DOI: 10.3745/KIPSTA.2009.16A.6.453

굼벨 분포 모델을 이용한 표절 프로그램 자동 탐색 및 추적

지 정 훈[†] · 우 균[††] · 조 환 규[†††]

요 약

소프트웨어의 지적 재산권 보호 및 인증에 대한 관심과 중요성이 커지면서 소프트웨어에 대한 표절 탐색 및 보호, 판단에 대한 연구가 활발하게 진행되고 있다. 지금까지 표절에 대한 연구는 주로 속성 계산, 토큰 패턴, 프로그램 파스트리, 유사도 측정 알고리즘 등을 이용해 두 프로그램을 비교하는데 초점을 두었다. 이와 더불어, 표절과 협동(collaboration)을 구분하는 것은 표절연구에서 매우 중요하다. 본 논문에서는 극단치 분포 확률 모델을 이용한 소스코드 클러스터링을 위한 알고리즘을 제안한다. 본 논문에서는 먼저 두 프로그램 P_a와 P_b 의 유사도를 측정하는 비대칭거리측정함수 $pdist(P_a, P_b)$를 제안하고, 모든 소스코드 쌍에 대해 $pdist(P_a, P_b)$를 통해 측정된 유사도를 간선무게로 하는 표절방향그래프(PDG)를 생성한다. 그리고 본 논문에서는 표절방향그래프를 굼벨거리그래프(GDG)로 변환한다. $pdist(P_a, P_b)$ 점수 분포는 극단치 확률 분포로 잘 알려진 굼벨분포(Gumbel distribution)와 매우 유사하다. 또한, 본 논문에서는 의사표절(pseudo- plagiarism)을 새롭게 정의한다. 의사표절은 프로그램의 강한 기능적 제약사항으로 인해 발생하는 가상 표절의 한 종류이다. 본 논문에서는 ICPC(International Collegiate Programming Contest)와 KOI(Korean Olympiad for Informatics) 대회에 제출된 18개 프로그램 그룹의 700개 이상의 소스코드에 대해 실험을 진행하였다. 실험결과 프로그램 그룹에 포함된 표절 프로그램들을 찾았으며, 소스코드 클러스터링 알고리즘은 의사표절과 실제표절 프로그램 그룹을 효과적으로 구분하였다.

키워드 : 의사표절, 표절탐색, 소스코드 클러스터링, 굼벨분포모델, 지역정렬

Automated Detecting and Tracing for Plagiarized Programs using Gumbel Distribution Model

Jeong-Hoon Ji[†] · Gyun Woo[††] · Hwan-Gue Cho[†††]

ABSTRACT

Studies on software plagiarism detection, prevention and judgement have become widespread due to the growing of interest and importance for the protection and authentication of software intellectual property. Many previous studies focused on comparing all pairs of submitted codes by using attribute counting, token pattern, program parse tree, and similarity measuring algorithm. It is important to provide a clear-cut model for distinguishing plagiarism and collaboration. This paper proposes a source code clustering algorithm using a probability model on extreme value distribution. First, we propose an asymmetric distance measure $pdist(P_a,P_b)$ to measure the similarity of P_a and P_b. Then, we construct the Plagiarism Direction Graph (PDG) for a given program set using $pdist(P_a,P_b)$ as edge weights. And, we transform the PDG into a Gumbel Distance Graph (GDG) model, since we found that the $pdist(P_a,P_b)$ score distribution is similar to a well-known Gumbel distribution. Second, we newly define pseudo-plagiarism which is a sort of virtual plagiarism forced by a very strong functional requirement in the specification. We conducted experiments with 18 groups of programs (more than 700 source codes) collected from the ICPC (International Collegiate Programming Contest) and KOI (Korean Olympiad for Informatics) programming contests. The experiments showed that most plagiarized codes could be detected with high sensitivity and that our algorithm successfully separated real plagiarism from pseudo plagiarism.

Keywords : Pseudo-Plagiarism, Plagiarism Detection, Source Code Clustering, Gumbel Distribution Model, Local Alignment

1. 서론 및 연구동기

최근에는 창작물에 대한 지적재산권 보호 및 인증에 대한 사회적 관심이 높아졌으며, 이에 따른 표절에 대한 사회적 관심도 크게 향상되었다. 또한, 디지털 저장매체의 발달과 인터넷 확산에 따라 표절은 갈수록 그 정도가 심해지고 있으며 그 방법 또한

† 준 회 원 : 부산대학교 컴퓨터공학과 박사과정
†† 종신회원 : 부산대학교 컴퓨터공학과 부교수(교신저자)
††† 정 회 원 : 부산대학교 컴퓨터공학과 교수
논문접수 : 2009년 7월 6일
수 정 일 : 1차 2009년 9월 24일
심사완료 : 2009년 9월 24일

더욱 교묘해지고 있다[1, 2] 특히, 문학작품이나 음악, 드라마, 논문 등, 표절로 인한 사회적 문제 또한 그 수가 크게 증가하였다.

프로그램 표절은 타인의 프로그램 소스코드를 복사하거나 간단한 편집과정을 통해 동일한 기능을 수행하는 소프트웨어를 제작하는 행위를 말한다[3]. 문서와 프로그램 표절은 그 형태가 다양하고 사람에 의해 수동으로 판단하기가 어렵기 때문에 오래 전부터 자동화된 표절 탐색 시스템이 요구되었으며[4, 5], 이에 대한 많은 연구가 진행되고 있다[6-10]. 국내에서도 프로그램 표절 분쟁 해결 및 소프트웨어 지적 재산권 보호, 교육을 위한 컴퓨터프로그램보호위원회(SOCOP)가 조직되었으며, 프로그램 유사성 비교 소프트웨어인 exEYEs가 개발되어 무료로 배포하고 있다. exEYEs는 8가지의 프로그래밍 언어로 작성된 소스코드와 일반 텍스트 파일의 유사성을 비교해 표절을 검사할 수 있다. exEYEs는 다음의 사이트에서 다운로드 받을 수 있다.

http://www.socop.or.kr/

대학에서도 학생들 사이의 과제 보고서 표절은 심각한 수준에 이르렀다고 할 수 있다. 한 연구에 따르면, MIT 대학의 기초 프로그래밍 강의 수강생들 중 약 30%의 학생들이 표절로 규제를 받았다는 조사 결과가 있다[11]. 또 다른 연구는 대학에서 10% 이상의 학생들이 과제를 수행하기 위해 표절을 해본 경험이 있었다는 결과가 있다[12]. 이와 같은 사실은 대학 내에서 표절이 심각한 문제가 될 수 있다는 것을 나타낸다.

프로그래밍 관련 강의에서 제출되는 프로그램 소스코드들에서 표절을 찾는 것이 어려운 이유는 사람에 의해 모든 학생들의 소스코드를 비교하는 것은 시간적으로 거의 불가능에 가깝기 때문이다. 예를 들어, 만약 한 강의에서 100개 이상의 소스코드가 제출되었다면 모든 소스코드를 비교하기 위해서는 5000쌍 이상 코드를 비교해야 한다. 한 쌍의 소스코드를 검사하는데 평균 1분이 소요된다고 가정하더라도 1일 평균 8시간 검사기준으로 약 10일 정도가 소요된다. 이와 같이 수강인원이 많은 오프라인(off-line) 강의나 인터넷을 통한 원격 강의에서는 자동화된 프로그램 표절 탐색 및 추적 시스템이 필수적으로 요구된다.

