그래프 모델링을 이용한 프로그램 복제의 판별

Clone detection in a software using graph modeling

Abstract

컴퓨터의 사용이 보편화되면서 필요에 따라 수많은 프로그램들이 생겨났고, 이와 함께 프로그램의 복제, 위조 등 지적 재산권 침해에 대한 문제가 대두되었다. 컴퓨터 프로그램의 복제는 개발자의 의욕을 크게 저하시킨다. 따라서 프로그램의 복제 여부를 쉽게 판별할 수 있다면 그것만으로도 소프트웨어 산업의 발달에 긍정적인 영향을 줄 것이다.

본 논문에서는 이전의 연구에서 사용된 스트링 매칭이나 신경망 네트워크(Neural-network)와는 다른 새로운 알고리즘을 제시하고자 한다. 즉, 프로그램의 구조를 분석하여 프로그램에 포함된 요소들 사이의 관계를 그래프로 나타낸 이후, 두 프로그램의 그래프를 Local search를 이용하여 서로 비교하는 방법이다.

실험 결과 그래프 모델링을 이용한 방법이 복제된 프로그램을 효과적으로 찾아낸다는 결론을 내릴 수 있었다. 이 연구가 더 심화되어 기존의 알고리즘과 함께 적용된다면 복제 프로그램에 대한 판별이 더욱 효율적으로 실행될 것이라 기대된다.

Keyword : 그래프 모델링, 복제판별, 그래프 매칭

< 목차 >

Ⅰ. 서론

1970년대에 개인 컴퓨터의 보급이 이루어지면서 컴퓨터는 생활에 없어서는 안 될 필수품으로 떠오르게 되었다. 대부분의 사람들이 컴퓨터를 사용하게 되면서 여러 가지 종류의 프로그램들이 만들어지기 시작했다. 그러나 컴퓨터 프로그램 발달과 함께 프로그램의 복제, 위조 등 지적 재산권 침해에 대한 문제가 나타났다. 특히 간단한 기능을 가진 상용 소프트웨어에 자주 발생하는 프로그램 복제는 소프트웨어 산업의 발달에 큰 장애 요소가 된다. 뿐만 아니라 대규모 프로그램의 경우에도 그 일부가 다른 프로그램에서 복제되기도 하여 법적 분쟁이 생기기도 한다. 따라서 이러한 문제를 해결하기 위해서는 프로그램의 복제 여부를 쉽게 판별할 수 있는 방법이 요구된다.

교육 현장에서는 프로그래밍 숙제를 복제하는 현상으로 인해 교수자와 학생이 모두 어려움을 겪고 있다고 한다. 특히, 학생들이 선배들의 숙제를 그대로 베끼거나 약간의 변형을 가해 제출할 수 있는 가능성으로 인해 아주 좋은 실습 숙제임에도 불구하고 반복해서 사용할 수 없는 어려움이 있다고 한다. 이에 과거의 모든 프로그래밍 과제물들을 데이터베이스화하여 이들 간의 복제 여부를 반자동으로 판정할 수 있는 방법이 있다면 조교들에게 과중한 부담을 지우지 않으면서도 공정한 운영을 할 수 있다고 한다.

그러면 프로그램의 복제를 판별하는 기준은 무엇이고, 어떻게 하면 찾아낼 수 있을까? 이 의문이 프로그램 복제 여부의 판별에 대한 연구를 시작하게 된 계기가 되었다.

이전에 사용되었던 방법은 두 프로그램의 특성(함수의 개수, 변수의 개수, 반복문의 개수 등)을 수치화한 후, 그것을 비교하여 유사성의 여부를 결정하는 것이었다.

그러나 본 논문이 제시하는 알고리즘은 프로그램의 소스 코드를 그래프로 모델링한 이후에, 그래프의 동형(Isomorphism) 여부를 계산한다. 변형 작업이 개입된다고 가정하므로 완전한 동형은 기대하기 어렵고 근사 동형 문제로 모델링한다.

프로그램을 그래프로 표현하면 프로그램의 구성 요소와 그들 사이의 관계를 나

타낼 수 있다는 장점이 있다. 또한, 두 프로그램이 일치하지 않더라도 유사도의 계산이 가능하며, 가장 유사한 구조를 가진 함수를 찾는 데도 사용할 수 있다는 특징이 있다.

Ⅱ장에서는 소스 코드가 어떤 방식으로 복제되는가에 대한 분석과 함께 알고리즘의 실행 과정에 대해 서술하였으며, Ⅲ장에서는 복제 여부가 이미 판정된 프로그램들을 대상으로 실험한 결과 및 분석과 함께 알고리즘의 성능과 효과를 제시하였다. Ⅳ장에서는 결론과 함께 앞으로의 발전 가능성에 대해 논하고자 한다.

