예약어 시퀀스 탐색을 통한 소스코드 표절검사

Source Codes Plagiarism Detection By Using Reserved Word Sequence Matching

[1]이영주, [2]김승, [3]강석호

[1] 서울대학교 산업공학과 (spicio@naver.com)
[2] 서울대학교 산업공학과 (seung275@netopia.snu.ac.kr)
[3] 서울대학교 산업공학과 (shkang@cybernet.snu.ac.kr)

Abstract

프로그램 소스코드 표절 검사에 대한 기존 방법은 크게 지문(finger-print)법과 구조기반 검사법으로 나뉘며, 주로 단어의 유사성이나 발생빈도를 사용하거나 소스코드 구조상의 특징으로 두 소스간의 유사성을 비교한다. 본 연구에서는 프로그래밍 언어의 예약어 시퀀스를 사용하여 소스코드들 간의 유사성을 비교하고, 이 결과를 FCA(Formal Concept Analysis)를 통해 해석하고 시각화 하는 방법을 제시한다. 일반적인 VSM(Vector Space Model)과 같은 단일 단어 분석으로는 단어의 인접성을 구분할 수 없으므로 단어의 시퀀스 분석이 가능하도록 알고리즘을 구성하였으며 이러한 방식은 지문법의 단점인 소스코드의 부분적인 표절 탐지의 난점을 해결할 수 있고 함수의 호출 순서나 수행 순서에 상관없이 표절을 탐지할 수 있는 장점을 가진다. 마지막으로 유사도 측정결과는 FCA를 이용하여 격자(lattice)로 시각화됨으로써 이용자의 이해도를 높일 수 있다.

1. 서론

WWW(World Wide Web)을 위시한 인터넷의 폭발적인 발달은 수많은 자료 및 정보의 공유를 가능케 하고 있으며, 웹 사용자가 접근할 수 있는 웹문서의 양은 현재 80억개 이상으로 추산된다 [4]. 그러나 이러한 정보의 폭발적인 증가에 비해 보안이나 복제방지 대책은 미흡한 실정이며 인터넷 등의 다양한 매체의 활성화에 따른 정보의 증가가 정보 복제를 더욱 쉽게 만드는 주요인이 되고 있다 [3].

문서의 표절 탐지에 관련된 연구는 표절 탐지 대상에 따라 크게 일반 문서에 대한 표절 탐지와 프로그램 소스코드의 표절 탐지로 나뉜다. 또한 표절탐지 기법에 따라 지문(finger-print)법과 구조기반(structure-related) 검사법으로도 구분된다.

일반 문서는 단락의 삽입, 순서 변경, 삭제가 비교적 자유로운 반면, 프로그램 소스코드에서는 구조의 변경에 제약이 있게 된다 [1]. 지문법은 일반 문서의 표절탐지에 잘 적용되며 특정 단어의 출현 빈도를 나열한 단어히스토그램 등의 방법을 사용하여 표절 검사를 수행한다. 구조기반 검사법은 일반 문서의 표절 보다 제어흐름(구조)을 가지고 있는 프로그램 소스코드 표절 검사에 많이 사용된다. 일반 문서와 달리 소스코드는 제어흐름의 변경이 어렵고, 프로그래밍 문법이 정해져 있기 때문에 구조적인 특징이 잘 나타난다 [1][2]. 소스코드에서 토큰(예약어)을 추출하여 나열하면 선형구조를 이루는 서열을 만들 수 있고, 이 토큰 서열들에서 공통 시퀀스(Common Subsequence)들을 찾아서 유사성의 척도로 삼을 수 있다.

문서간의 유사도 측정에서 일반적으로 쉽게 접근 할 수 있는 방법이 VSM(Vector Space Model)이다. VSM은 문서 분류나 클러스터링에 가장 폭넓게 사용되는 데이터 모델이다. VSM은 모든 문서집합에서 색인단어(indexing term)을 추출한 후 각각의 문서를 이 색인단어의 특징 벡터(feature vector)로 나타낸다. 각각의 특징벡터의 엘리먼트에는 해당 문서에 나타난 단어의 빈도수를 사용하여 가중치가 기입되며, 두 문서간의 유사도는 이러한 특징 벡터를 비교하여 측정된다. 특징 벡터 비교 방법으로는 흔히 Cosine measure나 Jaccard measure가 사용된다 [6]. VSM이 일반적인 문서간 유사도 측정법이기는 하지만 기본적으로 VSM은 단어 단위로 분석을 한다는 점에서 한계를 가진다. 즉, VSM에서는 색인 단어의 근접성(proximity)은 무시되고 있다 [6].

본 연구에서는 소스코드의 표절 탐지에 중점을 두고 지문법과 구조기반 검사법의 장점을 동시에 취할 수 있는 알고리즘을 제안한다. 본 연구에서 제안하는 알고리즘은 토큰(예약어)의 시퀀스 분석이 가능하도록 VSM에 기반한 알고리즘을 구성하여 공통 시퀀스들을 추출하는데 이를 활용하였다. 또한 분석결과는 FCA(Formal Concept Analysis)를 사용하여 사용자에게 분석결과를 시각화하여 보여줄 수 있도록 하였다.