프로그램 표절 탐색은 주로 두 소스코드 사이의 유사도를 계산하고 이를 기준으로 프로그램 집합에서 표절된 프로그램 쌍들을 찾아낸다. 현재까지 개발된 대부분의 표절탐색 시스템들은 잘 알려진 표절공격들을 효과적으로 찾는 방법에 초점이 맞추어졌으며, 이를 위한 여러 가지 방법이 제안되었다. <표 1>은 이전의 표절 연구에서 잘 알려진 표절공격 방법들이다[11].

프로그램 표절 연구에서 중요하게 고려되어야 하는 또 다른 문제는 표절과 협동(collaboration) 작업을 구분하는 것이다[13]. 두 프로그램 사이의 유사도만으로 표절과 협동 작업을 구분하기는 어렵다. 예를 들어, 학습 능력이 부족한 학생이 우수한 학생과 서로 협력하여 프로그래밍 과제를 수행했다면 이는 교육적으로 허용되는 행위로 표절과는 다르게 평가해야 한다. 하지만 두 학생의 프로그램은 거의 비슷하게 작성되고 표절 탐색 결과 표절된 프로그램으로 판정된다. 또 다른 예로 학생들이 고정된 크기의 배열을 이용하여 "버블소트(bubble sort)" 프로그램을 작성한다면 표절검사 방법에 상관없이 대부분의 프로그램 사이의 유사도가 높게 측정될 것이다. 그 이유는 프로그램의 제약조건이 매우 제한적이기 때문이다. 이런 경우에도 유사도가 높은 프로그램들을 모두 표절된 프로그램으로 볼 수 없다.

〈표 1〉 프로그램 표절 공격 방법들

1. 코드 복사, 주석 수정
2. 공백문자(white space) 및 코드형식 변환
3. 식별자(identifier) 이름 변환
4. 코드 블록 및 문장(statement) 재배치
5. 표현식에서 연산자 및 피연산자 순서 변경
6. 데이터 타입 변환 및 불필요한 문장 추가
7. 프로그램 제어 구조 수정

본 논문에서는 표절로 검출된 프로그램들로부터 위와 같은 경우를 구분하기 위하여 의사표절(pseudo-plagiarism)을 새롭게 정의한다. 그리고 극단치 분포 확률 모델을 이용하여 의사표절과 실제표절(real plagiarism)을 구분하는 방법을 제안한다. 본 논문에서는 확률 모델과 함께 의사표절과 실제표절을 최종 판정을 위한 클러스터링 알고리즘을 제안한다.

본 논문은 다음과 같이 구성된다. 2절에서는 소스코드 표절검사 방법에 대한 이전 연구들에 대해 소개한다. 3절에서는 본 논문에서 구현하는 표절검사 시스템의 구조에 대해 소개하고 4절에서는 두 프로그램 P_a와 P_b 사이의 유사도 계산을 위한 비대칭 거리 측정 함수($Asym(P_a, P_b)$)에 대해 설명한다. 5절에서는 의사표절을 새롭게 정의하고 의사표절과 실제표절을 구분하기 위한 확률적 모델과 클러스터링 알고리즘을 정의한다. 그리고 6절에서는 실험을 통해 이 논문에서 제안한 방법의 효과를 살펴본다. 마지막으로 7절에서 결론을 맺는다.

2. 관련 연구

2.1 표절검사 시스템

표절에 대한 연구는 정보검색(information retrieval) 및 문서처리(document processing)와 관련된 주제로 오래전부터 연구되었으며, 표절검사를 위한 많은 시스템들이 개발되었다[6-10]. 이들 표절검사 시스템들은 크게 일반 문서 표절검사 시스템과 소스코드 표절검사 시스템으로 나눌 수 있다.

일반 문서 표절 검사 시스템은 영문이나 한글 텍스트로 작성된 문서들 사이의 표절을 탐색한다. 일반 문서 표절은 원본 자료의 출처를 밝히지 않고 인용하거나 자신의 의견인 것처럼 문서를 작성하는 것을 말한다. 문서 표절의 경우, 짧은 시간에 새로운 문서를 만들어내기 때문에 비교적 간단한 기법, 즉 (1) 문단의 추가/삭제/재배치, (2) 부분 문장 편집, (3) 동의어를 이용한 일부 단어 수정, (4) 원본 문서 복사 등이 표절 기법으로 사용된다. 일반 문서의 표절검사를 위한 자동화 시스템으로는 Plagiarism.org[14], IntegriGuard[15], EVE2[16], CopyCatch[17] 등이 있다. 이들 시스템들은 주로 지문법(fingerprints)을 이용해 표절된 문서를 탐색한다.

소스코드 표절검사 시스템들은 프로그램의 소스코드를 입력으로 하여 전체 유사도를 계산하고 유사구간을 찾아낸다. 초기의 소스코드 표절검사 시스템들은 일반 문서 표절검사에서와

〈표 2〉 일반 문서와 소스코드를 위한 표절검사 시스템

시스템	검사범위	알고리즘	사용	비용
Plagiarism.org	일반문서	지문법	On-line	무료
IntegriGuard	일반문서	비공개	On-line	$4.95/월
EVE2	일반문서	비공개	Stand-alone	$19.99
CopyCatch	학생과제	lexical matching	Stand-alone	무료
SIM	소스코드	지역정렬	Stand-alone	무료
YAP3	소스코드	Greedy-Tiling-String	Stand-alone	무료
Clonechecker	소스코드	비공개	Stand-alone	상용
MOSS	소스코드	windowing	Web service	무료
JPlag	소스코드	Greedy-Tiling-String	Web service	무료
SID	소스코드	Data Compression	Web service	무료
CodeMatch	소스코드	지문법	Stand-alone	상용

동일하게 단순 스트링 비교 또는 지문법을 사용하였다. 그 후 소스코드 표절검사 연구에서는 프로그램의 구조적 특징을 고려한 검사 기법들이 제안되었다. 소스코드 표절검사 방법은 2.2절에서 설명한다. <표 2>는 일반 문서와 소스코드를 위한 잘 알려진 표절 시스템과 그들의 특징을 요약한 것이다.

2.2 프로그램 표절검사 기법

소스코드 표절검사 방법은 크게 속성계수법(attribute counting)과 구조적 검사기법으로 나눌 수 있다[18]. 속성계수법은 소프트웨어 측정값(software metric)에 기초하여 두 프로그램 사이의 유사도를 계산한다. 예를 들어, Halstead's 소프트웨어 측정법은 두 프로그램 사이의 정규화된 유사도를 구하는데 사용된다[19]. 이 방법은 프로그램에 사용된 연산자 및 피연산자의 종류와 이들이 프로그램에 나타난 횟수를 비교하여 두 프로그램 사이의 유사도를 계산한다. 최근에는 압축 알고리즘을 이용한 검사 방법이 소개되었다. SID 표절검사 시스템은 Kolmogorov 복잡도에 근거한 압축 알고리즘을 이용하여 두 프로그램의 유사도를 계산한다[10].

속성계수법은 프로그램 소스코드의 특성 변수들을 해시값(hashing value)으로 바꾸어 비교하기 때문에 빠르게 표절검사를 수행할 수 있지만, 두 프로그램 사이의 유사구간을 찾을 수 없다는 단점이 있다. 또한 이 방법은 사용되지 않는 코드 삽입 및 프로그램 실행과 관계없는 코드 삽입과 같은 표절공격에 취약하다.

