http://dx.doi.org/10.5392/JKCA.2013.13.11.037

# 클래스 구조 그래프 비교를 통한 프로그램 표절 검사 방법

## A Method for Detecting Program Plagiarism Comparing Class Structure Graphs

김연어*, 이윤정**, 우균***
부산대학교 전자전기컴퓨터공학과*, 부산대학교 IT기반 융합산업창의인력양성 사업단**,
부산대학교 전자전기컴퓨터공학과/LG전자 스마트제어센터***

Yeoneo Kim(yeoneo@pusan.ac.kr)*, Yun-Jung Lee(leeyj01@pusan.ac.kr)**,
Gyun Woo(woogyun@pusan.ac.kr)***

**요약**

코드 이동성이 증가함에 따라 코드 도용이 문제가 되고 있으며 이를 대처하기 위해 프로그램 비교를 위한 연구가 많이 진행되고 있다. 이 논문은 클래스 구조를 이용하여 Java 프로그램의 표절을 검사하는 방법을 제안한다. 제안 방법은 멤버 변수와 메소드 간의 참조 관계를 나타내는 그래프를 생성한다. 변수 참조 관계는 이분 그래프 형태로 나타나는데 이렇게 생성된 그래프를 대상으로 그래프 동형 검사를 적용하여 프로그램 간의 유사도를 측정한다. 이 논문에서는 제안 방법의 효과를 입증하기 위해 2012년 부산대학교 객체지향 프로그래밍 과제로 제출된 Java 프로그램을 대상으로 실험하였다. 그리고 제안 방법의 정확도를 평가하기 위해 기존 유사도 검사 프로그램인 JPlag와 Stigmata를 대상으로 F-measure 지표를 이용해 비교하였다. 그 결과 제안 방법의 F-measure가 JPlag보다 0.17, Stigmata보다 0.34 높은 것으로 나타났다.

■ 중심어 : | 프로그램 표절 검사 | 정적 분석 | 프로그램 분석 | 소프트웨어 비교 | 클래스 구조 |

**Abstract**

Recently, lots of research results on program comparison have been reported since the code theft become frequent as the increase of code mobility. This paper proposes a plagiarism detection method using class structures. The proposed method constructs a graph representing the referential relationship between the member variables and the methods. This relationship is shown as a bipartite graph and the test for graph isomorphism is applied on the set of graphs to measure the similarity of the programs. In order to measure the effectiveness of this method, an experiment was conducted on the test set, the set of Java source codes submitted as solutions for the programming assignments in Object-Oriented Programming course of Pusan National University in 2012. In order to evaluate the accuracy of the proposed method, the F-measure is compared to those of JPlag and Stigmata. According to the experimental result, the F-measure of the proposed method is higher than those of JPlag and Stigmata by 0.17 and 0.34, respectively.

■ keyword : | Program Plagiarism Detection | Static Analysis | Program Analysis | Software Comparison | Class Structure |

* 본 논문은 한국콘텐츠학회 JCCC 2013 제1회 융합콘텐츠 제주학술대회 우수논문입니다
접수일자 : 2013년 08월 16일 심사완료일 : 2013년 10월 24일
수정일자 : 2013년 10월 24일 교신저자 : 우균, e-mail : woogyun@pusan.ac.kr

## I. 서 론

최근 스마트폰의 보급으로 소프트웨어 코드 이동성이 증가하고 있다. 코드 이동성이 증가함에 따라 프로그램 저작권 논란도 같이 늘어나고 있어 라이선스 관리 문제가 크게 다루어지고 있다[1]. 특히 안드로이드 앱은 Java 기반의 플랫폼이기 때문에 역분석을 통한 소스 코드 표절이 쉽게 일어날 수 있어 프로그램 표절 검사 기법이 더욱 필요하다.

프로그램 표절 검사는 예전부터 연구가 활발히 이루어지고 있다. 기존의 검사 방법은 소스 코드의 문서적 특징을 이용하는 방법부터 프로그램의 의미를 고려하는 방법까지 다양하게 제시되었다. 그중 이 논문에서는 프로그램 의미를 고려하는 소프트웨어 버스마크(birthmark)를 이용하여 프로그램 표절을 검사하는 방법을 이용한다.

소프트웨어 버스마크는 프로그램의 고유한 특징을 이용해 프로그램을 식별하는 방법이다[2]. 이는 최근 프로그램 표절 검사 연구에서 많이 이용되고 있는 방법으로 일종의 워터마크(water mark)와 유사한 방법이라 볼 수 있다. 워터마크는 주로 위조지폐 식별을 위해 지폐에 삽입하거나 신분증 불법 복제를 방지하기 위해 사용되는 방법으로 복제 탐지에 유용한 방법이다[3].

소프트웨어 버스마크와 워터마크의 결정적인 차이점은 마크를 삽입하는 방법이다. 워터마크는 프로그램의 복제 방지를 위해 개발자가 직접 특정한 기호 코드에 삽입하는 방법이다. 하지만 소프트웨어 버스마크는 프로그램에 추가로 마크를 삽입하지 않고 탐지하는 방법이다.

소프트웨어 버스마크는 이용하는 프로그램의 특징에 따라 동적인 방법과 정적인 방법 두 가지로 분류된다[2][4-12]. 동적인 방법은 프로그램의 실행 중 특징을 추출하는 방법으로 프로그램 입력에 따라 결과 값이 달라질 수 있다. 정적인 방법은 프로그램의 코드 정보를 이용해 특징을 추출하는 방법으로 모든 가능한 특징을 뽑아낼 수 있다.