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 문서의 표절 탐지에 관련된 기존의 연구와 FCA의 기본 개념을 정리하였다. 3장에서는 본 연구에서 사용한 색인구조와 알고리즘을 정리하였다. 4장에서는 공통 시퀀스를 이용한 유사도 측정 방법을 제시하며, 5장에서는 FCA를 이용한 소스코드 간

관계 시각화 방법을 정리하였다. 6장에서는 본 연구에서 제안한 색인구조와 알고리즘에 따른 소스코드간의 유사도 측정과 소스코드간 관계 시각화 예를 실험을 통해 보이고, 마지막으로 7장에서 결론을 맺는다.

2. 관련연구

2.1 소스코드 표절 탐지

문서의 표절 연구에는 크게 지문법과 구조기반 방법으로 나눌 수 있는데, 두 방법의 특징은 <표 1>과 같이 간단히 정리할 수 있다.

소스코드 표절에 대해 Hamblen은 “한 프로그램에 약간의 루틴의 변형을 가해서 만들어내는 프로그램”이라고 정의하였다 [5]. 여기서 변형이라는 의미는 원본 소스코드의 주석문을 바꾸거나 변수의 타입, 지정자를 수정하는 것과 중복된 구문이나 불필요한 변수를 삽입하는 경우 등을 말한다. 또한 원본 소스코드를 논리적인 흐름에 영향을 주지 않도록 단순하게 구조를 뒤섞는 일도 여기에 포함된다.

<표 1> 지문법과 구조 기반 방법의 비교(출처: [1])

설명	지문법	구조기반방법
문서 비교를 위한 정보	유동적	고정적
문서의 길이	영향을 받지 않음	길이에 따라 얻어내는 정보량은 비례함
부분적인 복제 탐지	어려움	쉬움
구현 시스템	Ottenstein, Donaldson, Berghel	CHECK, Plaque, YAP, MOSS, JPlag, SIM, SID

토큰(예약어) 시퀀스를 이용한 소스코드 표절 탐지에 관한 연구 중 생물 정보학을 이용한 연구가 있다 [1]. 즉, 특정 유전체 서열이 어떤 생물학적인 정보를 가지는지를 유전체 서열의 비교분석을 통해서 분석해 볼 수 있으며 이러한 방법을 소스코드 표절 탐지에 활용할 수 있다. 유전체 서열 분석 연구는 DNA 서열이나 단백질 서열들을 비교하여 유사한 영역을 찾아내는 것인데, 이것은 동일한 서열을 가지는 유전자는 같은 기능을 가진다는 가정 하에 동종이나 타종의 유전체 서열의 기능 분석에 사용된다. 즉, 프로그램의 소스코드의 구조와 특성을 나타내는 정보를 추출하여 생물학에서 이용되는 여러 가지 정렬 시스템을 사용하여 비교한다. 다음 <표 2>는 소스코드를 아미노산 서열로 대응시키는 과정을 보여준다.

<표 2>에서처럼 사용자가 지정한 토큰을 소스코드에서 추출하면 “void void { long int int = = = ;}"이다. 따라서 이를 유전체 서열로 바꾸면 ”VVDL IISSSW"가 된다.

2.1 FCA(Formal Concept Analysis)

Formal Concept Analysis(FCA)는 이항관계(binary relationship)를 분석하기 위한 수학적 기법이다. FCA는 주어진 자료를 컨셉(concept)이라 불리는 추상개념으로 구조화한 후, 하위개념과 상위개념 관계로 순서화한 컨셉격자(concept lattice)로 표현한다 [7].

<표 2> 소스코드에서 키워드 추출한 후 유전체 서열로 바꾸는 과정 (출처: [3])

소스코드의 예	추출되는 키워드	대응되는 염기서열

#include <studio.h>
void main(void) {
long result;
int n1, n2, m1, m2, divs[100],
lnum, I, flag;
int index=0;
printf("Input 2 numbers for
calculating GCM...₩n");
scanf("%d %d:, &n1, &n2);
M1=n1; m2= n2;}
	void void {
	long
	int
	
	int =
	
	= = }
		V V D
		L
		I
		
		I S
		
		S S W

FCA의 입력으로 사용되는 자료는 컨텍스트(*context*)로 부르며, 객체의 집합 E(*Entity*), 속성의 집합 F(*Feature*), 객체와 속성의 이항관계 집합 R(*Relationship*)로 이루어진 트리플 (E,F,R)이다. 집합이 유한일 때, 분석대상의 컨텍스트는 표로 나타낼 수 있고, 다음 <표 3>은 그 예이다.