최근에는 속성계수법보다 구조적 검사기법이 소스코드 표절검사에 많이 이용된다. 그 이유는 일반 문서와 다르게 소스코드는 코드 블록, 제어문, 함수와 같은 프로그래밍 언어에 따른 구조적 특징이 고려되어야 하기 때문이다. 프로그램 표절검사를 위한 구조적 검사기법은 크게 두 단계로 진행된다. 먼저 소스코드를 입력으로 받아서 중간표현의 객체를 생성한다. 구조적 검사기법에서는 소스코드 문자열, 키워드 서열[8, 20-22], 파스트리[23] 등이 주로 사용된다. 소스코드 문자열은 프로그램 소스코드를 하나의 긴 문자열로 연결하여 비교한다. 그리고 키워드 서열은 소스코드를 파싱(parsing)하면서 사전에 정의된 주요 키워드를 추출하여 일련의 서열로 만든다. 마지막으로 파스트리는 컴파일러의 전단부에 해당하는 파서(parser)가 생성하는 프로그램 파스트리를 이용하여 표절검사를 수행한다. 중간표현을 생성한 다음에는 유사도 측정 알고리즘을 이용하여 두 중간표현의 정규화된 유사도를 계산한다. 구조적 검사기법의 유사도 계산에는 Greedy-String-Tiling[7, 9], 지역정렬(local alignment)[8, 21-22], 파스트리 비교 알고리즘[23-25] 등이 주로 사용된다.

지금까지 개발된 일반 문서와 프로그램 소스코드 표절검사 시스템들은 문서 및 소스코드 집합을 입력으로 받아서 각 프로그램 쌍들에 대한 유사도와 유사구간을 탐색한다. 하지만, 소스코드 표절검사에서 중요하게 고려되어야 하는 요소인 프로그램의 기능적 제약사항(functional requirement) 및 협업을 고려하지 않고 있다. 대부분의 표절 검사 시스템에서 최종 표절 판정은 검사자의 주관적 관점에 따라 결정된다.

3. 표절검사 시스템

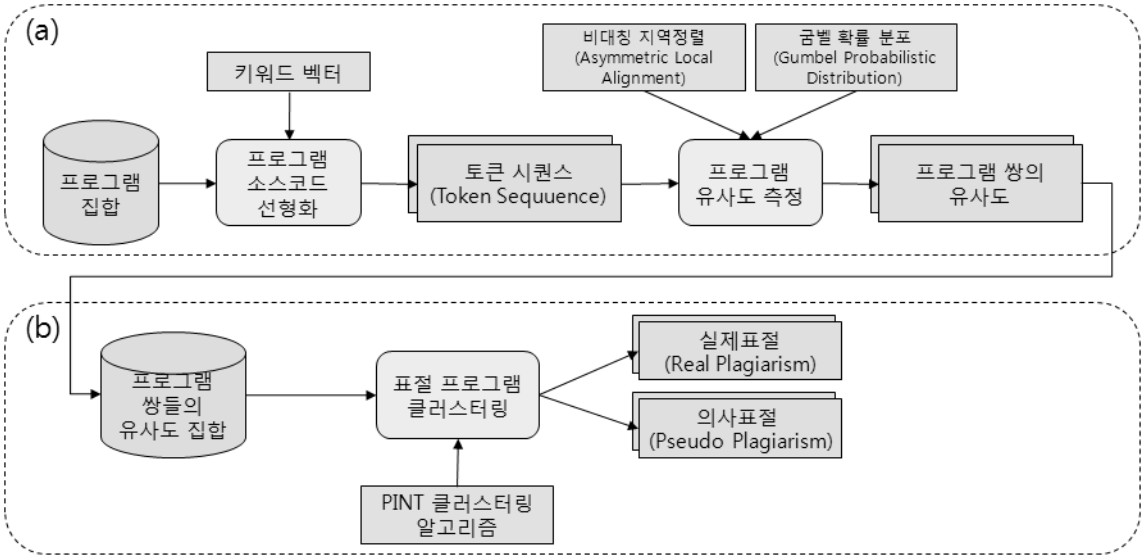
3.1 시스템 구조

본 절에서는 프로그램 의사표절(Pseudo-Plagiarism) 탐색을 위한 자동 표절검사 시스템인 PINT(Plagiarism INvestigating Tool)와 극단치 확률 분포 모델을 제안한다. PINT의 표절검사는 두 단계로 진행된다. 첫 번째 단계는 프로그램 소스코드를 입력으로 하여 소스코드 파싱을 거쳐 중간표현인 토큰 서열(token sequence)을 생성하고 각 프로그램 쌍들의 유사도를 계산한다. 다음으로 PINT 클러스터링 알고리즘을 이용해 의사표절과 실제표절(real plagiarism)을 구분한다. (그림 1)은 PINT 표절검사 시스템의 구조도를 보여준다.

(그림 1)의 (a)에서는 프로그램 소스코드 집합을 입력으로 받는다. 입력 소스코드는 C 또는 C++ 언어로 작성된 소스코드를 대상으로 한다. 표절검사 시스템 전단부에서는 프로그램 정적 추적(Program Static Tracing)을 통해 소스코드로부터 토큰 시퀀스를 생성한다. 토큰 시퀀스를 생성할 때, PINT는 사전에 정의된 키워드 벡터를 이용한다. 또한, 선형화에 사용된 키워드 벡터는 유사도 계산을 위한 지역정렬에서도 사용된다. 다음으로 PINT는 비대칭 지역정렬(Asymmetric Local Alignment)와 굼벨 분포(Gumbel Distribution) 모델을 이용해 프로그램 쌍들의 유사도를 구한다.

지역정렬을 이용하는 표절검사 시스템[8, 20]들은 유사도 계산에 고정적(fixed) 정렬 매트릭스를 이용한다. PINT의 비대칭 지역정렬에서는 프로그램 집합으로부터 추출한 키워드 벡터를 이용해 적응적(adaptive) 정렬 매트릭스를 이용한다. 또한, PINT 시스템에서는 표절검사의 정확성 향상을 위해 굼벨 분포 모델을 통해 프로그램 쌍들의 유사도 분포를 고려한다. (그림 1)-(a)에서는 최종적으로 프로그램 쌍들의 유사도를 기록한 중간파일을 생성한다.

(그림 1)-(b)에서는 본 논문에서 제안하는 클러스터링 알고리즘을 이용해 의사표절과 실제표절을 구분한다. PINT 클러스터링 알고리즘은 표절 그래프(Plagiarism Graph)를 기반으로 하고 있다. 표절검사 결과 실제표절에 해당할 경우, 시스템에서는 실제표절에 해당하는 프로그램 쌍들에 대한 유사도와 유사구



(그림 1) PINT 표절검사 시스템 구조도. (a) 프로그램 소스코드 유사도 계산, (b) 실제 표절된 소스코드 검사를 위한 클러스터링

간을 출력한다. PINT 클러스터링 알고리즘에 대한 정의는 5절에서 자세하게 설명한다.

3.2 소스코드 선형화(Source Code Linearization)

프로그램 소스코드 선형화는 표절검사 단계에서 가장 먼저 수행된다. 소스코드 선형화는 프로그램 집합의 소스코드로부터 파싱을 하면서 사전에 정의된 프로그램 키워드를 추출하는 작업이다. 소스코드 선형화에 정의된 키워드 벡터는 프로그래밍 언어에 따라 다르다. 일반적으로는 프로그래밍 언어에서 사용하는 키워드들이 토큰으로 사용된다. 예를 들어, C 언어의 경우 int, for, return 등과 같은 키워드들이 토큰으로 사용된다.

프로그램 집합으로부터 추출된 키워드들은 프로그램의 구조적 특징(structural characteristic)들을 반영한다. 본 논문에서는 키워드 벡터 K는 지역정렬에서 사용된다. 키워드의 전체 개수를 r이라고 정의하면, 키워드 백터를 $K=<k_1,k_2,...k_r>$ 같이 정의할 수 있다. PINT는 총 81개의 키워드를 사용한다 ($r=81$). (그림 2)-C 언어로 작성된 프로그램의 선형화 예를 보여준다.