이 논문에서는 정적인 방법인 Java 소스 코드를 이용해 클래스의 구조를 버스마크로 추출해 프로그램 표절을 검사하는 방법을 제안한다. 제안 방법은 클래스 멤버 변수와 메소드 간의 참조 관계를 이용해 이를 그래프로 표현하는 방법이다. 그리고 생성된 그래프를 그래프 동형 검사를 이용해 프로그램 간의 표절 관계를 검출하는 방법이다.

이 논문의 나머지 부분은 다음과 같이 구성된다. 우선 2장에서 기존의 표절 검사와 관련된 연구를 살펴본다. 그리고 3장에서 이 논문에서 해결해야 할 문제 및 제안하는 버스마크를 정의한다. 4장에서는 제안 방법으로 어떻게 표절을 검사하는지에 대해 알아본다. 그리고 5장에서 실험을 통해 제안 방법을 평가한다. 이후 6장에서 결론을 내는 것으로 마무리를 짓는다.

## II. 관련 연구

### 1. 전통적인 표절 검사 기법

기존 표절 검사 기법 중 버스마크 이전의 전통적인 표절 검사 기법은 주로 소스코드나 프로그래밍 언어의 구조를 이용해 표절을 검사한다. 이 방법은 모두 소스코드에 기반을 둔 검사 방법이기 때문에 소스코드가 반드시 필요하다. 이 분류에 속하는 표절 검사 기법은 크게 문자열 기반과 토큰 기반, AST(Abstract Syntax Tree) 기반, PDG(Program Dependency Graph) 기반으로 분류할 수 있다[13-16].

문자열 기반의 방법은 소스코드를 문자열로 취급하고 문서 유사도 검사 방법과 비슷하게 검사하는 방법이다[13]. 문자열 기반의 방법은 소스코드가 기존 문자열과 다른 점인 주석이나 공백 문자를 모두 제거한 문자열을 대상으로 검사하는 것이다. 그리고 해당 방법은 표절 검사를 위해 문자열 비교 알고리즘을 이용해 유사도 점수를 측정하고 이를 이용해 표절 여부를 알려준다. 하지만 이 방법은 제어 구조나 스트링 변경에 따라 유사도 점수가 달라지는 단점이 있다.

토큰 기반의 방법은 소스코드를 토큰으로 변경한 뒤 해당 토큰을 대상으로 유사도 검사를 하는 방법이다[14][15][20]. 해당 방법은 컴파일러의 어휘 분석기를 이용해 소스코드를 토큰으로 변경한다. 그리고 문자열 비

교 알고리즘이나 유전자 분석 기법에서 사용되는 정렬(alignment) 알고리즘을 이용해 유사도 점수를 측정하고 이를 이용해 표절 여부를 알려준다. 하지만 이 방법은 소스 코드를 삽입하거나 순서를 변경하는 등 소스 코드 구조를 변경하면 유사도 점수가 달라지는 단점이 있다.

AST 기반의 방법은 소스코드를 AST로 변경하여 표절 검사를 하는 방법이다[13]. 해당 방법은 컴파일러의 구분 분석기를 이용하여 소스코드의 AST를 생성한다. 이 방법은 AST가 트리로 표현되기 때문에 트리를 비교하는 방법을 이용해 유사도 점수를 측정하고 이를 이용해 표절 여부를 알려준다. 하지만 이 방법은 역시 소스 코드 구조를 변경하면 유사도가 달라지는 단점이 있다.

PDG 기반의 방법은 소스 코드를 PDG로 변경하여 표절 검사를 하는 방법이다[16]. PDG는 전체 프로그램의 제어 흐름과 데이터 간의 의존성 정보를 표현한 그래프로 소스코드 구문분석을 통해 생성한다. 이 방법은 PDG가 그래프로 표현되기 때문에 그래프 동형(graph isomorphism) 검사를 이용해 유사도 점수를 측정하고 이를 이용해 표절 여부를 알려준다. 하지만 이 방법은 루프 펼치기와 같은 제어 흐름을 변경하는 방법에 유사도 점수가 달라질 수 있다.

## 2. 버스마크를 이용한 표절 검사 기법

기존 표절 검사 기법 중 버스마크를 사용하는 표절 검사 기법은 크게 정적인 방법과 동적인 방법으로 나뉜다. 정적인 방법은 프로그램의 소스코드나 바이트 코드를 이용하는 것이다. 그리고 동적인 방법은 프로그램 실행 시 얻을 수 있는 정보를 이용하는 것이다.

정적인 방법은 프로그램을 실행하지 않고 버스마크를 얻어내는 방법이다. 이 방법은 크게 소스코드를 이용하는 방법과 바이트 코드나 바이너리 코드와 같은 실행 파일을 이용하는 방법으로 나눌 수 있다. 소스 코드를 이용하는 방법은 Tamada가 제안한 방법으로 4가지 종류의 정적인 버스마크를 이용하는 방법이다[2]. 첫 번째는 멤버 변수의 상수값을 이용하는 CVFV(constant values in field variables)이며, 두 번째는 메소드 호출 순서를 이용한 SMC(sequence of method call)이다. 그리고 세 번째는 상속 구조를 이용한 버스마크인 IS(inheritance structure)와 클래스 사용 관계를 표현한 UC(used classes)가 있다.

실행 파일을 이용하는 방법으로 바이트 코드를 이용하는 방법으로 대표적으로 Myles가 제안한 k-gram을 기반으로 한 방법이 있다[4]. 이 방법은 바이트 코드를 k개의 명령어 나열로 변경한 집합을 이용하는 방법이다. 그리고 이 외에도 바이너리 코드를 이용한 방법으로 Choi가 제안한 윈도우 환경의 바이너리 코드에서 API 호출 구조를 버스마크로 이용하는 방법이 제안되었다[5]. 이외에도 윈도우 바이너리 실행 환경 정보인 PE(Preinstallation Environment) 내부의 문자열이나 IAT(Import Adresss Table)을 이용하는 방법도 제안되었다[10][11].

