KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.6 pp.189~196 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.6.189

A Plagiarism Detection Technique for Source Codes Considering Data Structures

Kihwa Lee[†] · Yeoneo Kim[††] · Gyun Woo[†††]

ABSTRACT

Though the plagiarism is illegal and should be avoided, it still occurs frequently. Particularly, the plagiarism of source codes is more frequently committed than others since it is much easier to copy them because of their digital nature. To prevent code plagiarism, there have been reported a variety of studies. However, previous studies for plagiarism detection techniques on source codes do not consider the data structures although a source code consists both of data structures and algorithms. In this paper, a plagiarism detection technique for source codes considering data structures is proposed. Specifically, the data structures of two source codes are represented as sets of trees and compared with each other using Hungarian Method. To show the usefulness of this technique, an experiment has been performed on 126 source codes submitted as homework results in an object-oriented programming course. When both the data structures and the algorithms of the source codes are considered, the precision and the F-measure score are improved 22.6% and 19.3%, respectively, than those of the case where only the algorithms are considered.

Keywords : Program Plagiarism Detection, Static Analysis, Program Similarity, Similarity on Data Structures

데이터 구조를 고려한 소스코드 표절 검사 기법

이 기 화[†] · 김 연 어[††] · 우 균[†††]

요 약

표절은 불법이고 피해야 하지만 여전히 빈번하게 발생하고 있다. 특히, 소스코드 표절은 그 특성상 복사가 용이해 다른 저작물보다 더 빈번히 발생한다. 코드 표절을 방지하기 위한 다양한 연구가 있었다. 하지만 앞서 연구된 소스코드 표절 검사 기법을 살펴보면 프로그램이 알고리즘과 데이터 구조로 구성됨에도 불구하고 데이터 구조는 전혀 고려하지 않고 있다. 이 논문에서는 데이터 구조를 고려한 소스코드 표절 검사 기법을 제안한다. 구체적으로 말해서 두 소스코드의 데이터 구조를 트리 집합으로 나타내고, 헝가리안 메소드를 사용해 비교한다. 제안하는 기법의 효용성을 보이기 위해 객체지향 교과목에서 과제 답안으로 제출한 126개의 소스코드를 대상으로 실험하였다. 실험 결과 데이터 구조와 알고리즘을 모두 고려했을 때, 알고리즘만 고려한 경우보다 정확률과 F-measure가 각각 22.6%, 19.3% 향상됨을 보였다.

키워드 : 프로그램 표절 검사, 정적 분석, 프로그램 유사도, 데이터 구조 유사도

1. 서 론

예나 지금이나 아무런 노력 없이 개인의 이익을 목적으로 다른 사람의 창작물에 대한 권리를 침해하는 표절 행위는 늘 발생하고 있다. 특히 컴퓨터 프로그램은 그 특성상 수정 및 복사가 용이하기 때문에 다른 분야에 비해 표절의 대상이 되기 쉽다. 그래서 프로그램 표절 검사 기법을 연구하는 사람들은 표절된 프로그램을 효과적으로 검출하기 위해 많은 시간과 노력을 들여서 연구하고 있다.

프로그램 표절 검사 기법을 연구하는 사람들은 프로그램 표절을 효과적으로 검출하기 위해 소스코드 상의 특정 키워드 횟수와 같은 문법적 코드 정보나 텍스트, 토큰, 트리를 기반으로 하는 다양한 표절 검사 기법을 제안하였다[1, 2]. 무료 표절 검사 도구로서 비교적 우수한 성능을 보여 각광받고 있는 JPlag[3, 4]와 부산대학교 프로그래밍 언어 연구실에서 개발한 SoVAC[5, 6, 7, 8]은 토큰을 기반으로 하는 소스코드 표절 검사 도구다.

※ 본 연구는 BK21플러스, IT기반 융합산업 창의인력양성사업단에 의하여 지원되었음.
† 준 회 원 : 슈어소프트테크 전임연구원
†† 준 회 원 : 부산대학교 전자전기컴퓨터공학과 박사과정
††† 종신회원 : 부산대학교 전자전기컴퓨터공학과 교수, LG전자 스마트제어센터 행정관리부장
Manuscript Received : February 10, 2014
First Revision : April 22, 2014
Accepted : April 29, 2014
* Corresponding Author : Gyun Woo(woogyun@pusan.ac.kr)

하지만 앞서 연구된 프로그램 표절 검사 기법을 살펴보면 데이터 구조를 표절 검사에 활용한 사례는 찾아볼 수 없다. 대부분 표절 검사 기법이 알고리즘이 곧 프로그램 전체인 것처럼 알고리즘 유사도에만 집중하고 있다. Niklaus Wirth가 쓴 책 “Algorithms + Data Structures = Programs”[9]의 제목에서 보듯이 프로그램은 알고리즘과 데이터 구조로 구성된다. 따라서 진정한 의미의 프로그램 표절 검사 기법이라면 알고리즘뿐만 아니라 데이터 구조 역시 고려해야 할 것이다.

