, 유사도 점수가 상당히 높게 나타남을 볼 수 있다. 또 실제로 사용하지 않는 함수를 추가한 표절의 경우도 발견되었는데, 이 역시 함수 호출선형화 구조를 고려한 방법을 사용한 경우 모두 찾아낼 수 있었다. 순차적 시퀀스의 점수만으로 표절인지 아닌지 가려내기 어려울 때, 함수 호출 선형화와 구조적 가중치 및 키워드 유사성 행렬을 적용한 유사도 점수는 비교적 표절 여부를 명확하게 나타내고 있다.

실험 결과 중에 프로그램 표절인 경우 중에서, 순차적

표 9 실험 데이타 그룹 3 의 실제 표절 유무를 나타내는 테이블

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
P_1	-	1#	1#	1#	1#	0	0	0	0	0
P_2		-	1	1	1?	0	0	0	0	0
P_3			-	1*	1?	0	0	0	0	0
P_4				-	1	0	0	0	0	0
P_5					-	0	0	0	0	0
P_6						-	1	1*	0	0
P_7							-	1*	0	0
P_8								-	0	0
P_9									-	0
P_{10}										-

1 : 완전 표절, 전체 코드가 같으며, 코드 순서 편집이 있는 경우 포함
1# : 부분 표절, 전체 프로그램 흐름은 같으며, 함수의 1~2개를 완전 표절
1* : 함수의 순서나 함수를 나누거나 합한 경우
1? : if, for 문을 여러 개 넣어서 제어구조를 조작한 경우

표 10 실험 방법

	지역정렬		함수 호출 선형화	키워드 유사성 행렬	입력 데이타 표절 유무	실험 결과
실험 1	일치 점수	= 1	×	×	표 8	표11 표13
	불일치 점수	= - 1				
	공백 점수	= - 2				
실험 2	일치 점수	= 10	○	구조적인 키워드에 20점 적용	표 9	표12 표14
	불일치 점수	= -10				
	공백 점수	= -20				

시퀀스 유사도보다 함수 호출 선형화 와 구조적 가중치 및 키워드 유사성 행렬을 적용한 결과 점수가 낮게 나오는 경우가 1번 그룹에서 5건이 발견되었다. 소스 코드를 직접 확인해 본 바, 프로그램을 표절할 때 코드의 제어 구조를 변형한 경우이다. 예를 들면 하나의 if을 여러 개의 if문으로 쪼개고, 또 if문을 추가하였으며, for loop을 도는 회수를 for문 선언에서 정의하지 않고, for 문 내부에 여러 개의 if문을 두어서 break로 제어하는 구조로 바꾼 경우 등이었다. 이는 원본과는 많이 다른 프로그램 제어 구조를 나타낸다.

표 11과 표 13의 실험 결과를 보면 프로그램에서 함수 호출 선형화 방법과 구조적 가중치 및 키워드 유사성 행렬을 적용해서 뽑아낸 CodeDNA가 순차적 시퀀스를 사용하여 검사하는 것보다 더 나은 방법임을 알 수 있다. 적은 양의 실험데이타로 실험하였기 때문에, 구조적 가중치의 적정수준을 실험을 통해서 찾아내기는 어려우나, 구조적 가중치 및 키워드 유사성 행렬이 유사성 측정에 충분히 의미가 있음은 확인할 수 있다. 앞으로 많은 실험데이타를 확보하여 다양한 가중치를 가지고 실험을 한다면 충분히 구조적 가중치 및 키워드 유사성 행렬 값의 적정수준을 찾아낼 수 있을 것이다. 위의 실험을 바탕으로, 적당한 가중치 점수를 적용하게 되었을 때 예상되는, 표절 여부를 판단할 수 있는 점수 기준은 다음 표 15와 같다. 표 15에서 "분명한 표절"의 경우에는 실제로 사람이 프로그램 소스 코드를 직접 확인해보지 않더라도 거의 표절임이 확실한 경우이고, "표절 가능성 높음"의 경우는 부분 표절이 의심되는 경우이고, "확인 필요"의 경우에는 표절 검사 결과만으로는 표절

표 11 실험 데이타 그룹 1의 지역정렬 유사도 검사 결과

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
P_1	-	85	89	41	11	6	11	11	7	6
P_2		-	76	41	11	6	11	11	7	9
P_3			-	44	11	6	11	11	11	6
P_4				-	5	5	5	5	5	6
P_5					-	31	56	56	7	13
P_6						-	27	27	11	18
P_7							-	100	7	16
P_8								-	7	16
P_9									-	25
P_{10}										-

표 12 실험 데이타 그룹 1의 함수 호출 선형과 구조를 고려하여 키워드 시퀀스 추출 후 구조적 가중치 및 키워드 유사성 행렬을 적용한 유사도 검사 결과

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
	28*	28	28	113	138	241	164	164	169	94
P_1	-	100	100	67	28	32	28	28	32	32
P_2		-	100	67	28	32	28	28	32	32
P_3			-	67	28	32	28	28	32	32
P_4				-	7	0	0	0	0	0
P_5					-	13	44	44	0	23
P_6						-	17	17	0	0
P_7							-	100	0	9
P_8								-	0	9
P_9									-	43
P_{10}										-

* : 키워드 토큰의 개수

을 알기 어려운 경우이다. "표절 가능성 높음"과 "확인 필요"의 경우에, 표절 검사 결과만으로 표절이라고 판정하기에는 무리가 있으며, 직접 사람이 프로그램 소스 코드를 확인해보아야 하는 경우이다.