동적인 방법은 프로그램 실행 시 얻어지는 정보를 이용해 버스마크를 얻어내는 방법이다. 이 방법은 대표적으로 전체 실행 제어 흐름(WPP : whole program path)을 고려하는 방법과 API 호출 정보를 이용하는 방법, 힙 메모리를 이용하는 방법이 있다. WPP는 Myles가 제안한 방법으로 프로그램 실행 도중 얻어지는 프로그램 제어 흐름을 버스마크로 이용하는 방법이다[6].

API 호출 구조를 이용하는 방법으로 Tamada에 의해 제안된 방법으로, 윈도우 API 함수 호출 순서인 EXESEQ와 API 함수 호출 빈도인 EXEFREQ 버스마크를 이용한 방법이 있다[7][8]. 그리고 이 외에도 Wang에 의해 제안된 API 호출 중 시스템 함수 호출만을 고려한 방법이 있다[9].

마지막으로 힙 메모리 구조를 이용하는 방법은 Chan에 의해 제안된 방법이다[12]. 이 방법은 JVM의 메모리를 분석하는 방법이다. 이 방법은 힙 메모리에서 객체 간의 참조 관계만을 추출해 버스마크로 이용하는 방법이다.

## III. 문제 정의

이 논문에서는 프로그램 표절 검사를 위해 클래스 구조를 이용하는 방법을 제안한다. 제안 방법은 소프트웨

어 버스마크의 한 종류이다. 소프트웨어 버스마크는 프로그램의 고유한 특징을 이용해 프로그램을 식별하는 방법으로 Myles에 따르면 정의 1과 같다[6].

**정의 1.** (소프트웨어 버스마크) 프로그램 $p$, $q$가 존재하며, 프로그램으로부터 고유한 특징을 추출하는 함수 $f$가 존재한다. 그러면 아래의 조건을 만족하는 $f(p)$가 프로그램 $p$의 버스마크이다.

- $f(p)$는 유일하게 $p$에서 얻을 수 있다.
- $q$가 $p$의 복제 프로그램이라면 $f(p) = f(q)$이다.

정의 1과 같이 소프트웨어 버스마크의 첫 번째 조건은 프로그램별로 유일한 특징을 추출할 수 있어야 함을 의미한다. 첫 번째 조건이 만족된다면 유사한 프로그램을 서로 다른 사람이 작성하였을 때 서로 다른 버스마크가 생성된다. 그리고 두 번째 조건은 복제된 프로그램은 항상 원본 프로그램과 같은 특징이 추출되어야 함을 의미한다. 두 번째 조건이 만족된다면 프로그램의 복제 여부를 버스마크를 통해 검출할 수 있다.

제안 방법은 기존 버스마크가 알고리즘과 같은 행동에 초점을 맞추는 것과 달리 자료 구조의 형태를 이용하는 방법이다. 이를 위한 제안 방법의 버스마크는 클래스 구조 중 멤버 변수와 메소드 간의 연관관계를 이용해 그래프로 표현한다. 그리고 제안하는 버스마크는 클래스 연관관계 그래프(CRGB : Class Relation Graph Birthmark)라 하며 정의는 아래와 같다.

**정의 2.** (CRGB) 클래스 연관관계 그래프 $G=(V_v, V_m, E, \alpha)$는 무향 이분 그래프이며 아래 조건을 만족한다.

- $V_v$은 클래스 멤버 변수 집합으로 멤버 변수와 상속된 변수가 속한다. $V_m$는 클래스 메소드 중 멤버 변수를 참조하는 메소드의 집합이다. 또한 $V_v \cap V_m = \varnothing$ 을 만족한다.
- $E$는 $\langle v_{v1}, v_{m1} \rangle \subseteq E, v_{v1} \in V_v, v_{m1} \in V_m$을 만족한다.
- $\alpha : (V_v, V_m) \rightarrow E$는 함수로 $V_v$와 $V_m$을 입력으로 받아 $E$를 반환하는 함수이다.

CRGB는 정의 2와 같이 정점(node)은 멤버 변수의 집합과 메소드의 집합으로 나누어 정의된다. 그리고 간선(edge)은 멤버 변수의 집합과 메소드의 집합 사이에 존재하며 $\alpha$함수를 통해 생성된다. $\alpha$함수는 멤버 변수 집합과 메소드의 집합을 입력으로 받아 간선 집합을 반환하는 함수로 알고리즘 1과 같이 정의할 수 있다.

알고리즘 1은 멤버 변수와 메소드를 입력으로 받아 연관관계가 있는 정점 간에 간선을 추가해주는 함수이다. 이를 위해 각 메소드별로 방문을 하고 메소드에서 모든 식을 뽑아낸다. 그리고 모든 식에서 멤버 변수 사용이 있는지 확인한다. 그리고 멤버 변수 사용이 있는 식이 존재한다면 해당 식이 사용된 메소드와 멤버 변수 간에 연관 관계가 있는 것으로 간주하고 CRGB에서 간선을 추가한다. 그리고 [그림 1]과 같은 Java 소스 코드에 제안하는 방법을 적용하면 [그림 2]와 같은 CRGB를 얻을 수 있다.