데이터 구조는 프로그램 표절 공격에도 쉽게 변하지 않는 강인도(resilience)를 가지고 있다. 일반적으로 알려진 소스코드 표절 공격이 데이터 구조에 미치는 영향은 거의 없다. 자료형 변환이 데이터 구조에 영향을 미치기는 하나 데이터 구조에 있는 자료형 변환은 이 변수에 접근하는 코드의 수정을 동반하기 때문에 어렵다. 실제로 부산대학교 2012년 2학기 객체지향 프로그래밍 교과목에서 학생들이 과제 답안으로 제출한 소스코드 쌍 중에 표절로 판정된 10개의 소스코드 쌍을 확인한 결과 데이터 구조의 변경이 없는 것으로 나타났다.

이 논문에서는 알고리즘뿐만 아니라 데이터 구조 모두 고려하는 소스코드 표절 검사 기법을 제안한다. 이 기법은 먼저 소스코드를 데이터 구조 영역과 알고리즘 영역으로 구분한 후 이 두 영역 간 유사도를 측정한다. 그다음 두 영역 간 유사도에 각각 가중치를 부여하여 최종 유사도를 산출한다.

이 논문의 구성은 다음과 같다. 우선 2장에서는 관련 연구로 지문법과 텍스트 기반 유사도 측정 기법, 토큰 기반 유사도 측정 기법, 트리 기반 유사도 측정 기법에 대해 간략하게 언급한다. 3장에서는 이 논문에서 제안하는 기법을 기반으로 설계한 표절 검사 시스템에 대해 설명한다. 그리고 4장에서는 제안하는 기법의 효용성을 평가하기 위한 실험을 설명하고, 5장에서 이 논문의 결론을 맺는다.

2. 관련 연구

지문법(fingerprints)은 가장 쉽게 구현할 수 있는 표절 검사 기법으로서, 소스코드 상에 나타난 특정 키워드 개수나 라인 수 등과 같은 문법적 코드 정보를 활용하여 소스코드 간 유사도를 측정하는 방법이다[5]. 수치 기반 유사도 측정 기법 혹은 메트릭 기반 유사도 측정 기법으로도 불리며, 구현이 쉽고 속도가 빠른 장점이 있다. 하지만 소스코드의 구조적 정보는 얻지 못하기 때문에 유사도의 정확률이 떨어진다[10, 11, 12, 13].

텍스트 기반 유사도 측정 기법은 소스코드를 문자열의 나열로 변환한 다음, 변환된 문자열 비교를 통해 유사도를 측정한다. 일반적으로 문자열로 변환 시 소스코드 상에 있는 주석 및 공백을 제거하고, 키워드나 식별자는 특정 문자로 바꾸는 과정을 거친다. 이러한 과정을 거친 다음에 문자열 비교 알고리즘을 이용해 유사도를 측정한다. 이 기법은 소스코드를 정규화된 문자열로 변환하기 때문에 언어에 독립적이라는 장점이 있다[5]. 하지만 지문법과 마찬가지로 소스코드의 구조적 정보를 얻지 못해 정확률이 떨어지는 단점이 있다[14, 15].

토큰 기반 유사도 측정 기법은 어휘 분석기나 구문 분석기를 이용하여 소스코드로부터 토큰 서열을 생성하고, 이 토큰 서열 비교를 통해 유사도를 측정한다[5, 6, 7, 8, 16]. 주석 및 공백, 자바의 import 문 등 유사도 비교에 필요 없는 요소에 대해서는 토큰을 생성하지 않는다. 생성된 토큰 서열은 문자열 매칭 알고리즘을 사용하여 유사도를 측정한다. 공백 삽입이나 식별자 변경, 코드 형식(code formatting) 변경에 강하다[2]. 하지만 이 기법 역시 소스코드의 구조적 정보를 얻지 못하는 단점이 존재한다[3, 5, 17, 18].

트리 기반 유사도 측정 기법은 구문 분석기를 사용해 소스코드로부터 파스트리(parse tree)나 추상 구문 트리(AST: abstract syntax tree)와 같은 트리를 생성한 다음 생성된 트리 비교를 통해 유사도를 측정한다[5]. 트리 비교 알고리즘은 부분 트리 일치 알고리즘(subtree matching algorithm)을 많이 사용하고 있다[19, 20]. 텍스트 기반이나 토큰 기반 유사도 측정 기법과 달리 구조적 정보를 알 수 있어 함수의 위치 변경이나 병합, 소스코드 수정에 다른 기법보다 견고한 장점이 있다. 하지만 문장 재정렬에 취약한 단점이 있다[1, 2, 21, 22].