프로그램 표절 검사를 기능을 이용하여, 프로그램 자동 채점 웹 서버 시스템 Evaluation System for Programming Assignment(그림 5)를 구현하였다. 이 시스템은 웹 기반 자동 프로그램 채점 기능을 가지며 여기에 프로그램 표절검사 기능을 추가한 것이다. 이 웹 서버 시스템은 실제 프로그램 강의 과목에 사용되었으며, 강의 기간 동안 프로그램 자동 채점 웹 서버 시스템을 통해서 학생들의 과제 프로그램을 채점, 표절 검사를 하였다. 웹 기반 자동 채점 프로그램인 Evaluation System for Programming Assignment는 웹을 통해서 접근할 수 있으며, 학생들이 각자의 계정을 가지고 로그인하고, 과제를 제출할 수 있다. 학생이 웹 서버로 과제 프로그램을 제출하게 되면, 웹 서버에서는 미리 정해진 데이타들을 가지고 프로그램을 수행하고, 그 결과가 맞는지 틀린지에 따라 과제 프로그램 점수를 채점한다. 프로그램 자동 채점 웹 서버를 통해서 학생들이 과제 프로그램을 제출할 때, 같은 과제에 대해서 먼저 제출한 다른 프로그램들과 표절검사 단계를 거치게 하였다. 이 표절 검사에서 다른 프로그램과 유사도가 높게 나오는 경우에는 표절로 의심되어 제출이 취소된다. 프로그램을 표절한 경우에는 과제 제출이 인정되지 않게 된다. 이러한 방법을 사용함으로써 학생들의 프로그램 과제 표절을 많이 방지할 수 있었다.

표 13 실험 데이타 그룹 4의 지역정렬 유사도 검사 결과

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
P_1	-	60	96	53	11	9	7	8	5	5
P_2		-	62	52	11	9	5	8	4	6
P_3			-	54	11	9	7	8	5	5
P_4				-	11	9	5	8	5	5
P_5					-	50	9	11	11	13
P_6						-	7	9	7	11
P_7							-	40	7	13
P_8								-	11	16
P_9									-	17
P_{10}										-

표 14 실험 데이타 그룹 4의 함수 호출 선형화 구조를 고려하여 키워드 시퀀스 추출 후 구조적 가중치 및 키워드 유사성 행렬을 적용한 유사도 검사 결과

	P_1 71*	P_2 71	P_3 62	P_4 71	P_5 104	P_6 125	P_7 14	P_8 200	P_9 316	P_{10} 212
P_1	-	100	70	73	33	39	92	23	36	21
P_2		-	70	73	33	39	92	23	36	21
P_3			-	69	38	45	85	27	41	24
P_4				-	23	39	100	23	26	21
P_5					-	70	85	18	21	21
P_6						-	42	0	7	10
P_7							-	100	100	85
P_8								-	4	5
P_9									-	12
P_{10}										-

* : 키워드 토큰의 개수

표 15 실험결과를 바탕으로 한 표절 기준 점수표

	순차적 시퀀스추출	함수 호출 선형화	함수 호출 선형화 + 키워드 유사성 행렬
분명한 표절	90점 이상	80점 이상	80점 이상
표절 가능성 높음	70점 이상	50점 이상	50점 이상
확인필요	40점 ~ 70점	40점 ~ 50점	40점 ~ 50 점

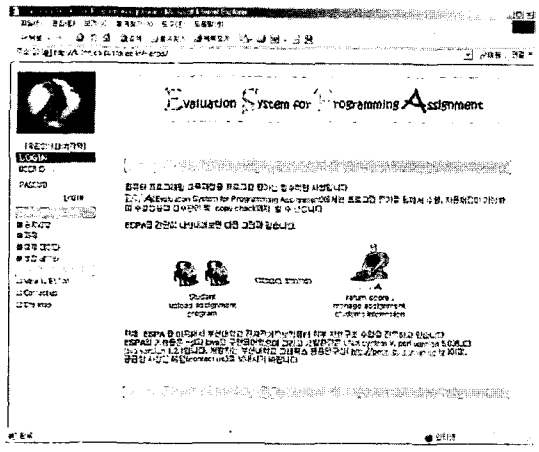

그림 5 표절 검사기능이 있는 프로그램 자동 채점 웹 서버

6. 결론 및 향후 연구 과제

본 논문에서는 선형 구조를 이루는 대표적인 객체인 프로그램 소스 코드의 표절 검사를 위한 방법론을 제시하였다. 프로그램의 전체적인 특징을 나타내는 토큰을 사용자가 정의하여 추출하되 프로그램의 제어흐름에 따라 순서를 배치하여 서열을 만든 후, 이 서열들의 비교에 유전체 서열 정렬 기법을 적용하였다. 그리고 모든 키워드의 일치 여부가 동일하게 적용되지 않고, 그룹에 따라 유사성 정도를 설정하여 표절 검사에 정확성을 높였다.