**알고리즘 1. 간선 집합을 계산하기 위한 $\alpha$함수. 메소드에서 멤버 변수가 사용되는지 확인하여 간선집합을 생성한다.**

```
α(V : member variable list , M : method list)
{
    E = ∅;
    for method in M {
        for expr in method.ExprList() {
            var = UsedVar(V, expr);
            if (var != nil) {
                E = E ∪ edge(var, method);
            }
        }
    }
    return E;
}
```

```
Class Person {
  String name; int age; Job job; Hobby hobby;
  public void print() {
    print("이름:" + name + "나이:" + age );
  }
  public void work() {
      job.work();
  }
  public void play() {
      hobby.play();
  }
}
```

**그림 1. CRGB를 생성하기 위한 Java 예제 코드. 사람을 나타낸 클래스로 세 개의 메소드(print, work, play)로 구성되어 있다.**

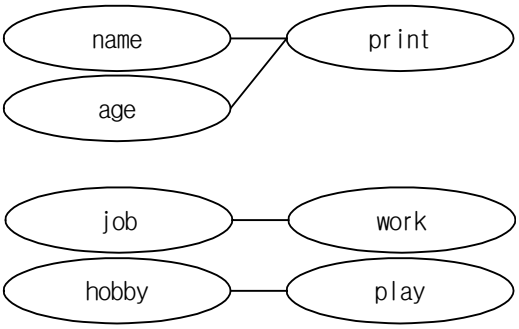


**그림 2. Person 클래스로 생성된 CRGB. print()는 name과 age 변수와 관련 있으며, work()는 job 변수와 play()는 hobby 변수와 연관 관계가 있다.**

[그림 1]의 Person 클래스는 사람을 표현한 클래스로 print, work, play 메소드로 구성되어 있다. print()에서는 name 변수와 age 변수를 참조하기 때문에 [그림 2]와 같이 〈name, print〉, 〈age, print〉 간선이 만들어진다. 그리고 work()에서는 job 변수를, play()에서는 hobby 변수를 참조하기 때문에 〈job, work〉, 〈hobby, play〉 간선이 만들어진다.

## IV. 시스템 디자인

이 장에서는 제안하는 프로그램 표절 검사를 위해 제안하는 방법이 어떤 구조로 이루어져 있는지 알아본다. 이 논문에서 제안하는 시스템은 [그림 3]과 같은 구조로 구성된다. 제안 시스템은 크게 CRGB 생성기 부분과 유사도 계산기 부분으로 나누어진다. CRGB 생성기는 소스 코드를 입력으로 받아 CRGB를 생성한다. 이 때 Java 구문분석기(parser)가 사용된다. 그리고 유사도 계산기에서는 CRGB를 입력으로 받아 유사도 점수를 계산해 결과 파일로 보여준다.

### 1. CRGB 생성기

CRGB 생성기는 Java 소스코드를 분석해 제안 버스마크를 생성하는 부분이다. 제안 방법에서는 Java 소스코드를 분석하기 위해 Java 파서인 javaparser-1.0.8[17]을 이용한다. javaparser는 Java 1.5 문법을 지원하는 파서로 Java로 개발된 라이브러리이다. 이 파서는 방문자 패턴을 이용해 구현되어 구문별 원하는 코드를 생성할 수 있게 되어 있다. CRGB 생성기는 입력된 소스코드에서 클래스를 기준으로 CRGB를 생성한다. 예를 들어 하나의 Java 소스코드에 여러 클래스가 포함된 소스코드는 클래스별 CRGB가 생성된다.

CRGB 생성기에서는 CRGB의 정확성을 높이기 위해 두 가지 방법을 이용한다. 첫 번째는 멤버 변수가 같은 메소드에 인자로 넘어갈 때에도 연관 관계가 있다고 간주한다. 이는 메소드 인자에 멤버 변수가 사용되고 있다면 해당 멤버 변수 값이 메소드에 전달되어 사용되기 때문에 서로 연관 관계에 있다고 볼 수 있다. 두 번째는 CRGB 생성 시 각 정점의 값을 변수의 타입이나 메소드의 반환 타입으로 사용한다. 이는 변수의 이름이나 메소드의 이름은 쉽게 변할 수 있지만, 타입 정보는 쉽게 변하지 않는 정보라는 점을 이용한 것으로서 이를 이용하여 CRGB의 정확성을 높일 수 있다. [표 1]은 제안 방법에서 각 정점에 부여하는 타입 정보와 적용 사례이다.

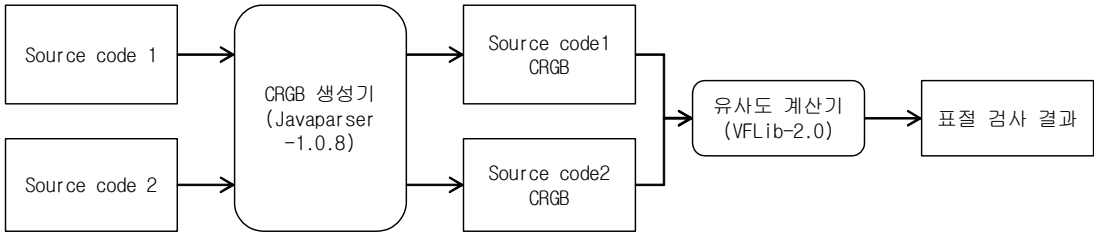


**그림 3. 제안하는 표절 검사 시스템 전체 구조. 제안 시스템은 소스 코드를 입력으로 받아 CRGB를 생성하고 생성된 CRGB를 유사도 계산기로 넘겨 표절 검사를 실행한다.**

**표 1. CRGB 정점에 부여되는 값. CRGB 정점에는 멤버 변수의 타입, 메소드 반환 타입이 값으로 부여된다.**

| 분류 | 포함 종류 | 정점 부여 값 |
|---|---|---|
| 기본 자료형 | Java 원시 타입 | Primitive |
| 사용자 정의 자료형 | 사용자 정의 클래스 | Ref |
| 일반 참조 자료형 | JDK 제공 클래스, 외부 라이브러리 | 실제 클래스 이름 |
| void 자료형 | Void 타입 | Void |