Ⅱ. 연구 방법

1. 소스 코드 복제의 유형

프로그램을 복제하고자 하는 의도가 있는 사람은 프로그램의 인터페이스나 소스 코드 중 일부를 원본 프로그램에서 복제한 후 수정을 가해 복제 프로그램이 아닌 것처럼 위장하려 한다. 따라서 알고리즘을 작성함에 앞서 프로그램 복제는 어떤 유형으로 일어나는 지를 알아보았다.

조사 결과 복제 과정에서 일어날 수 있는 소스 코드의 변형은 다음과 같은 것들이 있었다.

(1) 주석의 삽입 또는 제거

주석은 프로그램의 실행 과정에 아무런 영향을 미치지 않으므로, 복제된 프로그램은 원본 프로그램의 주석을 바꾸는 경향이 있다. 프로그램을 복제할 수 있는 가장 간단한 방법이다.

(2) 불필요한 변수의 선언

실제 프로그램에서 아무 기능을 하지 않는 변수를 선언하거나, 동일 기능을 하는 변수 두 개를 선언하여 사용하는 유형이 있다. 아래 표 Ⅰ은 이와 같이 필요 없는 변수가 선언된 프로그램을 보여준다.

표 Ⅰ 불필요한 변수가 선언된 프로그램의 예

정상적인 프로그램	int x=1, y=2; print(x+y);
불필요한 변수 z를 선언한 경우	int x=1, y=2, z; print(x+y);

x와 동일 기능을 하는 변수 z를 선언한 경우	int x=1, y=2, z;
	z=x;
	print(y+z);

(3) 함수나 변수의 이름 바꾸기

이는 프로그램에 포함된 함수나 변수의 이름을 다르게 바꾸는 방법으로, 프로그램의 실행 결과와 계산 과정을 변화시키지 않은 채 겉으로 보기에 소스 코드가 달라 보이게 할 수 있다. 프로그램의 구성 요소는 일괄적으로 그 이름을 바꿀 수 있으므로 프로그램 복제를 할 수 있는 손쉬운 방법이라고 할 수 있다.

(4) 불필요한 연산

어떤 변수에 0을 더하거나 1을 곱하는 무의미한 연산을 하거나, 어떤 수를 더한 후 다시 빼는 연산을 하는 등의 유형이 있다. 정상적인 프로그램에서는 이런 연산이 필요치 않으므로 이와 같은 연산이 발견된 경우에는 복제된 프로그램일 가능성이 매우 높다고 할 수 있다.

(5) 명령의 순서 바꾸기

이는 프로그램 실행 결과에 영향을 주지 않는 명령들의 순서를 재배치하는 경우이다. for문을 do~while문으로 바꾸는 등 반복문의 종류를 바꾸는 경우도 있다.

(6) 불필요한 분기

불필요한 분기에는 조건의 만족 여부와 상관없이 같은 내용의 실행을 하는 경우, 프로그램에서 절대로 참이 될 수 없거나 절대로 거짓이 될 수 없는 조

건으로 분기하는 경우 등이 있다. 이와 같은 경우가 발견되었다면 복제된 프로그램일 가능성이 매우 높다.

(7) 함수의 분리 또는 결합

프로그램을 작성할 때에는 그 기능에 따라 함수로 구분 짓는 것이 일반적이다. 그러나 복제 프로그램은 복제임을 감추기 위하여 의도적으로 함수를 쪼개거나 합치는 방법을 사용하기도 한다.

2. 연구에 사용할 그래프의 형태

프로그램의 유사도는 프로그램을 어떤 형태의 그래프로 구성하는가와 유사도를 어떻게 정의하는가에 의하여 결정된다. 여기서는 우선 그래프의 구성 형태에 대해 서술하고, 유사도의 정의 방법은 4절에서 설명할 것이다.

그래프 G는 정점의 집합 V와 간선의 집합 E로 나타내어지므로, V를 정의하는 방법에 따라 그래프의 형태가 달라진다.

V를 정의하는 방법에는 정점을 함수로 하는 방법, 문장으로 하는 방법, 변수(또는 상수 등)로 하는 방법 등이 있다.

첫째, V를 함수의 집합으로 정의(그림 1 참조)하면 간선의 집합 E는 함수간의 호출 관계가 될 것이다. 함수의 개수는 문장이나 변수 등에 비해 비교적 적기 때문에 이를 이용하면 간단하게 그래프를 구성하고 비교할 수 있는 장점이 있다. 그러나 정점이 함수의 복잡한 특성을 반영하기 어렵다는 단점이 있다.