3. 데이터 구조를 고려한 소스코드 표절 검사 시스템

3.1 시스템 구조

데이터 구조를 고려한 표절 검사 시스템은 데이터 구조 유사도 측정, 프로그램 정적 추적, 유사도 가중치 계산, 최종 유사도 산출이라는 4개의 주요 모듈로 구성된다. 데이터 구조를 고려한 소스코드 표절 검사 시스템의 개략적인 구조는 Fig. 1과 같다.

데이터 구조 유사도 측정 모듈은 소스코드 쌍을 입력으로 받아 소스코드 쌍의 데이터 구조 간 유사도를 측정한다. 그리고 프로그램 정적 추적 모듈 역시 소스코드 쌍을 입력으로 받아 소스코드 쌍의 알고리즘 영역 간 유사도를 측정한다. 그

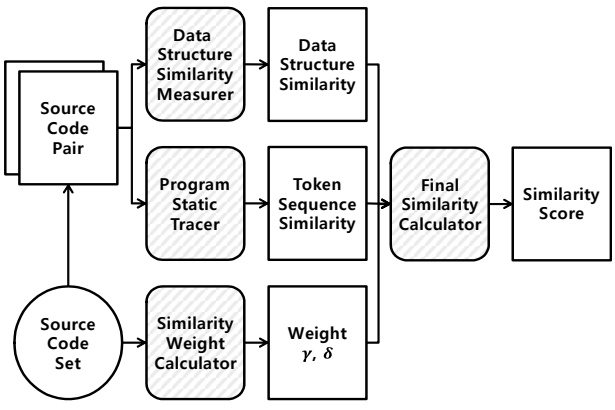

Fig. 1. The Architecture of the proposed plagiarism detection system considering data structures

리고 유사도 가중치 계산 모듈은 소스코드 집합을 입력으로 받아 앞서 구한 소스코드 쌍의 데이터 구조 유사도와 알고리즘 유사도에 부여할 가중치 γ, δ를 계산한다. 마지막으로 최종 유사도 산출 모듈에서는 데이터 구조 유사도와 알고리즘 유사도에 가중치 γ, δ를 부여하여 최종 유사도를 산출한다.

3.2 알고리즘 영역과 데이터 구조 영역 구분

데이터 구조를 고려한 소스코드 표절 검사 시스템에서는 소스코드 간 알고리즘 유사도와 데이터 구조 유사도를 측정하기 위해 소스코드를 알고리즘 영역과 데이터 구조 영역으로 구분한다. Fig. 2는 자바로 작성한 Point 클래스에서 알고리즘 영역과 데이터 구조 영역을 보여준다.

Fig. 2와 같이 소스코드를 알고리즘 영역과 데이터 구조 영역으로 구분한 다음에 각 영역 간 유사도를 구하고, 각각의 유사도에 가중치를 부여하여 최종 유사도를 산출한다.

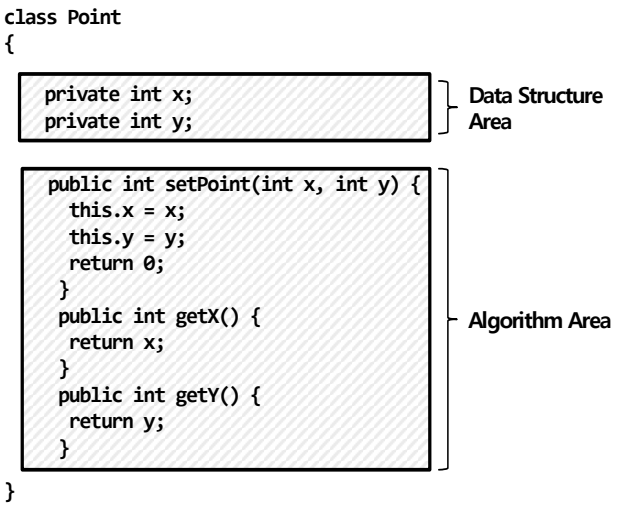

Fig. 2. The algorithm area and The data structure area of Point class written in Java

3.3 데이터 구조 유사도 측정 모듈

데이터 구조 유사도 측정 모듈은 소스코드 쌍의 데이터 구조 영역 간 유사도를 측정한다. 이를 위해 먼저 각각의 소스코드 상에 존재하는 데이터 구조를 선형화한다. 그다음 지역 정렬(local alignment)[23]을 이용하여 선형화된 모든 데이터 구조 간에 유사도 점수를 구한다. 그러고 나서 헝가리안 메소드(Hungarian Method)[24]를 이용해 최적 배정한 점수를 구한다. 마지막으로 데이터 구조 유사도 정규화를 수행한다. Fig. 3은 데이터 구조 유사도 측정 과정을 나타낸 것이다.