(그림 2)의 왼쪽 부분은 C언어로 작성된 $main()$과 $swap()$ 함수를 나타낸다. 그리고 (그림 2)의 오른쪽 부분은 함수에 대한 토큰 서열을 나타낸다. 소스코드 선형화에서는 정적 분석을 통한 추적(tracing) 방법을 사용한다. 선형화를 수행할 때, 사용자 정의(user-defined) 함수 호출을 만날 경우, 선형화 지점을 호출되는 함수로 이동하여 선형화를 진행한다. (그림 2)에서는 $main()$ 함수에서 $swap()$ 함수 호출을 만났을 때, FUNC_CALL 토큰을 기록하고, $swap()$ 함수로 선형화 지점을 이동시킨다. $swap()$ 함수의 선형화 작업을 모두 마치면 다시 $main()$으로 돌아와서 선형화를 진행한다. 프로그램 정적 추적의 장점은 불필요한 함수 코드 삽입, 함수 재배치와 같이 빈번하게 사용하는 표절 공격을 효율적으로 탐색할 수 있다.

Source Code	Token Sequence

1. int main() {
2. int num1 = 100;
3. int num2 = 200;
4.
5. swap(&num1, &num2);
6. printf("num1 = %d, num2 = %d", num1, num2);
7. }
8.
9. void swap(int *m, int *n) {
10. int *temp;
11.
12. temp = m;
13. m = n;
14. n = temp;
15. }
	1. FUNC_CALL, # main(),
	2. BLOCK_START,
	3. INT, ASSIGNMENT,
	4. INT, ASSIGNMENT,
	5. REFERENCE, REFERENCE
	6. FUNC_CALL, # swap(int*, int*)
	7. BLOCK_START,
	8. INT, PTR
	9. ASSIGNMENT
	10. ASSIGNMENT
	11. ASSIGNMENT
	12. BLOCK_END, FUNC_END, # swap
	13. UNREACHABLE_FUNC, # printf
	14. BLOCK_END,
	15. FUNC_END # main

(그림 2) 프로그램 선형화, 프로그램 소스코드와 토큰 서열

4. 프로그램 의사표절 탐색 모델

4.1 프로그램 유사도 계산

PINT는 프로그램 선형화 다음으로 지역정렬(local alignment)을 수행한다. 지역정렬을 적용하기 전에 프로그램으로부터 서열을 준비한다. 먼저 두 프로그램 P_a와 P_b로부터 추출한 키워드 서열을 r_a와 r_b라고 정의한다. 여기서, 생물학적 관점에서 r_x는 프로그램 P_x의 'DNA'로 가정할 수 있다.

지역정렬[26]은 Smith와 Waterman에 의해 제안된 정렬방식으로 두 서열 사이의 유사구간을 찾기 위해 오래전부터 매우 많이 사용된 방법이다. 지역정렬은 본 논문의 표절검사에서 사용하는 기본 알고리즘이다. 지역정렬을 사용하는 이전 표절검사 시스템들은 고정적 점수 행렬(fixed scoring matrix-일치:+1, 불일치:-1, 갭(gap) 삽입:-2)을 사용하고 있다. 본 논문에서는 고정적 점수 행렬 대신 적응적 점수 행렬(adaptive scoring matrix)을 사용한다. 적응적 점수 행렬(W_D)는 프로그램 그룹(D)로부터 추출한 키워드들의 출연 빈도(frequency)에 의해 동적으로 생성된다. 프로그래밍 과제와 같이 동일한 목적으로 작성된 프로그램 집합을 D라 하면, f_i는 프로그램 집합 D의 키워드 K_i의 출연 빈도를 나타낸다. 본 논문에서 사용하는 적응적 지역정렬의 점수 행렬을 정의하면 다음과 같다.

$$W_D = \begin{cases} -\alpha \cdot \log_2(f_i \cdot f_j) & (\text{if } f_i = f_j) \\ +\beta \cdot \log_2(f_i \cdot f_j) & (\text{if } f_j \neq f_j) \\ \gamma \cdot \log_2(f_i^2) & (\text{갭}(gap)\text{삽입}) \\ \delta \cdot \log_2(f_j^2) & (\text{갭}(gap)\text{삭제}) \end{cases}$$

적응적 지역정렬의 점수행렬은 프로그램 그룹에서 빈번하게 사용된 키워드(예를 들어, '=')들의 일치에 대해서는 낮은 점수를 부여하고, 프로그램에서 자주 사용되지 않는 키워드(예를 들어, 'if' or 'switch')들에 대해서는 높은 점수를 할당한다. 불일치나 갭에 의한 감점(penalty)에 대해서도 동일하다. 실험결과 적응적 지역정렬은 고정적 지역정렬에 비해 '의미 없는 코드 삽입' 및 '키워드 수정' 등과 같은 표절공격에 대해 더욱 견고하다. <표 3>은 실험에 사용된 프로그램 그룹의 키워드 출현빈도를 보여준다. <표 3>에서 많이 사용되지 않은 'switch' 키워드의 정렬 점수는 빈번하게 사용된 '=' 키워드에 비해 4배 더 높다.

그러므로 적응적 지역정렬의 점수 매트릭스는 프로그램 그룹에 따라 매번 다르게 사용된다. 또한, 점수 매트릭스에서는 4가지 제어 변수($\alpha,\beta,\gamma,\delta$)를 이용하여 실험적으로 프로그램 그룹의 특성에 따라 매트릭스를 조정할 수 있다. 본 논문에서는 4가지 제어 변수를 $\alpha=0.5, \beta=0.5, \gamma=0.4, \delta=1.2$로 설정하였다. 그 이유는 프로그램의 제어구조에 대한 이해가 없이 표절을 할 경우, 코드를 삽입하는 것보다 기존의 코드를 삭제하는 것이 더욱 더 어렵기 때문이다. 본 논문에서는 두 프로그램의 유사도 계산을 위해 비대칭 거리 측정법($Asym()$:asymmetric distance metric)을 제안한다. $Asym()$ 메트릭은 Smith-Waterman 알고리즘을 표절검사에 적합하도록 확장한 것이다.

〈표 3〉 ICPC 2006-C 프로그램 그룹의 키워드 빈도수

높은 빈도 키워드	출현빈도	낮은 빈도 키워드	출현빈도
Assignment "="	14.00%	"switch"	0.01%
Block Start "{"	11.83%	"-="	0.02%
Block End "}"	11.83%	"void"	0.02%
Increment "++"	6.76%	"goto"	0.03%
"if"	6.44%	Bit OR "|"	0.03%

[정의 1] $Asym(P_a,P_b)$는 적응적 점수 행렬인 W_D를 이용해 두 프로그램 P_a와 P_b의 키워드 서열인 r_a와 r_b사이의 가장 유사한 구간의 지역정렬 점수를 나타낸다.

본 논문의 $Asym()$ 지역정렬 방법은 이전의 지역정렬 방법과 다른 두 가지 특징이 있다. 첫 번째는 적응적 점수 행렬을 사용한다는 것이고, 두 번째는 4가지 제어변수를 사용한다는 것이다. 이로 인하여, 비대칭 거리 측정법인 $Asym()$은, P_a를 기준으로 하였을 때 P_b와의 유사성과 P_b를 기준으로 하였을 때 P_a와의 유사성이 다르다는 특징이 있다($Asym(P_a,P_b) \neq Asym(P_b,P_a)$).