<표 3> 예시 컨텍스트

	f_1	f_2	f_3
e_1	X	X	
e_2			X
e_3	X		
e_4			X

주어진 객체의 집합 $E'(\in E)$에 속하는 객체의 공통속성(*Common Feature*) CF는 다음과 같다.

$$CF(E') = {}_{\text{def}}\{f \in F \mid \forall e \in E', (e,f) \in R\}$$

이와 유사하게, 주어진 속성의 집합 $F'(\in F)$에 속하는 속성의 공통객체(*Common Entity*) CE는 다음과 같다.

$$CE(F') = {}_{\text{def}}\{e \in E \mid \forall f \in F', (e,f) \in R\}$$

예를 들어, <표 3>에서 $CF(\{e_1,e_3\})=\{f_1\}$이고 $CE(\{f_3\})=\{e_2,e_4\}$이다.

컨텍스트 (E,F,R)의 컨셉은 $CF(E')=F'$, $CE(F')=E'$ 를 만족하는 (E',F')로 정의된다. 즉, 컨셉 (E',F')에서 객체 E'는 속성 F'를 공통요소로 가지며, 속성 F'는 객체 E'를 공통요소로 가진다. 예를 들어, <표 3>에서 $(\{e_1,e_3\},\{f_1\})$, $(\{e_1\},\{f_1,f_2\})$는 주어진 컨텍스트의 컨셉이다. 컨셉은 다음과 같이 부분순서(partial order)를 가진다.

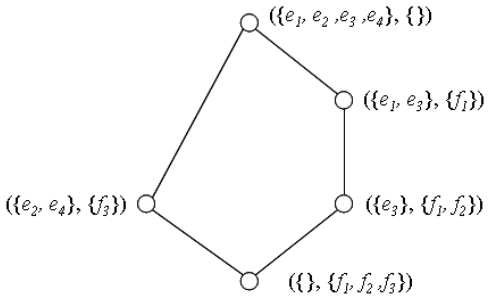
$$(E_i,F_i) \le (E_j,F_j) \Leftrightarrow E_i \subseteq E_j$$

마찬가지로 다음 관계도 성립한다.

$$(E_i,F_i) \le (E_j,F_j) \Leftrightarrow F_i \subseteq F_j$$

예를 들어 $(\{e_1\},\{f_1,f_2\}) \le (\{e_1,e_3\},\{f_1\})$이며, 두 컨셉은 하위개념과 상위개념 관계로 파악될 수

있다. 이와 같은 방법으로 모든 컨셉을 순서화하면, <그림 1>과 같은 컨셉격자를 얻을 수 있다. 다이어그램의 하위노드는 많은 속성을 가진 컨셉을, 상위노드는 많은 객체를 가진 컨셉을 의미한다. 따라서 하위노드로 갈수록 구체적인 개념이, 상위로 갈수록 일반적인 개념이 나타난다. 이와 같이 컨셉격자를 이용하면, 주어진 컨텍스트를 시각화 할 수 있어 분석이 용이해진다.



<그림 1> 예시 컨텍스트에 대한 컨셉격자

3. 색인구조

본 연구에서는 비교되는 소스코드들에 사용된 공통 예약어 시퀀스를 소스코드들간의 유사도 판단의 근거로 삼고자 한다. 따라서 여기에 사용될 색인구조 또한 이러한 공통 예약어 시퀀스를 잘 나타낼 수 있어야 한다.

VSM은 색인구조로 단어-문서 행렬(term-document matrix)을 사용하며 예약어와 소스코드를 각각 단어와 소스코드에 대응시킬 때 단어-문서 행렬을 하나의 색인구조의 대안으로 생각할 수 있다. 그러나 이러한 구조에서는 예약어의 시퀀스 정보가 무시되어 올바른 분석이 힘들다. 따라서 본 연구에서는 기존 단어-문서 행렬과 다른 색인구조를 사용해야 하며 [6]에서 제시한 DIG(Document Index Graph) 구조를 소스코드 분석 도메인에 알맞게 수정, 보완하여 사용한다.

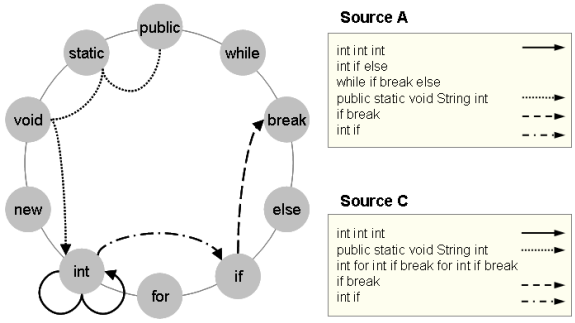
3.1 색인구조

본 연구에서 사용하는 색인그래프(indexing graph)는 유방향 그래프 $G=(V,E)$ 구조를 가진다. 이 때 V,E 는 다음과 같다.

$V=\{v_1,v_2\cdots,v_n\}$, 노드의 집합으로, 노드 v는 전체 소스코드 집합에서 유일한 예약어를 나타낸다. 본 연구에서 분석 대상으로 삼는 자바 예약어는 49개이므로, 노드는 49개 까지 존재할 수 있다.

$E=\{e_1,e_2\cdots,e_m\}$, 엣지의 집합으로, 엣지 e는 노드의 순서쌍 (v_i,v_j)를 나타내며, 노드 v_i로부터 노드 v_j로의 연결을 의미한다. 엣지 (v_i,v_j)는 어떤 코드에서 예약어 v_j가 예약어 v_i 뒤에 연속하여 나타남을 의미한다.