1) 데이터 구조 선형화

데이터 구조 선형화는 구조체나 클래스를 구성하고 있는 멤버 변수의 데이터 타입을 정해진 규칙에 따라 나열하여 데이터 타입 시퀀스로 만드는 것이다. 여기서는 데이터 타

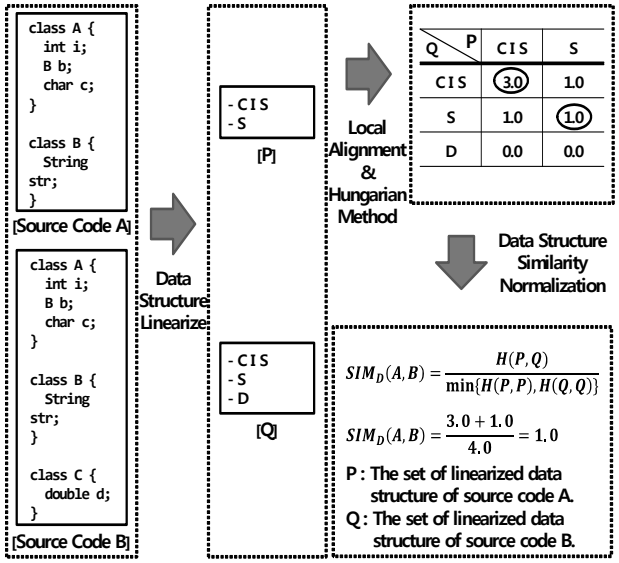

Fig. 3. The measurement process for the similarity of two data structures

입 시퀀스를 생성하기 위해 데이터 타입에 순서를 부여한 뒤 정렬하는 작업, 그리고 데이터 타입에 해당하는 토큰으로 변경하는 작업을 거친다.

데이터 구조를 선형화하는 가장 간단한 방법은 위에서부터 순차적으로 데이터 타입을 읽어 나열하는 것이다. 하지만 이러한 방식의 데이터 타입 시퀀스 생성은 두 가지 문제점이 있다. 첫 번째는 데이터 구조는 동일하지만 작성 형태에 따라 데이터 타입 시퀀스가 달라진다는 것이고, 두 번째는 자바에서 제공하는 기본 타입과, 같은 의미의 래퍼 클래스(wrapper class)를 다른 데이터 타입 시퀀스로 간주할 수 있다는 것이다.

이 논문에서는 두 가지 문제점을 해결하기 위해 데이터 타입에 해당하는 순서와 토큰을 정의했다. 첫 번째 문제는 데이터 타입에 순서를 부여한 다음 데이터 타입을 정렬해서 해결했고, 두 번째 문제는 기본 타입과 래퍼 클래스를 하나의 토큰을 생성하도록 함으로써 문제를 해결했다. Table 1은 데이터 타입에 해당하는 순서와 토큰을 정의한 표이다.

Table 1에서 순서는 데이터 타입을 정렬하기 위해 부여한 것으로 같은 의미를 가지는 기본 타입과 래퍼 클래스는 동일한 순서를 가진다. 토큰은 같은 의미를 가지는 기본 타입과 래퍼 클래스를 하나의 토큰으로 표현하기 위해 정의한 것이다.

배열 타입일 경우 배열이 일차원이면 기본 타입의 순서를 두 번 쓰고, 이차원일 경우 기본 타입을 세 번 쓴다. 예를 들어, “int[][]” 타입의 경우에 이 타입의 순서는 444가 된다. 이러한 방식을 사용하면 이차원 배열까지 의도한 대로 순서를 부여할 수 있다. 그리고 동일한 의미를 가지는 기본 타입과 래퍼 클래스를 하나의 토큰으로 나타냈다. 배열은 문자 ‘A’로 나타냈다.

Table 1. Orders and tokens of data types to linearize data structures

Java Data Type	Order	Token
boolean, Boolean	1	B
char, Character	2	C
byte, Byte	3	H
short, Short	3	H
int, Integer, BigInteger	4	I
long Long	5	L
float, Float	6	F
double, Double	7	D
String	8	S
[] (1-dimensional array)	-	A
[][] (2-dimensional array)	-	AA
Java API	999	P
User Defined Class	1000	-

토큰 서열에 사용자 정의 클래스가 있으면 해당 데이터 구조의 토큰 서열로 바꿔준다. 단, 재귀는 1회에 한해 해당 데이터 구조의 토큰 서열을 생성하고, 이후 토큰 R을 생성한다.

데이터 구조는 동일하지만 클래스 작성 형태가 다른 경우의 예는 Fig. 4와 같다. 이러한 경우 데이터 타입에 순서를 부여하고 정렬하면 Fig. 4(a)와 Fig. 4(b)는 동일한 데이터 타입 시퀀스 “char int double int[]”를 생성할 수 있다. 그다음 데이터 타입을 해당 토큰으로 바꿔주면 토큰 서열 “C I D IA”를 얻는다.