본 논문에서 제시한 유전체 서열의 정렬 기법을 이용한 소스 코드 표절 검사 방법론의 특징으로는 첫째, 유전체 서열의 정렬 기법을 응용, 확장하여 사용하고 있다. 둘째, 프로그램의 제어 흐름에 맞추어 서열 구조를 재구성하는 함수 호출 선형화방법으로 키워드를 추출하기 때문에, 프로그램에서 사용하지 않거나 불필요한 코드는 표절 검사를 위한 시퀀스에 포함되지 않는다. 셋째로는 일치되는 키워드에 대하여 항상 동일한 점수를 적용하지 않고, 유사 그룹으로 분할하여, 각 그룹별 키워드의 특징을 고려한 키워드 유사성 행렬을 정의하여 표절검사에 적용함으로써 정확한 표절 검사가 가능하다는 것이다

그리고 향후 연구과제로는 첫째, 각 대학생의 프로그램 과제를 CodeDNA로 추출하여 데이타베이스를 구축하고, 둘째 학급 단위의 그룹 내의 프로그램 과제 표절 검사에서 표절 의혹의 계통도를 확인하는 것, 셋째로 선형 구조를 이루는 객체인 악보의 표절 여부 검사에 관한 연구 등이 있다.

참 고 문 헌

[1] http://www.calstatela.edu/centers/write_cn/plagiarism.htm
[2] http://www.rbs2.com/plag.htm
[3] http://www.gyosuclub.com/
[4] Tak W.Y. and Hector. G., "Duplicate detection in information dissemination," Proc. Very Large Databases Conference, pp. 66-77, 1995.
[5] Alan P. and James O.H., "Computer algorithms for Plagiarism Detection," IEEE Transactions on Education, Vol.32, No.2, pp. 94-99, 1989.
[6] http://www.plagiarism.org
[7] http://www.integriguard.com
[8] http://www.canexus.com/eve/abouteve.shtml
[9] http://www.copycatch.freeserve.co.uk
[10] http://www.wordcheksystems.com/.
[11] Sergey B, James D. and H.G, "Copy detection mechanisms for digital documents," Proc. ACM SIGMOD International conference on Management of data, pp. 398-409, 1995.
[12] http://www.few,vu.nl/~dick/sim.html
[13] http://glimpse.arizona.edu/javadup.html
[14] Antonio. S., Hong V.L., and Rynson. W.H.L., "CHECK: A document plagiarism detection system," Proc. ACM Symposium on Applied Computing, pp. 70-77, 1997.
[15] Whale, "Identification of Program Similarity in Large populations," The Computer Journal, Vol.33, No.2, pp. 140-146, 1990.
[16] Michael. J.W., "Detection of similarities in student programs: YAP'ing may be preferable to Plague'ing," Proc. SIGSCI Technical Symposium, pp. 268-271, 1992.

[17] Michael. J.W., "YAP3: improved detection of similarities in computer programs and other texts," Proc. SIGCSE'96, pp. 130-134, 1996.
[18] http://ftp.cs.berkeley.edu/~aiken/moss.html
[19] http://wwwipd.ira.uka.de:2222/
[20] 이광근 교수와의 서신, private communication.
[21] 조환규, "Genomic Sequence alignment and its application for Computing Linear Structure Similarity," 2002년 제 1차 한국생물정보학회 워크샵, 2. 2002.
[22] http://www2.ebi.ac.uk/clustalw/
[23] Julie D.T.,Desmond G.H., and Toby. J.G.,, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," Nucleic Acids Res. Vol,22, No.22, pp. 4673-4680, 1994.
[24] Jeong-Hyeon C., Ho-Youl J., Hey-Sun K. and Hwan-Gue C., "PhyloDraw: a phylogenetic tree drawing system," Bioinformatics, Vol.16, No.11, pp. 1056-1058, 2000.

강 은 미

2001년 부산대학교 전자계산학과 졸업(학사). 2001년~현재 부산대학교 전자계산학과 석사과정. 관심분야는 프로그램 표절, 생물정보학

황 미 녕

2000년 부산대학교 전자계산학과 졸업 2002년 부산대학교 전자계산학과 석사 2002년 한국전자통신연구원을 거쳐 현재 한국과학기술정보연구원 바이오인포매틱스센터에 재직 중. 관심분야는 정보검색, 데이타클러스터링, 생물정보데이타의 유사성분석

조 환 규

1984년 서울대학교 계산통계학과 졸업(학사). 1986년 한국과학기술원 전자계산학과 졸업(공학석사). 1990년 한국과학기술원 전자계산학과 졸업(공학박사). 1990년~현재 부산대학교 전기전자정보컴퓨터공학부 교수. 관심분야는 그래프 이론, 생물정보학, 그래픽스