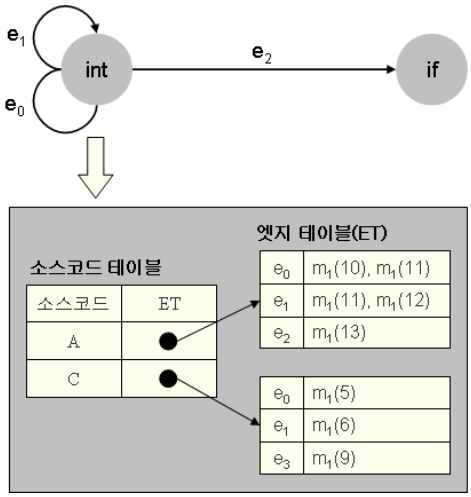
다음 <그림 2>는 최소공배수를 구하는 소스코드 A, B, C 중 서로 다른 알고리즘을 사용한 소스코드 A, C에서 추출한 공통 예약어 시퀀스를 그래프로 표현한 예이다. 소스코드 A, B, C의 실제 코드 내용은 부록에 첨부하였다.



<그림 2> 예시 색인그래프

3.2 저장구조

색인그래프는 테이블 형태로 저장된다. 다음 <그림 3>은 색인그래프가 저장되는 예를 보여준다.



<그림 3> 색인그래프 저장 예시

예시 색인그래프에서 노드 int, if 와 엣지 e_0, e_1, e_2가 존재할 때, 노드 int는 소스코드 테이블과 엣지 테이블(ET)을 가진다. 소스코드 테이블은 예약어 int가 포함된 공통 예약어 시퀀스를 가진 소스코드이름(소스코드)과 노드를 포함한 엣지의 정보(ET)를 저장한다. 엣지 테이블은 앞서 소스코드 테이블에 저장된 엣지 정보를 표현하며, 노드를 포함한 엣지 이름과 소스코드에서 예약어 int의 위치를 저장한다. 예를 들어, <그림 3>에서 예약어 int를 포함한 공통 예약어 시퀀스는 소스코드 A, C에서 모두 나타나며, 소스코드 A에서는 첫 번째 메소드의 열 번째, 열한 번째, 열두 번째, 열세 번째 예약어에 나타난다.

3.3 공통 예약어 추출 알고리즘

[6]에서 제시한 알고리즘은 웹페이지에서 일치구문을 추출하기 위한 것으로, 소스코드에서 공통 예약어 시퀀스를 추출하기 위해서는 소스코드 전처리 및 최장 공통 시퀀스(longest common sequence)를 찾는 과정이 추가적으로 필요하다. 따라서 본 연구에서는 [6]에서 제시한 알고리즘을 소스코드 도메인에 알맞게 수정한 공통 예약어 시퀀스 추출 알고리즘을 제안한다. 다음은 공통 예약어 시퀀스 추출 알고리즘에 사용된 기호 정의이다.

정의

s_i : i번째 소스코드
c_i : 소스코드 s_i에 포함된 클래스의 수
m_{ij} : 소스코드 s_i의 j번째 메소드
r_{ijk} : 메소드 m_{ij}의 k번째 예약어
l_{ij} : 메소드 m_{ij}에 포함된 예약어의 수
G_{i-1} : 소스코드 s_{i-1}까지 누적된 그래프
단, 처리된 소스코드가 없으면 G_0은 공집합
M : 소스코드 $s_1 \sim s_{i-1}$과 s_i 사이의 공통 예약어 시퀀스 목록
crs_{ij} : i번째 소스코드에 속한 j번째 공통 예약어 시퀀스

소스코드 집합으로부터 공통 예약어 시퀀스를 추출하는 알고리즘은 다음과 같다.

공통 예약어 시퀀스 추출 알고리즘

s_i 선택
for s_i의 모든 클래스 **do**
  if 코드가 메소드에 속해 있지 않음 **then**
    클래스 출현순서대로 코드를 $m_{i0},\cdots,m_{ic_i-1}$에 할당
  else
    출현순서대로 메소드를 m_{ic_i}부터 할당
  end if
end for
for s_i의 모든 m_{ij} **do**
  v_1에 r_{ij1} 입력
  if v_1이 G_{i-1}에 존재하지 않음 **then**
    G_{i-1}에 v_1 추가
  end if
  for r_{ijk}, $k=2,\cdots,l_{ij}$ **do**
    v_k에 r_{ijk} 입력
    v_{k-1}에 r_{ijk-1} 입력
    $e_k=(v_{k-1}, v_k)$
    if v_k가 G_{i-1}에 존재하지 않음 **then**
      G_{i-1}에 v_k 추가
    end if
    if e_k가 G_{i-1}에 존재 **then**
      e_k를 포함하는 공통 예약어 시퀀스를 추출하여 M에 추가
    end if
    if e_k가 G_{i-1}에 존재하지 않음 **then**
      e_k를 G_{i-1}에 추가
    end if
    v_{k-1}과 v_k의 소스코드 테이블 갱신
  end for
end for
G_i에 G_{i-1} 입력
for s_i에서 추가된 공통 예약어 시퀀스 **do**
  if $crs_{ij} \subset crs_{ik}$ **then**
    crs_{ij} 삭제
  end if
end for
M 출력

4. 유사도 측정

일치구문을 이용한 유사도 측정 방법은 [6]에서 연구된 바 있으나, 빈도 차이를 반영하지 못 하는 단점이 있다. 따라서 본 연구에서는 빈도 차이를 반영할 수 있는 유사도 측정 방법을 제시한다.