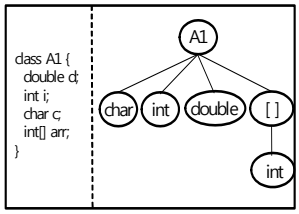

Fig. 4(a) class A1

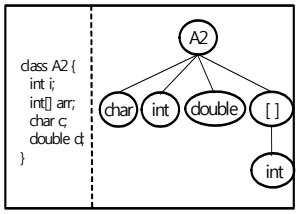

Fig. 4(b) class A2

Fig. 4. Examples sorted data structures of which are essentially same

2) 지역 정렬 및 헝가리안 메소드

선형화된 데이터 구조 간 유사도를 측정하기 위해 지역 정렬과 헝가리안 메소드를 사용했다. 지역 정렬은 Smith와 Waterman이 유전자 서열 간 부분적으로 가장 길게 일치하는 유사 영역을 찾기 위해 제안한 알고리즘이다. 헝가리안 메소드는 Harold Kuhn이 비용 행렬(cost matrix)에서 최적 배정(optimal assignment)을 다항 시간에 찾기 위해 제안한 알고리즘이다.

지역 정렬[23]을 사용하는 이유는 선형화된 데이터 구조 간 유사도 점수를 구하기 위해서다. 정렬(assignment)은 생물정보학(bioinformatics)에서 유전자 서열 간 유사 영역을 찾기 위해 제안되었으나 선형화된 데이터이기만 하면 어떤 데이터에 대해서도 적용할 수 있기 때문에 표절 검사에 자주 사용된다[4, 5, 19].

헝가리안 메소드[24]를 사용하는 이유는 소스코드 상에 존재하는 구조가 다른 여러 데이터 구조의 시퀀스를 하나의 시퀀스로 만들기가 어렵기 때문이다. 그래서 헝가리안 메소드를 이용해 데이터 구조 간 최적 배정을 한 다음 최적 배정된 데이터 구조 간 점수를 합하는 방식으로 데이터 구조 유사도 점수를 구한다. 본래 헝가리안 메소드는 최적 배정하여 비용이 최소인 경우를 찾으나 이 논문에서는 비용이 최대인 경우를 찾기 위해 사용한다.

3) 데이터 구조 유사도 정규화

데이터 구조 선형화, 지역 정렬, 헝가리안 메소드 단계를 거쳐 소스코드 간 데이터 구조 유사도 점수를 계산한 다음에는 마지막으로 유사도 점수를 정규화한다. 데이터 구조 유사도 점수를 정규화하는 수식은 식 1과 같다.

$$SIM_D(A,B) = \frac{H(P,Q)}{min\{H(P,P), H(Q,Q)\}} \quad (1)$$

식 1에서 SIM_D의 인자 A와 B는 소스코드이고, P는 소스코드 A의 선형화된 데이터 구조를 원소로 갖는 집합, Q는 소스코드 B의 선형화된 데이터 구조를 원소로 갖는 집합이다. H 함수는 헝가리안 메소드 기법을 구현한 함수로서 최적 배정을 통해 선택된 원소의 합을 돌려준다. 그러므로 H 함수를 통하여 얻은 값은 헝가리안 메소드의 특성상 두 프로그램 간의 가장 지역 정렬 점수가 높은 쌍을 반환하기 때문에 두 프로그램 간 가장 유사한 데이터 구조끼리 쌍을 이루고 있다. 그리고 이를 정규화하기 위해 P와 Q중 자기 자신의 최적 배정 값이 작은 집합을 이용하였다[8].

3.4 프로그램 정적 추적 모듈

프로그램 정적 추적은 지정훈이 제안한 토큰 서열 생성 방법으로[5, 6, 7] 함수의 호출 관계를 고려한 토큰 서열을 생성한다.

1) 분기 구문 정규화

프로그램 정적 추적 전에 하는 분기 구문 정규화 과정은 소스코드 표절 공격 중 분기 구문 변경에 대응하기 위한 방법이다. 예를 들어, if (A) B; else C; 구문은 동일한 의미의 if (￢A) C; else B; 구문으로 변경이 가능하다[5]. 이 논문에서는 분기 구문을 정규화하는 기준으로 연산자 토큰에 순서를 부여하였으며, 그 결과 토큰 순서에 따라 분기 구문을

정규화했다. Fig. 5는 토큰에 순서를 부여한 다음 정렬한 분기 구문 예를 보여준다.