4.2 프로그램 유사도 측정 분포

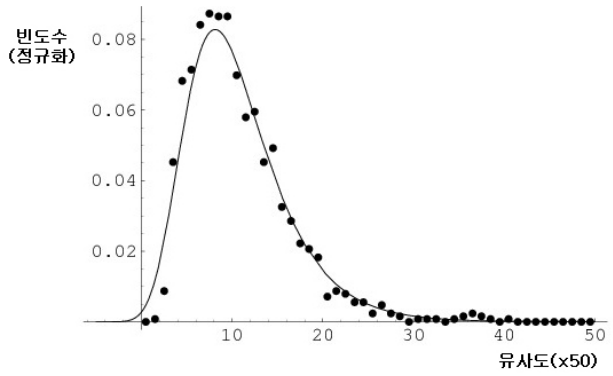
표절검사에서 독립된 두 프로그램 사이에 유사한 구간이 발견되었다면, 두 프로그램 사이에 실제 표절이 발생한 것인지, 다른 이유 때문에 우연히 비슷하게 코딩이 된 것인지에 대해 조사해야 한다. 많은 프로그램들 사이에 넓은 공통구간이나 부분적 일치 구간이 공통적으로 나타날 경우, 이것은 제약사항에 의해 만들어졌을 확률이 높다. 그러므로 독립된 두 프로그램 P_a와 P_b 사이의 유사도 점수들의 분포를 이해하는 것은 표절검사에서 중요하다. 유사도 점수 분포를 고려하는 것은 본 논문에서 제안하는 방법과 이전의 표절검사 시스템 사이의 가장 큰 차이점이다.

지역정렬 점수의 분포에 대한 이해는 효과적이고 신뢰할 수 있는 생물학적 서열(DNA, RNA, Protein 등) 탐색에 있어 중요하기 때문에 오래전부터 연구가 진행되어왔다[27, 28]. 갭을 사용하지 않는(ungapped) 지역정렬의 점수는 이미 극단치 분포 중의 하나인 굼벨 분포(Gumbel Distribution)의 특징이 있다는 것을 연구를 통해 밝혀졌다[29]. 하지만 갭을 사용하는(gapped) 지역정렬의 점수가 굼벨 분포의 특징을 따르는지는 명확하게 밝혀지지 않았다. 단지, 실험을 통해 갭을 사용하는 지역정렬도 파라미터 조정을 통해 굼벨 분포의 형태를 따른다는 것이 알려져 있다.

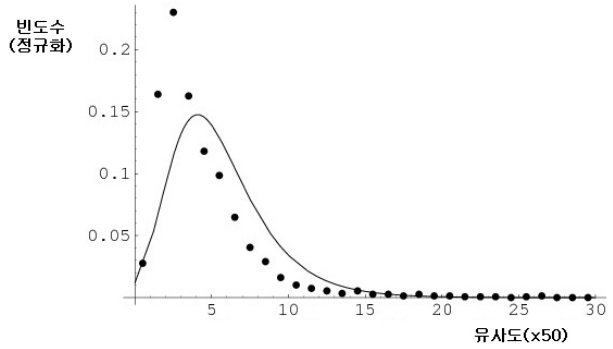
굼벨 분포의 확률 밀도 함수 $Gumbel(x;\mu,\rho)$과 두 개의 조정 파라미터 μ와 ρ는 다음과 같다.

$$Gumbel(x;\mu,\rho) = \frac{1}{\rho} \cdot \exp\left(\frac{-(x-\mu)}{\rho}\right) \cdot \exp\left(-\exp\left(\frac{-(x-\mu)}{\rho}\right)\right)$$

여기서 두 가지 조정 파라미터 μ와 ρ는 다음과 같다.



(그림 3) KOI2006-E2 그룹에 대한 $Asym()$ 분포(점), 굼벨 분포(실선)와 유사하다



(그림 4) ICPC2005-A 그룹에 대한 $Asym()$ 분포(점), 유사도 분포가 굼벨분포를 많이 벗어난다

$$\mu = a_D - \rho \cdot r$$
$$\rho = \sqrt{6} \cdot d_D / \pi$$

위 식에서 a_D는 평균을 나타내고, d_D는 집합 D의 유사도 점수들 사이의 표준편차를 나타낸다. 그리고 r은 Euler-Mascheroni 상수이다($r = 0.57721...$).

5. 클러스터링과 표절검사

5.1 실제표절과 의사표절(Pseudo Plagiarism)

본 절에서는 표절검사에서 기본적으로 해결해야 하는 문제에 대해 다룬다. 예를 들어, 프로그래밍 과제를 수행할 때 다음과 같은 요구사항들이 주어진다고 가정한다. 프로그래머는 코드를 작성할 때, 특정 자료구조 또는 템플릿 코드(template code)를 사용해야 하며, 프로그램 수행시간 및 실행파일 크기가 기준치 이하여야 한다. 이와 같은 제약사항들로 인해 프로그램의 자료구조와 알고리즘들이 유사하게 작성될 수 있다.

학생들에게 버블 소팅(bubble sorting) 알고리즘을 이용해 정렬 프로그램을 작성하도록 하면, 서로 독립적으로 작성되었지만, 모든 프로그램들 사이의 유사도는 매우 높아지게 된다. 이와 같은 현상은 프로그래밍 대회(programming contest)에서 빈번하게 나타난다. 한 예로 ICPC 지역 예선전에서 1번과 2번 문제는 학생들이 10분 이내에 직관적으로 쉽게 작성할 수 있었으며, 모든 소스코드는 100라인 이하로 작성되었다. 이와 같은 경우, 두 프로그램의 유사도가 매우 높다고 하여 두 프로그램 사이에 표절을 했다고 단정 지을 수 없다. 이와 같은 표절의 종류는 이전의 연구에서는 정의된 바가 없다. 본 논문에서는 새로운 두 종류의 표절형태인 의사표절(pseudo plagiarism)과 실제표절(real plagiarism)을 정의하고자 한다.

[정의 2] 실제표절은 프로그래머에 의해 악의적인 의도로 표절을 한 경우를 말한다. 의사표절은 프로그램에 주어진 제약사항들로 인해 독립적으로 작성되었지만 두 프로그램 사이의 유사도가 매우 높아 표절로 탐색되는 경우이다.

의사표절(pseudo-plagiarism)을 찾아내는 것이 표절검사에서 중요한 문제가 된다. 본 논문에서는 ICPC 동아시아 결선 대회와 KOI 프로그래밍 결선 대회에서 인터넷 사용 금지 및 참가자들의 이동 금지, 정해진 시간 내에 프로그램 제출 등과 같은 제약적인 환경에서 프로그램들이 작성된 경우, 이들 사이의 $Asym()$ 값의 분포는 굼벨 분포 함수를 따른다는 것을 발견할 수 있었다. 이 경우 생물학적 서열 검색에서 널리 이용되는 굼벨 분포는 독립된 두 프로그램 P_a와 P_b의 $Asym(P_a,P_b)$ 유사도 값의 확률을 계산하기 위한 좋은 모델이 된다. (그림 3)은 굼벨 분포와 이에 대한 KOI2006의 프로그램 그룹의 $Asym()$ 값의 분포를 보여준다.

만약 프로그램 유사도가 매우 높거나 유사도 값들이 평균 유사도 분포에 비해 매우 좁은 간격으로 분포될 경우에는 $Asym()$의 분포는 굼벨 분포를 따르지 않는다. (그림 4)는 ICPC2005-A 프로그램 그룹에 대한 $Asym()$ 분포로 굼벨 분포와 다른 경우를 보여준다. 또한, 프로그램에 대해 기능적 요구사항이 매우 강하거나 구체적일 경우에도 $Asym()$은 굼벨 분포를 벗어날 수 있다. 이 경우는 의사표절이 일어나는 특징을 나타내는 전형적인 상황이다.