둘째, V를 문장의 집합으로 정의(그림 2 참조)하면 프로그램의 그래프는 순서도와 비슷한 형태가 될 것이다. 이는 V를 함수의 집합으로 정의하는 것보다 프로그램의 형태를 더 구체적으로 나타낼 수 있다. 또한 프로그램의 시간적 순서를 명확하게 나타낼 수 있다는 장점이 있다. 그러나 프로그램의 한 문장이 하나의 정점이 되므로 $n(V)$가 프로그램의 길이에 비례하여 늘어난다는 것이 큰 문제점이 될 수 있다. 왜냐하면 복잡한 그래프일수록 매칭 과정에서 유사도를 계산하기 어렵기 때문이다. 하나의 문장이 의미하는 내용이 정점 하나에 포함되기

에는 너무 많다는 단점도 있다.

셋째, V를 변수나 상수 등으로 정의(그림 3 참조)하는 방법은 위의 두 가지 방법보다 구체적으로 프로그램의 요소들을 표현할 수 있다. 이 방법도 V를 문장의 집합으로 정의하는 것과 마찬가지로 $n(V)$가 프로그램 길이에 비례하여 증가하지만, 대부분의 경우에는 V를 문장의 집합으로 정의하는 것보다 적은 수의 정점만을 필요로 한다. 그러나 프로그램의 시간적 순서를 나타내기 어렵다는 단점이 있다.

본 연구에서는 V를 정의하는 방법으로 세 번째 방법을 사용하였다.

아래 그림 1, 그림 2, 그림 3은 20 이하의 자연수 중 4의 배수의 합을 구하는 프로그램을 위의 세 가지 방법으로 나타낸 것이다.

그림 1 : V를 함수의 집합으로 정의한 경우의 그래프의 예

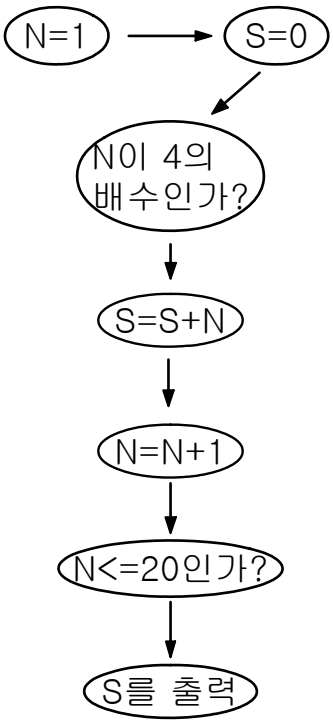

그림 2 : *V*를 문장의 집합으로 정의한 경우의 그래프의 예

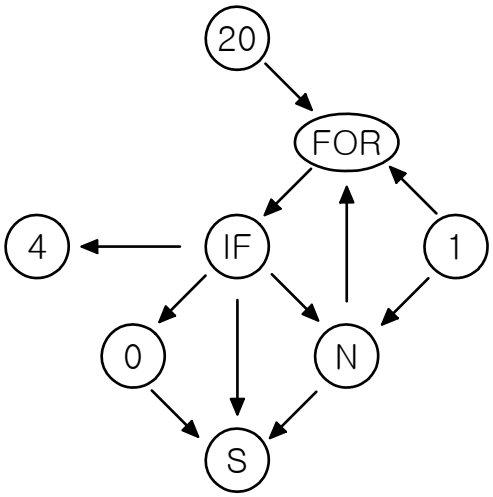

그림 3 : *V*를 변수의 집합으로 정의한 경우의 그래프의 예

3. 소스 코드의 파싱을 통한 그래프의 구성

파싱 프로그램은 대상 프로그램의 소스 코드를 각 요소의 관계에 따라 그래프로 구성하여 컴퓨터가 처리할 수 있는 형태로 바꾸는 역할을 한다.

(1) 변수의 구분

프로그램을 이루고 있는 함수는 그 기능에 필요한 변수나 상수, 반복문, 조건문 등을 내부에 포함하고 있다. 같은 이름의 변수라 할지라도 그것이 포함된 함수에 따라 다른 기능을 할 수도 있다. 따라서 함수를 기준으로 하여 변수들의 집합을 구분하였다. 이렇게 구분된 변수들의 집합은 매칭 과정에서 함수 사이의 유사도를 쉽게 계산할 수 있도록 한다. 아래 그림 4는 함수에 따라 변수를 구분하는 예를 보여준다. 그림 4의 max 함수는 두 개의 변수 x, y와 한 개의 조건문을 가지고 있으며, main 함수는 세 개의 변수 i, m, n과 두 개의 상수 1, -1, 그리고 한 개의 반복문을 가지고 있다.

```
int max(int x, int y)
{
    if(x>y) return x;
    return y;
}

void main()
{
    int i;
    int m=-1;
    for(i=1;i<=n;i++) m=max(m, a[i]);
}
```