[표 1]과 같이 기존 자료형은 int나 double과 같이 Java 언어에서 기본적으로 제공되는 원시 타입이 포함되게 되며 정점의 값은 Primitive로 생성된다. 사용자 정의 자료형은 소스코드 안에서 사용자가 직접 정의한 클래스가 사용되는 경우가 이곳에 포함되며 정점 값은 Ref로 정점이 생성된다. 그리고 일반 참조 자료형은 사용자 정의 클래스를 제외한 모든 클래스가 이 분류에 속하며, 정점 값은 실제 클래스 이름이 부여된 정점이 생성된다. 마지막으로 void형은 메소드의 반환 타입이 void로 설정된 함수가 여기에 속하며, 정점 값은 Void로 정점이 생성된다.

또한, 제안 방법에서는 메소드 정점에 대하여 지문법(fingerprints)을 이용해 유사도 검사의 정확도를 높인다. 지문법은 소스코드에 포함된 문법적 코드 정보를 이용하는 방법이다. CRGB만 이용하는 경우 클래스 구조만 고려하고, 메소드의 행동(behavior)은 고려하지 못하기 때문에 지문법을 이용해 메소드의 행동을 표현한다, 이를 위해 제안 방법은 메소드가 가지고 있는 문장의 종류를 카운트해 지문법 정보로 이용한다.

## 2. 유사도 계산기

유사도 계산기는 클래스별 CRGB를 입력으로 받아 CRGB간의 유사도를 계산하는 부분이다. CRGB는 그래프로 표현되는 버스마크이기 때문에 그래프 비교를 통해 유사도 점수 계산이 필요하다. 그러므로 제안 방법에서는 그래프 동형 검사를 이용해 유사도를 계산한다.

그래프 동형 검색은 NP-완전으로 알려진 문제로서 이미 많은 연구 결과가 발표된 바 있다. 이 논문에서는 그래프 동형 문제를 해결하기 위해 VFLib2.0[18]라는 그래프 라이브러리를 이용하였다. VFLib는 C++로 구현된 그래프 동형 비교 라이브러리로 VF, VF2, Ullmann, Schmidt-Druffel과 같은 그래프 비교 알고리즘을 제공하고 있다. 제안 방법에서는 그중 다른 알고리즘에 비해 성능이 좋다고 알려진 VF2 알고리즘을 이용하였다[19]. VF2 백트래킹(backtraking), 미리보기(look-ahead) 기반의 알고리즘으로 최선의 경우 $\Theta(n^2)$의 시간이 걸리며 최악에는$\Theta(MN)$ 시간이 걸리는 알고리즘이다[19].

제안 방법은 CRGB에서 정점에 값을 부여했기 때문에 정점이 같은지 확인하기 위해 두 가지 방법을 이용한다. 첫 번째로 멤버 변수 정점의 경우 문자열 매칭을 이용해 정점이 같은지 확인한다, 두 번째로 메소드의 경우 지문법 정보를 이용해 피어슨 상관계수를 구한 뒤 0.9 이상의 상관계수 값이 나오는 경우 같은 정점으로 정의한다.

제안 방법에서는 그래프를 효과적으로 다루기 위해 멤버 변수 집합을 기준으로 그래프 분할하여 유사도를 검사한다. 이 방법은 하나의 멤버 변수와 연관 있는 메소드 정점을 별도의 그래프로 생성하는 것으로 그래프를 분할한다. 제안 방법의 그래프 분할의 예는 [그림 4]와 같은 그래프를 [그림 5]와 같이 분할하는 것이다.

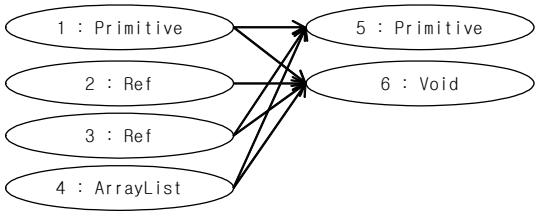


**그림 4. 분할하기 전의 CRGB. 멤버 변수 4개와 메소드 2개로 이루어진 클래스를 대상으로 추출한 CRGB이다.**

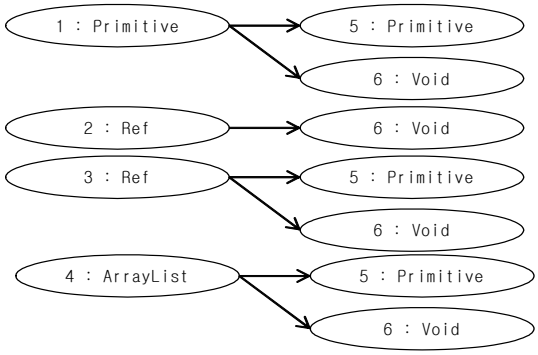


**그림 5. [그림 4]의 CRGB를 분할한 그래프. [그림 4]의 CRGB를 4개의 멤버 변수를 기준으로 분할한 결과이다.**

[그림 4]의 CRGB는 멤버 변수 4개와 그와 연관된 메소드가 2개인 클래스를 표현한 CRGB 버스마크이다. 그리고 제안 방법에 따라 멤버 변수 정점 4개를 기준으로 분할 그래프를 만든 것이 [그림 5]의 그래프이다. 유사도 검사기는 [그림 5]와 같은 그래프를 대상으로 그래프 동형 검사를 수행한다.