5.2 표절검사 알고리즘

본 절에서는 프로그램 그룹에 대해 표절된 프로그램 검출을 위한 새로운 알고리즘을 제안한다. 먼저, 표절검사 알고리즘에서는 클러스터링을 통해 의사표절과 실제표절을 구분한다. 다음으로 표절된 프로그램 쌍들이 있을 경우 사용자에게 이들을 출력한다. 그러므로 본 논문에서는 의사표절과 실제표절을 구분하기 위한 PINT 알고리즘을 제안한다. 지금부터 알고리즘 설명을 위한 몇 가지 사전 정의를 한다.

[정의 3] $pdist(p_a,p_b|D)$ - 적응적 점수 행렬 W_D를 이용해 표절거리(plagiarized distance)를 계산하는 정규화 함수 ($p_a,p_b \in D$). 여기서 $pdist(p_a,p_b|D)$는 다음과 같이 계산된다.

$$pdist(p_a,p|D) = 1 - Asym(p_a,p_b)/Asym(p_b,p_b)$$

$Asym(p_a,p_b)$는 $Asym(p_b,p_b)$ 보다 클 수 없기 때문에 $pdist(p_a,p_b|D)$ 값은 항상 1 보다 작다($0 \le pdist(p_a,p_b|D) \le 1$). 그리고 $pdist(p_a,p_b|D)$ 값이 0일 경우, 두 프로그램이 거의 유사함을 의미한다($p_a \equiv p_b$). 또한 $pdist(p_a,p_b|D)$ 함수는 비대칭 함수이다($pdist(p_a,p_b|D) \neq pdist(p_b,p_a|D)$, $pdist(p_a,p_b|\varnothing) \neq pdist(p_a,p_b|D)$). 프로그램 그룹 D가 고정된 경우에는 편의상 $pdist(p_a,p_b|D)$를 $pdist(p_a,p_b)$로 표시하겠다.

[정의 4] $PDG(V,E)$ - PDG 함수는 표절방향그래프(Plagiarism Direction Graph)로 프로그램 그룹 D와 $Asym()$ 함수로부터 구축된다. PDG 함수에서 정점(V)은 프로그램을 나타낸다($p_i \in D$). 그리고 프로그램 p_a와 p_b로 이루어진 각 프로그램 쌍들은 하나의 방향성 있는 간선(E)로 나타낸다.

PDG의 모든 간선들은 방향성과 무게(거리) 값으로 표시된다. 본 논문에서는 $PDG(E)$의 각각의 간선들의 무게를 다음과 같이 정의한다.

[정의 5] $pdist(p_a,p_b) > pdist(p_b,p_a)$의 경우, 간선 방향은 $(\overrightarrow{p_b,p_a})$로 표시하고, 반대의 경우인 $pdist(p_a,p_b) \le pdist(p_b,p_a)$는 $(\overrightarrow{p_a,p_b})$로 표시한다. PDG 간선의 무게는 $w(p_u,p_v) = \min\{pdist(p_u,p_v), pdist(p_v,p_u)\}$이다.

예를 들어, $pdist(a,b) = 0.2$와 $pdist(b,a) = 0.5$인 경우에 프로그램 b에서 프로그램 a 방향으로 표절이 발생한 것보다 a에서 b 방향으로 표절이 발생했을 확률이 높다고 가정한다. 그 이유는 일반적으로 표절된 프로그램들은 유사도가 더욱 높아지기 때문이다. PDG는 완전 그래프(complete graph)이기 때문에, 그래프로부터 의미 있는 클러스터를 생성하기 위해 표절거리를 이용해 간선들을 지우는 작업을 수행한다. PDG로부터 간선의 무게를 기준으로 임계값(threshold value) 보다 큰 간선들을 삭제하여 새로운 서브 그래프인 PDG_c를 생성한다.

[정의 6] $PDG_c(V,E)$ - PDG로부터 생성된 서브 그래프, PDG_c는 다음과 같이 생성된다. $PDG_c(V,E) = PDG(V,E) - \{(p_i,p_j)|pdist(p_i,p_j|D) > c\}$, 여기서 c는 임계값을 나타낸다.

$PDG_c(V,E)$는 간선의 무게가 c 보다 큰 간선들을 제거한다. PDG는 프로그램 그룹의 유사도를 간선으로 나타내지만, 프로그램 그룹의 유사도 분포 특징을 반영하지 못한다. 본 논문에서는 PDG 단점을 보완하기 위하여 굼벨 분포를 유사도 분석에 적용한다. PDG에 굼벨 분포를 적용하기 위해 다음을 정의한다.

[정의 7] $gdist(p_a,p_b|D)$ - 두 프로그램 p_a와 p_b 사이의 굼벨 거리(gumbel distance)로 $gdist(p_a,p_b|D) = \int_c^1 G(x;\mu_D,\rho_D)dx$ ($G()$는 굼벨 분포의 확률밀도함수이고, $c = Asym(p_a,p_b)$)와 같이 정의된다.

[정의 8] $GDG(V,E)$ - 굼벨 방향 그래프로 PDG와 동일한 방법으로 생성된다($GDG(V) = PDG(V)$와 $GDG(E) = PDG(E)$). $GDG(E)$의 간선 무게는 $gdist(p_a,p_b|D)$이다. 그리고 $GDG_c(V,E)$는 $PDG_c(V,E)$와 동일하게 정의된다.

[정의 9] $MPC_c(V,E)$ - $GDG_c(V,E)$의 최대 표절 컴포넌트(Maximal Plagiarism Component)로 $GDG_c(V,E)$에서 정점이 가장 많은 컴포넌트를 의미한다.

$pdist(a,b)$와 $gdist(a,b)$는 모두 0 ~ 1 사이로 정규화된 거리 메트릭이다. 그러나 $gdist$는 파라미터들을 통해 프로그램 그룹의 유사도 분포 특징들까지 정규화에 반영할 수 있다. 그러므로 PINT 알고리즘에서는 GDG를 이용해 클러스터링을 한다.

본 논문에서는 18개의 프로그램 그룹을 통한 실험결과, $gdist(p_a,p_b) < 0.001$에 대해 p_a와 p_b 사이에 표절이 있는 경우로 밝혀졌다. 이 경우는 $GDG(E)$의 모든 간선 무게가 0.001보다 작은 경우에 대해 표절이 발생했을 확률이 매우 높음을 의미한다. 또한, 본 논문의 실험에서는 10명 이상의 참가자들이 표절에 연관되기는 매우 힘든 것으로 나타났다. 다시 말해서, $|MPC_{0.001}(V,E)| > 10$인 경우는 의사표절이 발생했을 가능성이 매우 높다. 이 결과와 함께, PINT 알고리즘을 다음과 같이 제안한다.

[알고리즘] $PINT$ - 프로그램 그룹에서 의사표절과 실제표절 구분.

입력 - D, 기능적으로 동일한 프로그램 그룹. c, MPC에서 컴포넌트 크기를 결정하기 위한 임계치 상수

출력 - 의사표절로 판정 또는 실제표절 의심 쌍들에 대한 출력

과정1 - 소스코드로부터 프로그램 DNA_i 추출, DNA_i는 p_i의 선형화된 키워드 서열($p_i \in D$).