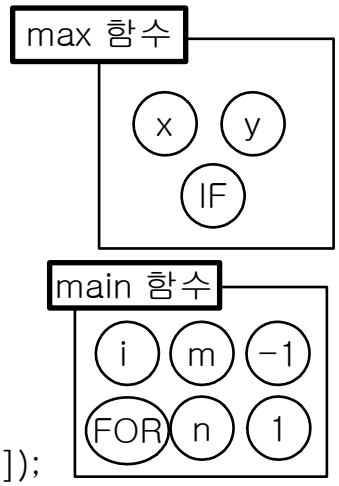

그림 4 : 함수에 따라 변수를 구분한 그림

(2) 그래프의 구성 요소

그래프는 정점과 간선의 두 요소로 구성된다.

먼저, 정점은 프로그램에 포함된 변수, 상수, 조건문, 반복문을 의미한다. 이 때, 각 정점에 그 정점의 특성을 네 가지 종류(변수, 상수, 조건문, 반복문)로 나누어 명시하였다. 이는 이후 매칭 과정에서 두 원소를 대응시킬 때, 적어도 두 원소의 속성이 같아야 대응이 이루어질 수 있도록 하기 위함이다.

다음으로, 간선은 두 정점 사이에 값을 전달하는 등의 관계가 있음을 나타내며, 방향성을 가지고 있다. 간선을 생성할 때 사용한 규칙은 아래와 같다.

첫째, 대입 연산의 경우 우변의 원소에서 좌변의 원소로 향하는 간선을 생성한다.

예를 들어, a=b+c; 라는 명령이 있다고 하자. 이 경우에는 b에서 a로 향하는 간선 및 c에서 a로 향하는 간선을 생성한다(그림 5 참조).

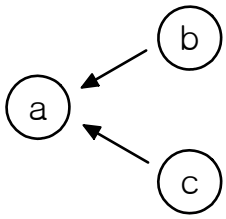

그림 5 : a=b+ c;를 그래프로 나타낸 모습

배열을 참조하는 경우에는 첨자에서 배열 이름으로 향하는 간선을 더 생성한다. 그 예로 a[i]=b+ c;라는 명령이 있다고 하자. 이때에는 b에서 a로 향하는 간선, c에서 a로 향하는 간선 및 배열 첨자 i에서 a로 향하는 간선 등 세 개의 간선이 생성된다(그림 6 참조). 첨자 i가 연산에 직접 참여한 것은 아니지만, 간접적으로 a의 값을 변화시키는 역할을 하기 때문이다.

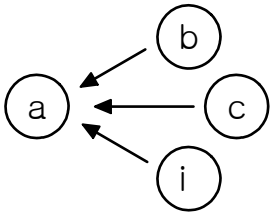

그림 6 : a[i]=b+ c;를 그래프로 나타낸 모습

d[i]=d[i-1]+ i;와 같이 대입 연산의 양변에 같은 이름을 가진 원소가 있을 때에는 그래프에 루프(loop)를 생성한다(그림 7 참조).

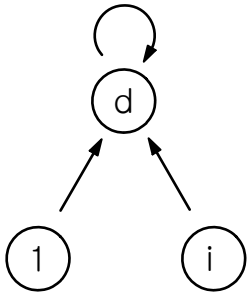

그림 7 : d[i]=d[i-1]+i;를 그래프로 나타낸 모습

둘째, 조건문과 반복문이 있을 경우, 조건문 또는 반복문으로부터 내부에 포함된 모든 요소로 향하는 간선을 생성한다.

예를 들어, 아래와 같은 반복문이 있다고 하자.

```
for(i=1;i<=n;i++)
{
  if(i%4==0) s-=i;
  s+=i;
}
```

위의 경우에는 for문에 포함된 요소인 if, i, n, s, 0, 1, 4 등에는 for문으로부터 나온 간선이 생성된다.

(3) 파싱 과정에서 수행된 소스 코드의 단순화

소스 코드가 전혀 다른 형태인 그래프로 변환되는 것이기 때문에, 이 과정에서 다음과 같이 프로그램 소스 코드를 단순화가 이루어졌다.

첫째, 프로그램의 시간적 순서가 고려되지 않았다. 이는 그래프의 정점을 함수에 포함된 요소로 정의했고, 간선을 요소들 사이의 관계로 나타내기로 정의했기 때문이다.