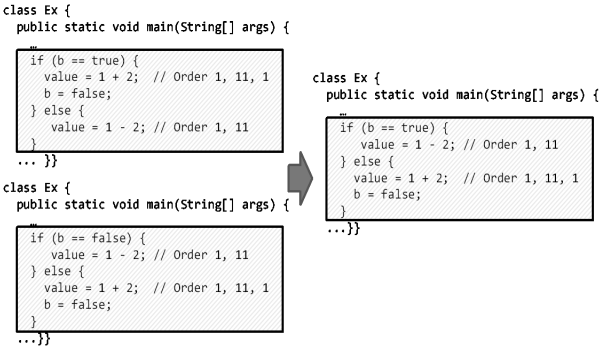

Fig. 5. An example of normalizing branch structures

2) 프로그램 정적 추적

프로그램 정적 추적은 main 함수를 시작으로 return 문을 만날 때까지 차례로 토큰을 생성한다. 토큰 생성 도중에 시스템 호출이나 라이브러리 함수 호출문을 만나면 토큰 "UC"를 생성한다. 프로그램 정적 추적 모듈과 데이터 구조 유사도 측정 모듈은 서로 독립적으로 유사도를 측정하기 때문에 토큰 "UC"에서 'C'는 데이터 구조 유사도 측정 모듈에서 사용하는 토큰과 전혀 관계가 없다. 사용자 정의 함수를 만나면 호출된 함수 정의 부분에 있는 토큰을 차례로 생성하고, return 문을 만나면 함수 호출이 되었던 자리로 복귀한다. 순환 함수일 경우에는 1회에 한해서만 토큰을 생성한다.

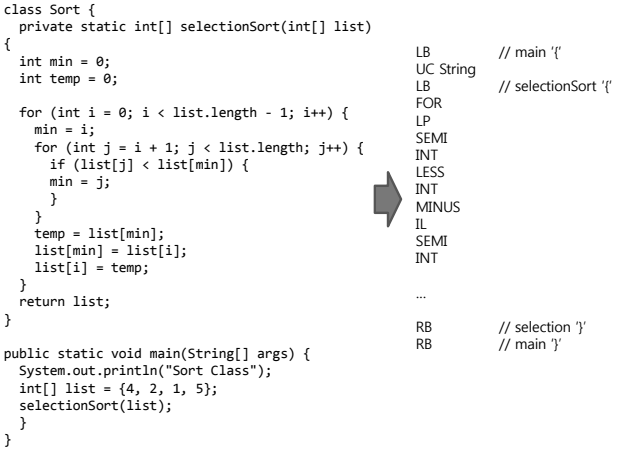

Fig. 6. Generating the token sequence using static tracing of programs

Fig. 6은 토큰 서열 생성 규칙에 따라 선택 정렬 소스가 토큰 서열로 변환되는 것을 보여준다. 먼저 메인 함수의 토큰을 생성하다 사용자가 정의한 selectionsort(list) 함수를 만나 호출된 함수 정의 부분에 있는 토큰을 차례로 생성하고 있다.

3) 토큰 서열 유사도 정규화

프로그램 정적 추적을 사용해 생성한 토큰 서열 간 유사도 점수를 정규화하는 수식은 식 2와 같다.

$$SIM_C(A,B) = \frac{L(R,S)}{min\{L(R,R), L(S,S)\}} \quad (2)$$

식 2에서 SIM_C의 인자 A와 B는 소스코드이고, R은 소스코드 A의 토큰 서열, S는 소스코드 B의 토큰 서열을 의미한다. L 함수는 지역 정렬 점수를 돌려준다.

3.5 유사도 가중치 계산 모듈

소스코드 쌍의 데이터 구조 영역과 알고리즘 영역 간 유사도를 측정한 다음에는 두 유사도에 부여할 가중치 γ, δ를 계산한다. 가중치 γ, δ는 표절 검사하는 소스코드 집합에서 데이터 구조 영역과 알고리즘 영역이 평균적으로 차지하는 비율로 계산했다. 가중치 γ를 구하는 수식은 식 3과 같고, δ를 구하는 수식은 식 4와 같다.

$$\gamma = \frac{\sum_{k=1}^{n}|C_k|}{\sum_{k=1}^{n}|D_k| + \sum_{k=1}^{n}|C_k|} \quad (3), \quad \delta = \frac{\sum_{k=1}^{n}|D_k|}{\sum_{k=1}^{n}|D_k| + \sum_{k=1}^{n}|C_k|} \quad (4)$$

위 식에서 $|D_k|$는 k번째 소스코드에서 데이터 구조 영역에 있는 선언(declaration)의 수를 의미하고, $|C_k|$는 k번째 소스코드에서 알고리즘 영역에 있는 문장(statement)의 수를 의미한다.