과정2 - 모든 DNA로부터 적응적 점수 행렬 W_D 생성

과정3 - $G(x;\mu_D,\rho_D)$와 $PDG(V,E)$, $GDG(V,E)$ 생성

과정4 - $GDG_c(V,E)$ 생성($c = 0.0001$, 실험적 임계치 상수)

과정5 - 의사표절 검사, $MPC_c(V,E) > M$인 경우 의사표절로 판정($M = 10$)

과정6 - 실제표절 검사, 표절 프로그램 쌍 출력((p_a,p_b)) $\in GDG_c(E)$

6. 실 험

본 논문에서는 18개의 프로그램 그룹에 대해 PINT 알고리즘

을 적용하여 의사표절과 실제표절을 탐색하는 실험을 진행했다. <표 4>는 18개 프로그램 그룹에 대한 실험 데이터를 보여준다. 18개 프로그램 그룹에는 하나의 의사표절 그룹(ICPC2005 Problem-A)과 실제표절 그룹(ICPC2005 Problem-E)이 포함되어 있다. 의사표절이 나타난 프로그램은 많은 대회 참가자들이 문제를 풀 수 있도록 하기 위해 매우 쉽게 출제되었기 때문에, 학생들이 제출한 프로그램들은 길이가 매우 짧으며 대부분 유사한 알고리즘을 이용해 작성되었다. 실제표절은 ICPC2005 예선에서 나타났다. ICPC2005 예선대회는 온라인(online)으로 프로그램을 제출하며, 장소 및 진행에 대한 제약사항이 없다.

<표 4>에서 "제약사항" 항목은 각 프로그래밍 대회에서 감독관에 의해 대회가 진행될 경우에는 "Yes", 그렇지 않을 경우에는 "No"로 표시하였다. "No"인 경우는 참가자들이 자신의 집이나 전산실 등에서 온라인 방식으로 진행된 것을 말한다. 실제표절이 발생한 프로그램 그룹은 ICPC 동아시아 결선대회 문제로 난이도는 중간 수준이며, 제약사항이 없는 환경에서 실시되었다.

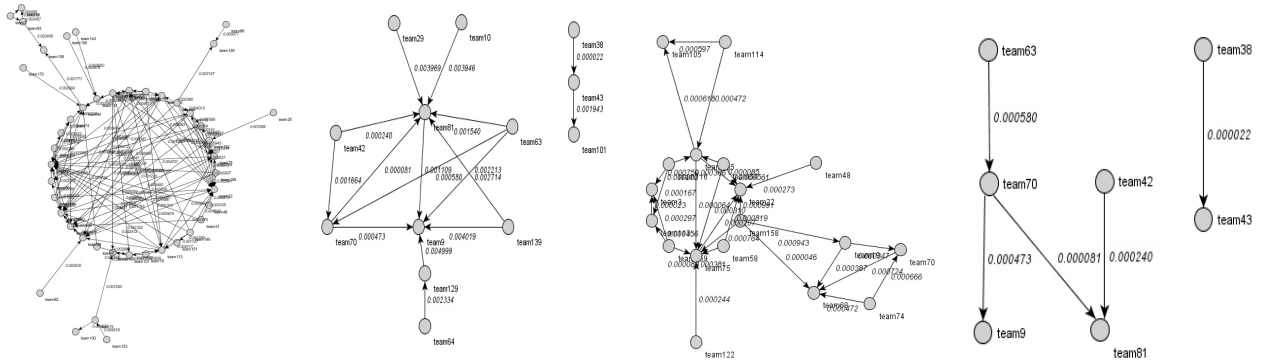
(그림 5)는 $GDG_{0.005}$와 $GDG_{0.001}$인 경우에 대한 의사표절과 실제표절 탐색 결과를 그래프를 통해 나타낸 것이다. (그림 5)-(a)와 (그림 5)-(b)는 $GDG_{0.005}(V,E)$인 경우에 대한 의사표절과 실제표절을 보여준다. (그림 5)-(a)는 $|MPC_{0.005}(V,E)|$가 45인 경우로 의사표절에 해당한다. (그림 5)-(b)는 $|MPC_{0.005}(V,E)|$가 10으로 실제 표절에 해당한다. (그림 5)-(c)와 (그림 5)-(d)는 $GDG_{0.001}(V,E)$인 경우로 (그림 5)-(c)는 의사표절을 나타낸다($|MPC_{0.001}(V,E)|=18$). 마지막으로 (그림 5)-(d)는 $c=0.001$로 했을 때, 실제표절 탐색 결과를 보여준다($|MPC_{0.001}(V,E)|=5$).

<표 5>는 의사표절과 실제표절 검사 실험의 탐색 결과를 부분적으로 보여준다. <표 5>의 "표절의심" 항목은 $GDG_{0.001}(V,E)$의 간선 개수를 나타낸다. "표절의심" 프로그램 쌍들은 사람이 직접 소스코드를 검사해 최종 표절 판정을 내린 후

〈표 4〉 실험에 사용된 테스트 프로그램 그룹 : KOI 프로그래밍 대회(2004~2006, 8그룹)과 ICPC 예선 및 본선 대회 (2004~2006, 10그룹)

그 룹	프로그램 수	프로그램 쌍	평균 코드길이	제약사항
KOI2004-H1	51	2,550	78.81	Yes
KOI2004-H2	51	2,550	163.29	Yes
KOI2004-H3	27	702	84.05	Yes
KOI2006-H1	57	3,192	143.06	Yes
KOI2006-H2	34	1,122	60.53	Yes
KOI2006-M1	46	2,070	63.78	Yes
KOI2006-M2	36	1,260	107.64	Yes
KOI2006-E2	37	1,332	86.96	Yes
ICPC2004-B	48	2,256	89.18	Yes
ICPC2004-C	22	462	55.17	Yes
ICPC2004-E	35	1,190	44.44	Yes
ICPC2005-A	153	23,256	65.30	No
ICPC2005-B	109	11,772	67.49	No
ICPC2005-E	38	1,406	44.14	No
ICPC2005-G	44	1,892	47.60	No
ICPC2006-A	180	32,220	43.77	No
ICPC2006-B	175	30,450	54.29	No
ICPC2006-C	157	24,492	58.98	No

ICPC 위원회에 보고되었다. 조사결과, 소수의 학생들이 프로그래밍 대회 중 실제로 소스코드를 표절한 것으로 밝혀졌다. 실제표절로 최종 판정 받은 학생들의 수는 <표 5>의 "확정" 항목에서 보여준다. ICPC2005-E에서 실제표절은 (그림 5)-(d)에서 Team81과 Team70, Team43과 Team38로 확인할 수 있다.



(a) $PP(GDG_{0.005})$ (b) $RP(GDG_{0.005})$ (c) $PP(GDG_{0.001})$ (d) $RP(GDG_{0.001})$

(그림 5) $GDG_{0.005}(V,E)$와 $GDG_{0.001}(V,E)$의 굼벨 거리 그래프, (a) $GDG_{0.005}$에 대한 의사표절($|MPC_{0.005}(V,E)=45$), (b) $GDG_{0.005}$에 대한 실제표절($|MPC_{0.005}(V,E)=10$), (c) $GDG_{0.001}$인 경우의 의사표절($|MPC_{0.001}(V,E)=18$), (d) $GDG_{0.001}$인 경우의 실제표절($|MPC_{0.001}(V,E)=5$)

〈표 5〉 의사표절과 실제표절 검사 결과, 6개의 표절의심 프로그램 쌍들 중 실제 두 개의 프로그램 쌍에 대해 최종 표절 확정

그룹	프로그램 수	표절의심	확정	타입
ICPC2004-C	22	1	0	실제표절
ICPC2005-A	153	29	0	의사표절
ICPC2005-E	38	5	2	실제표절

7. 결 론

본 논문에서는 지역정렬과 극단치 확률 분포인 굼벨 분포를 이용해 자동으로 표절 프로그램을 탐색하는 PINT 알고리즘을 제안하였다. PINT 알고리즘은 입력 프로그램 그룹에 대해 의사표절과 실제표절로 구분한다. 본 논문에서 제안하는 표절탐색 방법의 주요 특징을 요약하면 다음과 같다.