소스코드간의 유사도 측정 기본 아이디어는 다음과 같다.

$$\mathrm{sim}(x,y)=\frac{x\cap y}{x\cup y}$$

이 식을 적절히 변형하여, 공통 예약어 시퀀스를 이용한 소스코드 유사도 측정 방법은 다음과 같다.

소스코드간 유사도 측정

P : 공통 예약어 시퀀스 수
l_i : 공통 예약어 시퀀스 길이 ($i=1,\cdots,P$)
f_{1i}, f_{2i} : 소스코드 1, 2에서 공통 예약어 시퀀스 빈도
$|m_{1j}|, |m_{2k}|$: 소스코드 1, 2의 각 메소드 길이
o_1, o_2 : 소스코드 1, 2에서 공통 예약어 시퀀스가 중복적으로 추출된 예약어 수
$g(l_i)$: 공통 예약어 시퀀스 길이 가중치 함수, $g(l_i)=(l_i)^{\gamma}$

$$\mathrm{sim}_p(s_1,s_2)=\frac{\sqrt{\sum_{i=1}^{P}\left[g(l_i)(f_{1i}+f_{2i})\left(\frac{\min\{f_{1i},f_{2i}\}}{\max\{f_{1i},f_{2i}\}}\right)\right]^2}}{\sum_j |m_{1j}|+\sum_k |m_{2k}|+o_1+o_2}$$

5. 소스코드간 관계 시각화

공통 예약어 시퀀스 추출 알고리즘을 이용하면 소스코드 인덱스를 생성할 수 있는데, 인덱스의 빈도를 1이상과 0으로 구분하면 소스코드와 인덱스는 이항관계로 표시할 수 있다. 이항관계로 표시된 소스코드와 인덱스를 각각 객체 집합과 속성 집합으로 놓으면 하나의 컨텍스트를 생성할 수 있다.

컨텍스트에 기반한 컨셉격자는 소스코드간의 관계를 부분순서를 가지는 상위컨셉과 하위컨셉 관계로 시각화한다. 하위컨셉은 상위컨셉이 가지는 모든 인덱스를 가지며, 상위컨셉이 가지지 않은 하나 이상의 인덱스를 더 가진다. 따라서 컨셉이 컨셉격자의 아래쪽에 위치할수록 더 많은 인덱스를 가진다.

이와 같은 컨셉격자의 성질을 이용하면, 다음과 같은 방법으로 소스코드간의 관계를 시각적으로 이해할 수 있다.

• **컨셉 간의 거리**
유사도가 높은 소스코드들은 인덱스로 사용하는 공통 예약어 시퀀스를 많이 공유하므로 컨셉격자 상에서 서로 가까운 컨셉에 속하게 되며, 유사도가 낮은 소스코드는 공통 예약어 시퀀스를 적게 공유하므로 서로 먼 컨셉에 속하게 될 가능성이 크다.

• **동일 컨셉 상의 소스코드**
동일 컨셉에 속한 소스코드는 완전히 동일한 공통 예약어 시퀀스를 가지며, 유사도 차이는 공통 예약어 시퀀스의 빈도로만 결정된다. 유사도 측정 방법에서 빈도가 미치는 영향은 비교적 작은 편이므로, 같은 컨셉에 속한 소스코드간의 유사도는 매우 높을 가능성이 크다.

• **컨셉의 위치**
컨셉격자의 아래쪽에서 가까운 컨셉에 속한 소스코드일수록 공유하는 공통 예약어 시퀀스가 많아지므로, 컨셉에 속한 소스코드간의 유사도가 높아지는 경향을 보인다. 즉, 하위컨셉에 속한 소스코드간의 유사도는 상위 컨셉에 속한 소스코드간의 유사도보다 높을 가능성이 크다. 특히 최하위 부분에 위치한 컨셉에 속한 소스코드들은 표절로 강하게 의심할 수 있다. 그러나 최상위 부분에 위치한 컨셉에 속한 소스코드들은 일반적으로 사용되는 코드에서 추출된 예약어 시퀀스를 가지고 있을 가능성이 높아 표절일 가능성은 낮을 것이다.

즉, 소스코드와 공통 예약어 시퀀스의 이항관계에 기반한 컨셉격자는 소스코드간의 유사 관계를 시각화하여 표절 탐지에 도움을 줄 수 있다.

6. 실험

본 연구에서 제시한 알고리즘 및 유사도 측정 방법을 사용하여 최소공배수를 구하는 소스코드간 유사도를 측정하고 그 결과를 시각화하였다. 실험에 사용된 소스코드의 특징은 <표 4>와 같다.