Table 2. Experimental results. The precision and the F-measure have been promoted with the help of comparing data structures

Evaluation	HW6		HW7		HW8		HW9	
	CS	DS+CS	CS	DS+CS	CS	DS+CS	CS	DS+CS
Precision	10%	20%	100%	100%	50%	100%	25%	33%
Recall	100%	100%	100%	100%	100%	100%	100%	100%
F-measure	18%	33%	100%	100%	67%	100%	40%	50%

4. 실험 및 평가

데이터 구조를 고려한 소스코드 표절 검사 기법의 효용성을 평가하기 위해 3장의 설계한 시스템을 구현하고 시스템 성능을 평가하였다. 이 시스템은 Windows 7(Professional SP1)에서 오픈 소프트웨어 Eclipse와 javaparser(버전-1.0.8)[25]를 사용하여 구현하였다. javaparser는 구글 코드에 공개되어 있는 자바 파서로서 자바 1.5를 지원하고, AST를 생성한다. 방문자 패턴(visitor pattern)을 지원하기 때문에 다루기가 쉽다.

테스트 집합으로는 부산대학교 2012년 2학기 객체지향 프로그래밍 교과목에서 학생들이 과제 답안으로 제출한 소스코드 집합 중 4개를 선택했다. 소스코드 개수는 총 126개이고, 평균 라인 수는 101개, 평균 가중치는 γ가 89%, δ가 11%였다. 시스템 평가 지표로는 재현율(recall)과 정확률(precision), F-measure를 이용했다. F-measure는 재현율과 정확률을 같은 중요도로 결합한 것이다.

Table 2는 시스템 평가 결과를 정리한 것이다. Table 2에서 CS는 알고리즘 유사도를 의미하며 기존의 프로그램 유사도 검출 방법에 이용하는 방법 중 하나이다[8]. 그리고 DS+CS는 알고리즘 유사도와 데이터 구조 유사도를 종합한 유사도다. 시스템 평가 결과를 보면 재현율은 데이터 구조와 상관없이 항상 100%로 나타난다. 하지만 정확률과 F-measure는 알고리즘만 고려한 경우보다 데이터 구조를 함께 고려한 경우의 유사도가 각각 22.6%, 19.3% 향상된 결과를 얻을 수 있었다. 이는 데이터 구조 유사도가 알고리즘 유사도만 사용했을 경우에 발생하는 거짓 경보 문턱 값(threshold value)을 낮춰 시스템의 정확률을 향상시키고 있음을 의미한다.

5. 결 론

이 논문에서는 데이터 구조를 고려한 소스코드 표절 검사 기법을 제안했다. 이 기법은 먼저 소스코드를 데이터 구조 영역과 알고리즘 영역으로 구분해 각각의 영역에 대한 유사도를 구하고, 그다음 각각의 유사도에 가중치를 부여하여 최종 유사도를 산출한다.

제안하는 표절 검사 기법의 효용성을 평가하고자 객체지향 프로그래밍 교과목의 과제로 제출된 Java 소스코드 집합을 테스트 집합으로 하여 실험을 수행하였다. 실험 결과 재현율은 데이터 구조를 고려한 경우나 고려하지 않은 경우 모두 100%로 나타났다. 하지만 정확률과 F-measure는 데이터 구조를 고려한 경우가 고려하지 않은 경우보다 각각 22.6%, 19.3% 향상된 결과를 얻었다. 정확률이 검출된 소스코드 쌍 중 실제 표절 소스코드 쌍의 수임을 고려할 때 이와 같은 결과는 데이터 구조를 고려함으로써 표절 검출의 거짓 경보를 줄일 수 있음을 의미한다. 구체적으로 말해서 거짓 경보의 수를 줄일 수 있는 더 정확한 문턱 값(threshold value)을 제시할 수 있음을 의미한다.

향후 연구로는 알고리즘 유사도와 데이터 구조 유사도에 부여하는 가중치 γ, δ를 계산하는 방법을 개선하는 연구가 필요하다. 이 논문에서는 가중치 γ, δ를 소스코드 집합에서 데이터 구조 영역과 알고리즘 영역이 평균적으로 차지하는 비율로 계산한다. 하지만 이와 같은 방법은 지역 변수만을 사용하여 작성한 소스코드의 경우를 고려하고 있지 않기 때문에 이 부분을 개선한다면 좀 더 나은 정확률을 보일 것으로 기대된다.

Reference

[1] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo, “Comparison and evaluation of clone detection tools,” *IEEE Transactions on Software Engineering*, Vol.33, No.9, pp.577-591, 2007.

[2] Chanchal K. Roy, James R. Cordy, and Rainer Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” *Science of Computer Programming*, Vol.74, No.7, pp.470-495, 2009.

[3] Lutz Prechelt, Guido Malpohl, and Michael Philippsen, “Finding plagiarisms among a set of program with JPlag,” *Journal of Universal Computer Science*, Vol.8, No.11, pp.1016-1038, 2002.

[4] Michael J. Wise, “Neweyes: a system for comparing biological sequences using the running karp-rabin greedy string-tiling algorithm,” In *Intelligent Systems in Molecular Biology*, pp.393-401, 1995.

[5] Jeong-Hoon Ji, *Program similarity analysis framework using adaptive sequence alignment technique*, PhD thesis, Pusan National University, 2010.