하지만 [그림 5]와 같이 분할 그래프는 메소드 측 정점이 중복되어 생성된다. 그러므로 그래프의 전체적 크기가 늘어나 효과적이지 못하다고 생각할 수 있다. 하지만 전체적인 그래프 정점의 개수는 늘어나지만, 그래프 동형 검사 시 대상이 되는 단독 그래프의 크기가 줄어들기 때문에 VF2와 같은 시간 복잡도를 가지는 알고리즘에서는 장점이 된다.

제안 방법에서는 유사도 점수를 측정하기 위한 기준으로 두 CRGB 사이의 간선 일치도를 소스코드의 유사도 점수로 사용한다. 위에서 언급한 바와 같이 분할 그래프는 그래프의 전체적인 정점 수가 중복되어 늘어나게 되지만 원본 CRGB와 간선의 수는 변함이 없다. 그러므로 분할 그래프에서도 원본 CRGB와 변함없는 정보를 유지하는 간선을 기준 대상으로 유사도 점수를 계산한다. 다음 식 (1)은 유사도 점수를 계산하는 방법이다.

$$sim(p,q)=\frac{|matched\,edge(CRGB_p,CRGB_q)|}{\max(|CRGB_p.Edge|,|CRGB_q.Edge|)}*100 \quad (1)$$

제안 방법에서 유사도는 식 (1)과 같이 두 CRGB의 전체 간선의 수 중 큰 쪽을 비교 대상으로 한다. 그리고 VF2 알고리즘을 통해 얻은 동형 그래프로 판단된 그래프의 간선의 수를 모두 합산한다. 그리고 전체 CRGB의 간선 중 몇 퍼센트가 동형 그래프에 포함되는가가 유사도 점수로 사용된다.

## V. 실험 및 평가

이 논문에서 실험은 Windows 7 professional SP1 환경에서 실행되었다. 제안 시스템은 CRGB를 생성하는 부분과 CRGB를 비교하고 유사도를 계산하는 부분으로 나누어져 있다. 그 중 CRGB 생성기는 Eclipse 환경에서 Java 1.7을 기준으로 개발되었으며, 유사도 계산기는 Visual Studio 2010 환경에서 C++를 이용하여 개발되었다.

그리고 제안 방법을 평가하기 위해 기존에 널리 알려진 유사도 분석 프로그램인 JPlag[20]와 Stigmata 2.0[21]을 이용한다. JPlag는 C/C++와 Java, Scheme, C# 등 다양한 언어를 지원하는 토큰 기반의 유사도 분석 프로그램이다. 이 프로그램은 웹을 통해 서비스되고 있으며 표절 검사를 원하는 소스코드 집합을 입력으로 주면 각 코드 간에 유사한 소스코드 집합을 보여준다. 그리고 Stigmata는 Java의 바이트 코드를 대상으로 버스마크를 생성하고 유사도를 분석하는 프로그램이다. 이 프로그램은 다양한 버스마크를 지원하지만, 이 실험에서는 많이 연구되고 있는 k-gram 버스마크를 이용한다. 그리고 실험 결과 k의 값이 4인 경우 가장 표절을 잘 검출하였기 때문에 k의 값은 4로 사용한다.

이 논문에서는 실험을 위한 대상 프로그램으로 2012년 객체지향 프로그래밍 수업에서 과제로 제출된 소스코드 그룹을 대상으로 선정하였다. 선정된 대상 코드 그룹은 기존에 표절이 있다고 알려진 그룹이다. 그리고 그 중 JPlag에서 유사도 점수가 높은 상위 10개 코드를 대상으로 한다. 실험 대상에 대한 정보는 [표 2]와 같다.

**표 2. 실험 대상 정보. JPlag에서 유사도 점수가 높은 상위 10개 프로그램의 전체 LOC는 716으로 나타났으며 전체 CRGB의 정점 수는 76개로 나타났다.**

| 코드명 | LOC | CRGB 정점 수 | 표절 대상 |
|---|---|---|---|
| A | 52 | 5 | X |
| B | 80 | 6 | X |
| C | 57 | 8 | X |
| D | 187 | 7 | X |
| E | 46 | 6 | X |
| F | 67 | 8 | X |
| G | 63 | 8 | X |
| H | 66 | 8 | X |
| I | 49 | 10 | J |
| J | 49 | 10 | I |
| 합계 | 716 | 76 | 2 |

[표 2]와 같이 10개의 실험 대상은 LOC는 716라인으로 구성된 프로그램이다. 그리고 실험 대상을 이용해

CRGB를 생성하면 그래프의 전체 정점은 76개로 생성된다. 그리고 10개의 프로그램을 대상으로 순서에 상관없이 두 프로그램 간의 유사도 검사를 수행하기 때문에 중복되는 쌍을 제거하면 45개 쌍을 대상으로 실험하게 된다. 다음 [표 3]은 선정된 코드 그룹을 대상으로 JPlag와 k-gram, 제안 방법을 적용해 얻어진 유사도 쌍의 유사도 점수 결과이다.

실험 대상은 [표 3]과 같이 전체 10개 소스코드에서 45개 쌍의 유사도 점수를 계산하였다. 기존의 객체지향 프로그래밍 수업에서는 60~70% 이상의 유사도 점수가 나오게 되면 표절로 의심하였기 때문에 이 논문에서도 60% 이상의 유사도 점수가 나오게 되면 표절로 의심한다. 그 기준을 적용해 보면 JPlag의 경우 3개의 소스코드 쌍을 확인해보아야 하며 k-gram 기반의 Stigmata는 5개 소스코드 쌍을 확인해야 한다. 하지만 제안 기법인 CRGB의 경우 2개의 소스코드 쌍만 확인하면 된다.