- 본 논문에서는 비대칭 지역정렬 함수인 $pdist(p_a,p_b|D)$를 제안하였다. 실험결과, 비대칭 표절검사 방법은 프로그램 그룹에 대해 표절 프로그램 탐색 및 표절 방향 탐색에 효과적이다.
- 본 논문에서는 $Asym()$의 유사도 점수 분포를 실험적으로 검증하였다. 동일한 기능을 수행하는 프로그램 소스코드에 대한 적응적 점수 행렬을 사용하는 비대칭 및 갭(gapped) 지역정렬의 유사도 분포는 굼벨 확률 분포를 따른다는 것을 실험을 통해 검증했다.
- 본 논문에서는 의사표절을 새롭게 정의하였으며, 의사표절 구분을 위해 $gdist(a,b)$, "Gumbel Distance", $GDG(V,E)$ 모델을 정의하였다.
- 마지막으로 의사표절과 실제표절 구분을 위한 클러스터링 알고리즘인 PINT 알고리즘을 제안하였다.

의사표절은 기능적 제약사항이 강한 프로그래밍 대회나 과제 등에서 빈번하게 발생할 수 있다. 이에 대해, PINT 알고리즘의 *MPC* 모델은 표절검사의 정확도 향상을 위한 좋은 모델이 될 수 있다.

참 고 문 헌

[1] S. Eissen and B. Stein. Intrinsic plagiarism detection. In *Proceedings of ECIR. Volume 3936 of Lecture Notes in Computer Science., Springer,* pages 565-569, 2006.

[2] S. Mann and Z. Frew. Similarity and originality in code: plagiarism and normal variation in student assignments. In *Proceedings of the 8th Austalian Conference on Computing Education*, pages 143-150. 2006.

[3] A. Parker and J. O. Hamblen. Computer algorithms for plagiarism detection. *IEEE Transaction on Education*, 32(2):94-99, 1989.

[4] B. Cheang, A. Kurnia, A. Lim, and W. Oon. On automated grading of programming assignments in an academic institution. *Computer and Education,* 41:121-131, 2003.

[5] A. Knight, K. Almeroth, and B. Bimber. An automated system for plagiarism detection using the internet. In Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2004 pages 3619-3625, 2004.

[6] A. Aiken. Moss(measure of software similarity) plagiarism detection system. 1998.

[7] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016-1038, 2002.

[8] D. Gitchell and N. Tran. Sim: a utility for detecting similarity in computer programs. In *Proceedings of the SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 266-270, 1999.

[9] M. J. Wise. Detection of similarities in student programs: Yap'ing may be preferable to plague'ing. In Proceedings of the 23rd SIGCSE Technical Symposium, 24(1):268-271, 1992.

[10] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker. Shared information and program plagiarism detection. *IEEE Transactions on Information Theory*, 50(7):1545-1551, 2004.

[11] C. Daly and J. Horgan. Patterns of plagiarism. *SIGCSE Bull.*, 37(1):383-387, 2005.

[12] M. Joy and M. Luck. Plagiarism in programming assignments. *IEEE Transactions of Education*, 42(2):129-133, 1999.

[13] J. Garter. Collaboration or plagiarism: What happens when students work together. In *Proceeding of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education(ITICSE-99)*, volume 31 of *SIGCSE Bulletin inroads*, pages 52-55, 1999.

[14] Plagiarism.org, Site available at http://www.plagiarism.org.

[15] Integriguard, Site available at http://www.integriguard.com.

[16] EVE2, Site available at http://www.canexus.com/eve.

[17] CopyCatch, Site available at http://www.copycatchgold.com

[18] K. Verco and M. Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. In *Proceedings of the 1st Australian Conference on Computer Science Education*, pages 130-134, 1996.

[19] J. Donaldson, A. Lancaster, and P. Sposato. A plagiarism detection system. In *Proceedings of the 12th SIGCSE Technical Symposium on Computer Science Education.* pages 21-25, 1981.

[20] 강은미, 황미녕, 조환규. 유전체 서열의 정렬 기법을 이용한 소스 코드 표절 검사. *정보과학회논문지: 컴퓨팅의 실제*, 9(3):352-367, June 2003.

[21] 지정훈, 우균, 조환규. 제한된 프로그램 소스 집합에서 표절 탐

색을 위한 적응적 알고리즘. *정보과학회논문지: 소프트웨어 및 응용*, 33(12):1090-1102, 2006.

[22] J. Ji, G. Woo, and H. Cho. A source code linearization technique for detecting plagiarized programs. In *ITiCSE'07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 73-77, 2007.

[23] J. Soon, S. Park, and S. Park. Program plagiarism detection using parse tree kernels. In *Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006)*, volume 4099 of *Lecture Notes in Computer Science*, pages 1000-1004. Springer, 2006.

[24] 김영철, 김성근, 최종명, 염세훈, 유재우. 구문트리 비교를 통한 프로그램 유형 복제 검사, *한국정보과학회논문지*, 30(8):792-802, 2003.

[25] 김영철, 유재우. 구문트리에서 키워드 추출을 이용한 프로그램 유사도 평가, *한국정보처리학회논문지*, 12-A(2):109-116, 2004.

[26] T. F. Smith and M. S. Waterman, Identification of common molecular subsequences, *Journal of Molecular Biology* 147, 195-197, 1981.

[27] Karlin, S. and Altschul, S.F.: Methods for assessing the statistical significance of molecular sequence features by using general scoreing schemes. In *Proceedings of the National Academy Sciences of the USA. Volume 87*. 2264-2268, 1990

[28] Altschul, S.F., Bundschuh, R., Olsen, R., and Hwa, T.: The estimation of statistical parameters for local alignment score distributions. *Nucleic Acids Research 29*, 351-361, 2001.

[29] Kschischo, M., L¨assig, M., and Yu, Y.K.: Toward an accurate statistics of gapped alignment. *Bulletin of Mathematical Biology 67*, 169-191, 2005.

지 정 훈

e-mail : jhji@pusan.ac.kr

2003년 경성대학교 컴퓨터공학(학사)

2005년 경성대학교 컴퓨터공학(석사)

2009년~현 재 부산대학교 컴퓨터공학과 박사과정

관심분야 : 프로그래밍언어 및 컴파일러, 프로그램 표절검사, 자바가상기계, 프로그램 시각화

우 균

e-mail : woogyun@pusan.ac.kr

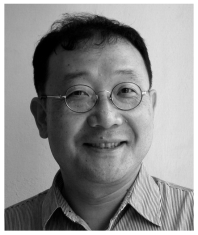
1991년 한국과학기술원 전산학(학사)

1993년 한국과학기술원 전산학(석사)

2000년 한국과학기술원 전산학(박사)

2009년~현 재 부산대학교 컴퓨터공학과 부교수

관심분야 : 프로그래밍언어 및 컴파일러, 함수형 언어, 그리드컴퓨팅, 소프트웨어 메트릭



조 환 규

e-mail : hgcho@pusan.ac.kr

1984년 서울대학교 계산통계학과(학사)

1990년 한국과학기술원 전산학(박사)

2009년~현 재 부산대학교 컴퓨터공학과 교수

관심분야 : 알고리즘 이론, 생물정보학