[6] Jeong-Hoon Ji, Gyun Woo, Sang-Hyun Park, and Hwan-Gue Cho, “An intelligent system for detecting source code plagiarism using a probabilistic graph model,” In *Machine Learning and Data Mining in Pattern Recognitions Posters*, pp.55-69, 2007.

[7] Yun-Jung Lee, Jin-Su Lim, Jeong-Hoon Ji, Hwan-Gue Cho, and Gyun Woo, “Plagiarism detection among source codes using adaptive methods,” *Transactions on Internet and Information Systems*, Vol.6, No.6, pp.1627-1648, 2012.

[8] Jeong-Hoon Ji, Gyun Woo and Hwan-Gue Cho, "A source code linearization technique for detecting plagiarized programs," In Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, pp.73-77, 2007.

[9] Niklaus Wirth, *Algorithms + Data Structures = Programs*, Prentice Hall, 1976.

[10] Karl J. Ottenstein, "An algorithmic approach to the detection and prevention of plagiarism," *ACM SIGCSE Bulletin*, Vol.8, No.4, pp.30-44, 1976.

[11] Maurice H. Halstead, "*Elements of Software Science (Operating and programming systems series)*," Elsevier Science Inc., 1977.

[12] Hal L. Berghel and David L. Sallach, "Measurements of program similarity in identical task environments," *ACM SIGPLAN Notices*, Vol.19, No.8, pp.65-76, 1984.

[13] Sam Grier, "A tool that detects plagiarism in pascal programs," In *ACM SIGCSE Bulletin*, Vol.13, pp.15-20, 1981.

[14] Stéphane Ducasse, Oscar Nierstrasz, and Matthias Rieger, "On the effectiveness of clone detection by string matching," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol.18, No.1, pp.37-58, 2006.

[15] J. Howard Johnson, "Identifying redundancy in source code using fingerprints," In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering-Volume 1*, pp.171-183, IBM Press, 1993.

[16] Jin-Su Lim, *A code plagiarism detection system considering the coding style*, Master thesis, Pusan National University, 2012.

[17] David gitchell and Nicholas Tran, "Sim: a utility for detecting similarity in computer programs," In *ACM SIGCSE Bulletin*, Vol.31, pp.266-270, ACM, 1999.

[18] Michael J. Wise, "Yap3: Improved detection of similarities in computer program and other texts," In *ACM SIGCSE Bulletin*, Vol.28, pp.130-134, ACM, 1996.

[19] Michel Chilowicz, Etienne Duris, and Gilles Roussel, "Syntax tree fingerprinting for source code similarity detection," In *17th IEEE International Conference on Program Compre hension*, pp.243-247, IEEE, 2009.

[20] Raimar Falke, Pierre Frenzel, and Rainer Koschke, "Empirical evaluation of clone detection using syntax suffix trees," Empirical Software Engineering, Vol.13, No.6, pp.601-643, 2008.

[21] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," In *Proceedings of the 29th international conference on Software Engineering*, pp.96-105, IEEE, 2007.

[22] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier, "Clone detection using abstract syntax trees," In *International Conference on Software Maintenance*, pp.368-377, IEEE, 1998.

[23] Temple F. Smith and Michael S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, Vol.147, No.1, pp.195-197, 1981.

[24] Harold W. Kuhn, "Variants of the hungarian method for assignment problems," *Noval Research Logistics Quarterly*, Vol.3, No.4, pp.253-258, 1956.

[25] jgesser, javaparser - Java 1.5 parser and AST [Internet], http://code.google.com/p/javaparser

이 기 화

e-mail : dlrlghkekk@gmail.com

2011년 경성대학교 컴퓨터학부(학사)

2014년 부산대학교 전자전기컴퓨터공학과 (석사)

2014년~현 재 슈어소프트테크 전임연구원

관심분야 : Program Analysis, Static Analysis, Program Plagiarism Detection

김 연 어

e-mail : yeoneo@pusan.ac.kr

2010년 동아대학교 컴퓨터학과(학사)

2012년 동아대학교 컴퓨터공학과(석사)

2012년~현 재 부산대학교 전자전기컴퓨터공학과 박사과정

관심분야 : Program Analysis, Static Analysis, Program Plagiarism Detection

우 균

e-mail : woogyun@pusan.ac.kr

1991년 한국과학기술원 전산학(학사)

1993년 한국과학기술원 전산학(석사)

2000년 한국과학기술원 전산학(박사)

2000년～2002년 동아대학교 컴퓨터공학과 전임강사

2002년～2004년 동아대학교 컴퓨터공학과 조교수

2005년～현 재 부산대학교 전자전기컴퓨터공학과 교수

2012년～현 재 LG전자 스마트제어센터 행정관리부장

관심분야 : Program Language, Compiler, Functional Language, Grid Computing, Software Metric, Program Visualization