**표 3. JPlag, k-gram 기반 Stigmata, CRGB의 유사도 측정 결과. 60% 이상을 표절로 의심하게 되면 JPlag는 3쌍이 표절로 의심되며 k-gram은 5쌍이 그리고 CRGB는 2쌍이 표절로 의심되는 결과가 나타났다.**

| 분포(%) | JPlag | Stigmata | CRGB |
|---|---|---|---|
| 100 - 90 | 1 | 1 | 2 |
| 90 - 80 | 1 | 1 | 0 |
| 80 - 70 | 1 | 1 | 0 |
| 70 - 60 | 0 | 2 | 0 |
| 60 - 50 | 1 | 3 | 1 |
| 50 - 40 | 4 | 2 | 0 |
| 40 - 30 | 1 | 7 | 0 |
| 30 - 20 | 4 | 16 | 0 |
| 20 - 10 | 4 | 8 | 0 |
| 10 - 0 | 28 | 4 | 42 |

그리고 이 논문에서는 CRGB의 성능을 평가하기 위해 F-measure를 이용해 정확도를 측정하였다. F-measure는 정보 검색 분야에서 사용되는 측정요소로 정확도(precision)와 재현율(recall)을 이용해 검사의 정확성을 측정하는 방법이다. F-measure 계산하는 식은 식 (2)와 같다.

$$F\text{-}measure = \frac{2 \times (precision \times recall)}{(precision + recall)} \quad (2)$$

식 (2)에서 정확도(precision)는 표절로 보고된 결과가 실제 표절과 얼마나 일치하는지 확인하는 지표이다. 정확도가 높으면 높을수록 양성오류(false positive)를 보고하는 경우가 줄어든다. 그리고 재현율(recall)은 표절로 보고한 결과 중 얼마나 실제 표절을 포함하고 있는지 확인하는 방법이다. 재현율이 높으면 높을수록 음성오류(false negative)를 내보내는 경우가 줄어든다.

이 논문에서는 10개의 대상 프로그램의 성능 평가를 위해 실제 표절 여부를 코드 검토하여 확인하였다. 검토방법으로 실제 코드의 유사한 부분을 찾은 후 과제를 제출한 학생과 면담을 통해 표절 여부를 확인하였다. 그 결과 프로그램 I와 J가 서로 표절을 한 사실을 확인할 수 있었다. 그리고 이 결과를 이용해 JPlag와 k-gram 기반의 Stigmate, CRGB의 성능 평가 결과는 [표 4]와 같다.

**표 4. JPlag, k-gram 기반 Stigmata, CRGB의 성능 평가. 세 방법 모두 재현율은 동일하게 1로 나타났지만, 정확도와 F-measure에서는 CRGB가 가장 우수하게 나타났으며 다음은 JPlag, k-gram 순으로 나타났다.**

| 평가 지표 | JPlag | Stigmata | CRGB |
|---|---|---|---|
| 정확도 | 0.33 | 0.20 | 0.50 |
| 재현율 | 1.00 | 1.00 | 1.00 |
| F-measure | 0.50 | 0.33 | 0.67 |

[표 4]의 결과 모든 유사도 계산기의 재현율은 1로 계산되었다. 하지만 정확도에서 JPlag는 0.33으로 k-gram 기반의 Stigmata는 0.20으로 나타났으며 CRGB는 0.50으로 가장 높게 나타났다. 그리고 F-measure의 경우 CRGB가 가장 높게 나타났으며, JPlag보다 0.17, Stigmata보다 0.34 높게 나타났다. 즉 제안 방법은 현재 실험에서 양성오류가 JPlag에서는 17%, k-gram에서는 30% 감소한 것을 확인할 수 있다.

## VI. 결 론

이 논문에서는 프로그램 표절을 검사하는 방법으로

클래스 구조를 이용하는 정적인 버스마크인 CRGB를 제안하였다. CRGB는 클래스의 메소드에서 멤버 변수가 사용되었다면 서로 연관 관계가 있는 것으로 간주해 멤버 변수와 메소드의 자료형을 그래프의 정점으로 추가하고 간선으로 연결하는 방법이다.

이 논문에서는 CRGB의 유사도 계산을 위해 그래프 동형 문제를 사용하였다. 그래프 동형 문제 해결을 위해 기존에 널리 알려진 라이브러리인 VFLib를 이용해 유사도 계산을 해결하였다. 그리고 그래프 동형 문제는 NP-완전 문제이기 때문에 그래프 분할을 통해 입력되는 그래프의 크기를 줄이는 방법을 제안하였다.

제안 방법을 평가하기 위해 2012년 객체지향 프로그래밍 수업에서 제출된 과제를 대상으로 실험을 진행하였다. 그리고 다른 유사도 검출 프로그램과 비교를 위해 JPlag와 Stigmata를 이용해 정확도와 재현율, F-measure를 계산하였다. 계산 결과 F-measure를 기준으로 CRGB가 JPlag보다 0.17, Stigmata보다 0.34 높게 나타났다.

향후 연구로 CRGB의 정확도를 높이는 방법에 관한 연구가 필요하다. 현재 제안한 CRGB는 클래스의 구조가 비슷하다면 다른 프로그램도 유사하다고 나올 가능성이 있다. 즉 소스코드의 형태에 따라 양성오류가 발생할 가능성이 높다. 그러므로 이를 개선하기 위한 연구가 필요하다. 그리고 제안 방법은 작은 프로그램보다 대규모 프로그램의 표절 검사에 적합할 것으로 예상되기에 이에 관한 실험 및 분석이 필요하다.

또 다른 향후 연구로는 클래스 간의 관계를 표현하는 방법에 관한 연구가 필요하다. 현재 제안방법은 클래스의 유사도를 분석하는데 초점이 맞춰져 있다. 하지만 객체 지향 프로그램의 특성상 객체 간의 관계 즉 상속이나 객체 간의 통신도 고려하는 유사도 분석이 이루어진다면 표절 검사의 정확도는 더 올라갈 것으로 생각된다.