<표 4> 소스코드 설명

소스코드	설명
A	원본 소스코드
B	소스코드 A와 동일한 알고리즘을 사용하였으나, 소스코드 A에 메소드 분할 또는 통합, 코드 재배치, 코딩 스타일 변형 등을 가한 유사 소스코드
C	소스코드 A와 다른 알고리즘을 사용하였으나, 소스코드 A와 유사한 방식으로 코딩된 소스코드

6.1 공통 예약어 시퀀스 추출

공통 예약어 시퀀스 추출 알고리즘을 이용하여 소스코드 A, B, C의 공통 예약어 시퀀스를 분석한 결과는 다음 <표 5>와 같다.

<표 5> 공통 예약어 시퀀스 분석 결과

공통 예약어 시퀀스	시퀀스 길이	빈도		
		A	B	C
public static void String int new int int new int	10	1	1	
int int int int	4	1	2	
int if else	3	1	1	
while if break else	4	1	1	
public static void String int	5	1	1	1
int int int	3	2	4	1
int if	2	1	1	2
if break	2	1	1	2

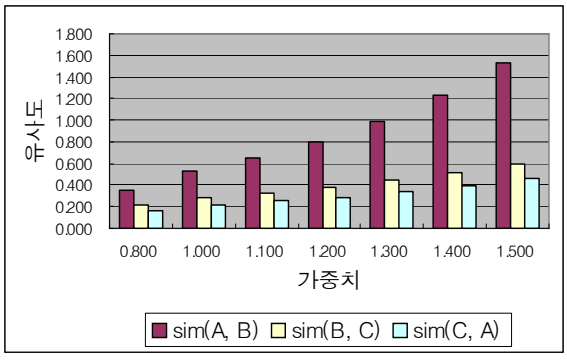
6.2 소스코드간 유사도 측정

앞서 제시한 유사도 측정 방법을 이용하여, 공통예약어 시퀀스 길이에 대한 가중치를 달리하면서 소스코드 A, B, C 간의 유사도를 측정한 결과는 다음 <표 6>과 같다.

<표 6> 유사도 측정 결과

가중치(γ)	sim(A, B)	sim(B, C)	sim(C, A)
0.8	0.352	0.217	0.164
1.0	0.526	0.287	0.217
1.1	0.647	0.331	0.250
1.2	0.799	0.382	0.290
1.3	0.989	0.442	0.336
1.4	1.227	0.513	0.390
1.5	1.526	0.596	0.454

유사도 측정 결과를 그래프로 나타낸 것은 다음 <그림 4>와 같다.



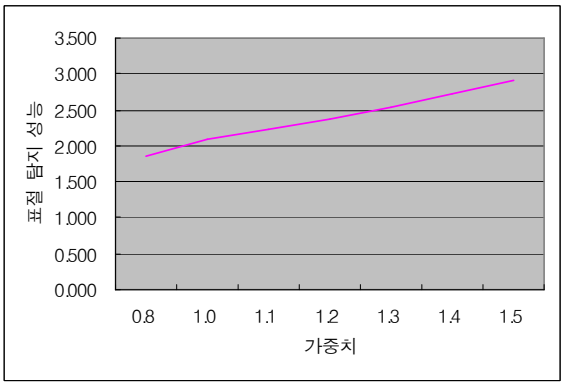
<그림 4> 유사도 측정 결과

유사도 측정 결과, 원본 소스코드 A와 이를 변형하여 작성한 소스코드 B 간의 유사도는 모든 가중치에서 소스코드 B와 C, 소스코드 C와 A 간의 유사도보다 높은 수치를 보였다.

유사한 소스코드간의 유사도와 유사하지 않은 소스코드간의 유사도가 잘 구분될수록 표절 탐지 성능이 높다고 말할 수 있다. 본 연구에서는 표절 탐지 성능을 측정하기 위한 방법으로, 유사한 소스코드간의 유사도와 유사하지 않은 소스코드간의 유사도 비율을 사용하였다. 즉, 표절 탐지 성능은 다음과 같이 측정한다.

$$\text{표절 탐지 성능} = \frac{\text{유사한 소스코드 간 유사도의 평균}}{\text{유사하지 않은 소스코드 간 유사도의 평균}}$$

소스코드 A, B, C에서 시퀀스 길이 가중치 변화에 따른 표절 탐지 성능은 다음 <그림 5>와 같다.



<그림 5> 가중치 변화에 따른 표절 탐지 성능

표절 탐지 성능은 시퀀스 길이 가중치가 증가함에 따라 증가하는 경향을 보이나, 적절한 가중치를 선택하여 소스코드간 유사도 범위가 일반적인 유사도 수치 범위 0~1을 가지도록 하는 것이 바람직하다.

예를 들어, γ=1.3에서 소스코드 A와 B의 유사도 0.989, 소스코드 B와 C의 유사도 0.442, 소스코드C와 A의 유사도 0.336이므로, 본 실험에서는 시퀀스 길이 가중치를 1.3으로 선택하는 것이 적절한 것으로 나타났다.