## 참 고 문 헌

[1] 차병래, “소프트웨어 소스 코드의 저작권 관리를 위한 디지털 라이센스의 검색”, 한국콘텐츠학회 논문지, 제7권, 제1호, pp.21-31, 2007.

[2] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, “Java Birthmark - Detecting the Software Theft,” IEICE Transactions on Information and Systems, Vol.88, No.9, pp.2148-2158, 2005.

[3] 나지하, 김종원, 김재석, “신분증 위변조 방지를 위한 이미지 워터마킹”, 한국콘텐츠학회논문지, 제11권, 제12호, pp.552-559, 2011.

[4] G. Myles and C. Collberg, “K-gram Based Software Birthmarks,” Proceedings of the 2005 ACM symposium on Applied computing, pp.314-318, 2005.

[5] S. Choi, H. Park, H. Lim, and T. Han, “A Static Birthmark of Binary Executables Based on API Call Structure,” Proceedings of the 12th Asian computing science conference on Computer and Network Security, pp.2-16, 2007.

[6] G. Myles and C. Collberg, “Detecting Software Theft via Whole Program Path Birthmarks,” Information Security Conference, pp.404-415, 2004.

[7] H. Tamada, K. Okamoto, M. Nakamura, and A. Monden, “Dynamic Software Birthmarks to Detect the Theft of Windows Applications,” In Proceeding of International Symposium on Future Software Technology 2004, 2004.

[8] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. ichi Matsumoto, *Design and Evaluation of Dynamic Software Birthmarks Based on API Calls*, Technical Report NAIST-IS-TR2007011, Nara Institute of Science and Technology, 2007.

[9] X. Wang, Y. Jhi, S. Zhu, and P. Liu, “Behavior Based Software Theft Detection,” Proceedings of the 16th ACM Conference on Computer and Communications Security, pp.280-290, 2009.

[10] Y. Kim, J. Moon, D. Kim, Y. Jeong, S. Cho, M. Park, and S. Han, “A Static Birthmark of

Windows Binary Executables Based on Strings," 2013 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp.734-738, 2013.

[11] J. Choi, Y. Han, S. Cho, H. Yoo, J. Woo, M. Park, Y. Song, and L. Chung, "A Static Birthmark for MS Windows Applications Using Import Address Table," 2013 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp.129-134, 2013.

[12] P. Chan, L. Hui, and S. Yiu, "Dynamic Software Birthmark for Java Based on Heap Memory Analysis," 12th IFIP TC 6 / TC 11 International Conference on Communications and Multimedia Security, pp.94-107, 2011.

[13] C. Roy and J. Cordy, *A Survey on Software Clone Detection Research*, Technical Report 541, Queen's University at Kingston. 2007.

[14] S. Narayanan and S. Simi, "Source Code Plagiarism Detection and Performance Analysis Using Fingerprint Based Distance Measure Method," 2012 7th International Conference on Computer Science & Education, pp.1065-1068, 2012.

[15] J. Ji, G. Woo and H. Cho, "A Source Code Linearization Technique for Detecting Plagiarized Programs," ACM SIGCSE Bulletin, Vol.39, No.3, pp.73-77, 2007.

[16] C. Liu, C. Chen, J. Han, and P. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis," Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.872-881, 2006.

[17] https://code.google.com/p/javaparser/

[18] http://www.cs.sunysb.edu/~algorith/implement/vflib/implement.shtml

[19] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) Graph Isomorphism Algorithm for Matching Large Graphs," IEEE Trans on Pattern Analysis and Machine Intelligence, Vol.26, No.10, pp.1367-1372, 2004.

[20] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding Plagiarisms Among a Set of Programs with JPlag," J. UCS, Vol.8, No.11, pp.1016-1038, 2002.

[21] http://stigmata.sourceforge.jp/

## 저 자 소 개

**김 연 어(Yeoneo Kim)** 정회원

- 2010년 2월 : 동아대학교 컴퓨터공학과(공학사)
- 2012년 2월 : 동아대학교 컴퓨터공학과(공학석사)
- 2012년 3월 ~ 현재 : 부산대학교 박사과정

<관심분야> : 프로그래밍 분석, 정적 분석, 표절 검사

**이 윤 정(Yun-Jung Lee)** 정회원

- 1995년 2월 : 부경대학교 전자계산학과(이학사)
- 1999년 2월 : 부경대학교 전산정보학과(이학석사)
- 2008년 8월 : 부경대학교 전자계산학과(이학박사)
- 2008년 9월 ~ 2012년 8월 : 부산대학교 U-Port 정보기술사업단 박사후연구원
- 2012년 9월 ~ 2013년 8월 : 부산대학교 컴퓨터 및 정보통신연구소 기금교수
- 2013년 11월 ~ 현재 : 부산대학교 IT기반 융합산업창의인력양성 사업단 박사후연구원

<관심분야> : 얼굴 애니메이션, 웹 콘텐츠 시각화, 사회연결망 분석

우 균(Gyun Woo) 정회원

- 1991년 : 한국과학기술원 전산학 (학사)
- 1993년 : 한국과학기술원 전산학 (석사)
- 2000년 : 한국과학기술원 전산학 (박사)
- 2000년 ~ 2004년 : 동아대학교 컴퓨터공학과 조교수
- 2004년 ~ 현재 : 부산대학교 전자전기컴퓨터공학과 교수

<관심분야> : 프로그래밍언어 및 컴파일러, 함수형 언어, 그리드컴퓨팅, 소프트웨어 메트릭, 프로그램 분석, 프로그램 시각화