6.3 소스코드 관계 시각화

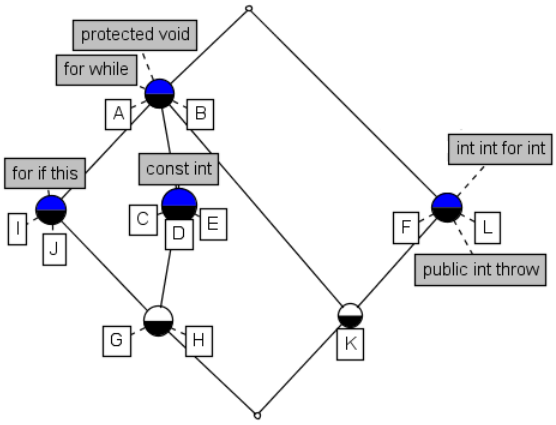
앞서 사용한 세 개의 소스코드 A, B, C간의 관계를 컨셉격자를 통해 시각화하면, 두 개의 컨셉으로 이루어진 단순한 컨셉격자가 생성된다. 컨셉이 두 개인 단순한 컨셉격자는 소스코드간 관계를 컨셉격자를 통해 해석하는 방법의 예시로 사용하기에 적절치 않다. 따라서 컨셉격자를 통한 소스코드간 관계 시각화 및 해석의 예로 임의의 소스코드 12개를 사용하였다.

실험을 위해 임의의 소스코드 12개를 이용하여 공통 예약어 시퀀스를 추출하고, 다음과 같은 컨텍스트를 생성하였다.

<표 7> 12개 소스코드에서 추출한 공통 예약어 시퀀스에 기반한 컨텍스트

	for if this	protected void	for while	public int throw	int int for int	const int
A		X	X			
B		X	X			
C		X	X			X
D		X	X			X
E		X	X			X
F				X	X	
G	X	X	X			X
H	X	X	X			X
I	X	X	X			
J	X	X	X			
K		X	X	X	X	
L				X	X	

<그림 6>에서는 위의 컨텍스트를 이용하여 컨셉격자로 소스코드 간의 관계를 시각화하였다. 컨셉격자 생성은 ConExp-1.2[8]를 사용하였다.



<그림 6> <표 7>의 컨텍스트를 이용하여 생성한 컨셉격자

<그림 6>의 컨셉격자는 앞서 제시한 3가지 방법으로 해석할 수 있으며, 다음은 컨셉격자를 해석하는 예이다.

- **컨셉 간의 거리**

  소스코드 G, H와 소스코드 I, J가 속한 컨셉은 하나의 엣지를 통해 직접 연결되어 있다. 반면 소스코드 G, H와 소스코드 A, B가 속한 컨셉은 두 개의 엣지를 거쳐 연결되어 있다. 즉, 소스코드 G, H와 소스코드 I, J의 유사도에는 한 개의 공통 예약어 시퀀스 const int의 유무에 따른 차이가 반영된 반면, 소스코드 G, H와 소스코드 A, B의 유사도에는 두 개의 공통 예약어 시퀀스 const int, for if this의 유무에 따른 차이가 반영되어 있다. 따라서 소스코드 G, H와 소스코드 I, J 간의 유사도는 소스코드 G, H와 소스코드 A, B 간의 유사도보다 높을 가능성이 크다.

• **동일 컨셉 상의 소스코드**
소스코드 G, H는 하나의 컨셉에 속한다. 즉, 소스코드 G, H는 완전히 동일한 공통 예약어 시퀀스를 가지며, 그 빈도에서만 차이가 있을 수 있다. 빈도의 차이는 유사도에 미치는 영향이 비교적 작으므로, 소스코드 G, H는 유사도가 매우 높을 가능성이 크다. 따라서 소스코드 G, H는 표절로 의심할 수 있다.

• **컨셉의 위치**
소스코드 G, H와 소스코드 A, B는 각각 하나의 컨셉에 속한다. 그러나 소스코드 G, H는 소스코드 A, B에 비하여 컨셉격자 상에서 아래쪽에 위치한다. 즉, 소스코드 G, H 사이에는 소스코드 A, B 사이보다 공유하는 공통 예약어 시퀀스가 많다. 따라서 소스코드 G, H의 유사도가 소스코드 A, B의 유사도보다 높을 가능성이 매우 크다.

7. 결론 및 향후 연구과제

본 연구에서는 프로그래밍 언어의 예약어 시퀀스를 사용하여 소스코드들 간의 유사성을 비교하고, 이 결과를 FCA(Formal Concept Analysis)를 통해 해석하고 시각화 하는 방법론을 제시하였다. 본 연구에서 제시한 공통 예약어 시퀀스를 이용한 표절 탐지 기법의 기여는 다음과 같다.

- 일반적인 VSM(Vector Space Model)과 같은 단일 단어 분석으로는 단어의 인접성을 구분할 수 없으므로 단어의 시퀀스 분석이 가능
- 메소드 단위로 예약어 시퀀스를 이용함에 따라 함수의 호출 순서나 수행 순서에 상관없이 표절을 탐지
- 공통 예약어 시퀀스를 이용하여 소스코드를 컨셉격자로 시각화함으로써 소스코드간 관계에 대한 이해도를 높임

마지막으로 향후 연구과제는 다음과 같다.

- 다수의 실험을 통한 적절한 시퀀스 길이 가중치 결정
- 컨셉격자에서 컨셉간의 거리에 소스코드간 유사도 반영
- 큰 문서집합를 사용하여 FCA 분석시 컨셉격자의 복잡도 감소를 위한 컨셉 근사화(concept approximation)
- 기존 문서간 유사도 측정 방법 개선

참고문헌

[1] 강은미, 황미녕, 조환규, 유전체 서열의 정렬 기법을 이용한 소스코드 표절 검사, 정보과학회논문지9(3), 2003, pp.352-367.
[2] 김영철, 유재우, 유사도 평가를 위한 트리 비교 알고리즘, 정보처리학회논문지A, Vol.11, No.2, 2004, pp.159-164.
[3] 김영철, 최재영, 구문트리에서 키워드 추출을 이용한 프로그램 유사도 평가, 정보처리학회논문지A, Vol.12, No.2, 2005, pp.109-116.
[4] Barabasi A., Albert R., and Jeong H.: Scale-free characteristics of random networks: the topology of the world-wide web. Physica A 281 (2000) 69-77
[5] J. O. Hamblen & Parker, "Computer Algorithm for Plagiarism Detection", IEEE Transactions on Education 32(2), pp. 94-99. May. 1989
[6] Khaled M. Hammouda, Mohamed S. Kamel, Efficient Phrase-Based Document Indexing for Web Document Clustering, IEEE Transactions on knowledge and data engineering, Vol.16, No.10, 2004, pp.1279-1296.
[7] Patrick du Boucher-Ryan, Derek Bridge, Collaborative Recommending using Formal Concept Analysis, Knowledge-Based System, Vol.19, 2006
[8] http://www.aifb.uni-karlsruhe.de/pipermail/fca-list/2003-July/000052.html

부록

• **소스코드 A (최대공약수와 최소공배수를 구하는 자바 소스코드)**
  * 밑줄은 추출되는 예약어를 나타냄

```
class   Lcm{
        public static void main(String[] args){
                int[] num = new int[2];
                int[] temp = new int[2];
                int r = 1;
                int lcm = 0;

                num[0] = Integer.parseInt(args[0]);
                num[1] = Integer.parseInt(args[1]);

                if( num[0] > num[1]) {
                        temp[0] = num[0];
                        temp[1] = num[1];
                } else {
                        temp[0] = num[1];
                        temp[1] = num[0];
                }

                while( r != 0){
                        r = temp[0] % temp[1];

                        if( r == 0) {
                                int gcm = temp[1];
                                break;
                        } else {
                                temp[0] = temp[1];
                                temp[1] = r;
                        }
                }

                lcm = ( num[0] * num[1]) / gcm;

                System.out.println("G.C.M : " + gcm);
                System.out.println("L.C.M : " + lcm);
        }
}
```

• **소스코드 B (소스코드 A의 유사코드, 동일 알고리즘 사용, 메소드 분할, 통합, 코드 재배치 등의 변형이 가해짐)**

```
class   Sim{
        public static void main(String[] args){
                int[] num = new int[2];
                int[] temp = new int[2];

                num[0] = Integer.parseInt(args[0]);
                num[1] = Integer.parseInt(args[1]);

                if( num[0] > num[1]) {
                        temp[0] = num[0];
                        temp[1] = num[1];
                } else {
                        temp[0] = num[1];
                        temp[1] = num[0];
                }

                System.out.println("G.C.M : " + gcm(temp));
                System.out.println("L.C.M : " + lcm(num, gcm(temp)));
        }
```

```
        private static int lcm(int[] num, int gcm){
                int result = 0;

                result = ( num[0] * num[1]) / gcm;

                return result;
        }

        private static int gcm(int[] num){
                int r = 1;
                int result = 0;

                while( r != 0){
                        r = num[0] % num[1];

                        if( r == 0) {
                                result = num[1];
                                break;
                        } else {
                                num[0] = num[1];
                                num[1] = r;
                        }
                }

                return result;
        }
}
```

• **소스코드 C (소스코드 A의 유사코드, 소스코드 A와 다른 알고리즘을 사용)**

```
class Sim3 {

        public static void main(String args[]) {
                int n = Integer.parseInt(args[0]);
                int m = Integer.parseInt(args[1]);
                int gcm = 0, lcm = 0;

                for(int i=1;i<=m*n;i++) {
                        if((i%n==0) && (i%m==0)){
                                lcm = i;
                                break;
                        }
                }

                System.out.println("L.C.M : " + lcm );

                for(int i=n;i>=1;i--){
                        if((m%i==0) && (n%i==0)){
                                gcm = i;
                                break;
                        }
                }

                System.out.println("G.C.M : " + gcm );
        }
}
```