둘째, 연산의 종류에 대한 정보가 무시되었다. 그래프에 포함된 간선에는 두 정점 사이에 어떠한 관계가 있는지 명시되어 있지 않기 때문에 사칙연산을 비롯하여 나머지 연산, 비트 연산 등에 대한 정보가 모두 그래프에 포함되지 않았다.

셋째, 함수 사이의 호출 관계가 고려되지 않았다. 이는 매칭 과정에서 함수와 함수가 일대일 대응을 하므로, 전체 프로그램의 유사도를 최대화하면서 함

수 사이의 호출 관계도 최대한 고려하는 대응을 찾는 데 어려움이 있기 때문이다.

마지막으로, 배열은 하나의 원소로 취급했다. 배열은 같은 속성을 가진 자료들의 집합이므로 한 개의 원소로 가정해도 프로그램 전체의 구조에는 큰 영향을 미치지 않기 때문이다.

4. 유사도의 정의 방법

(1) 각 함수 사이의 유사도

함수 f에 포함된 요소의 집합을 S_f라 하고, 함수 g에 포함된 요소의 집합을 S_g라 하자. 그리고 S_f의 인접행렬을 W_f라 하고, S_g의 인접행렬을 W_g라 하자. 유사도를 계산하기 위하여 $n(P_f)=n(P_g)$, $P_f \subset S_f$, $P_g \subset S_g$를 만족하는 S_f의 부분집합 P_f와, S_g의 부분집합 P_g를 정한다. 그리고 P_f의 인접행렬을 Adj_f, P_g의 인접행렬을 Adj_g라 한다.

그래프의 형태가 변하지 않도록 Adj_g의 행과 열을 적절히 교체하였을 때(그림 8 참조) Adj_f와 Adj_g에서 일치하는 성분의 개수의 최대값을 M이라고 하고, W_f의 성분의 개수와 W_g의 성분의 개수를 더한 $n(S_f)^2+n(S_g)^2$을 N이라 하자.

이 때, P_f와 P_g로 계산한 함수 f와 함수 g의 유사도 $T_{P_fP_g}$를 $\frac{2M}{N}$으로 정의한다. 함수 f와 함수 g의 유사도 T_{fg}는 모든 P_f와 P_g에 대해 계산된 $T_{P_fP_g}$ 중 최대값으로 한다.

	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	1	0	0	1
4	1	1	0	0

→

	1	2	3	4
1	0	0	1	1
2	1	0	0	1
3	1	1	0	0
4	0	0	1	0

그림 8 : 그래프의 두 번째 정점과 네 번째 정점을 교체했을 때 변경된 인접행렬. 인접행렬이 변경된 이후에도 그래프는 형태는 변하지 않는다.

(2) 두 프로그램의 유사도

프로그램 A에 포함된 함수의 집합을 F_A라 하고, F_A에 포함된 원소의 개수를 n이라 하자. 마찬가지로 프로그램 B에 포함된 함수의 집합을 F_B라 하고, F_B에 포함된 원소의 개수를 m이라고 하자(단, $n \leqq m$이라고 가정한다). (1)에서 정의한 방법을 사용하여 $f \in F_A$, $g \in F_B$인 모든 함수 f와 g에 대해 T_{fg}를 성분으로 하는 새로운 행렬 W를 만든다.

만약 $n < m$이라면, W의 행을 확장하여 $m \times m$ 크기의 정사각행렬로 만들고, 확장된 행에 있는 성분은 0으로 한다. 아래 그림 9는 $n = 3$, $m = 5$일 때 행렬 W를 확장하는 과정을 보여준다.

$\frac{212}{369}$	$\frac{100}{289}$	$\frac{214}{450}$	$\frac{394}{586}$	$\frac{254}{450}$
$\frac{166}{288}$	$\frac{144}{208}$	$\frac{202}{369}$	$\frac{244}{505}$	$\frac{110}{369}$
$\frac{172}{369}$	$\frac{118}{289}$	$\frac{302}{450}$	$\frac{322}{586}$	$\frac{408}{450}$

→

$\frac{212}{369}$	$\frac{100}{289}$	$\frac{214}{450}$	$\frac{394}{586}$	$\frac{254}{450}$
$\frac{166}{288}$	$\frac{144}{208}$	$\frac{202}{369}$	$\frac{244}{505}$	$\frac{110}{369}$
$\frac{172}{369}$	$\frac{118}{289}$	$\frac{302}{450}$	$\frac{322}{586}$	$\frac{408}{450}$
0	0	0	0	0
0	0	0	0	0

그림 9 : $n=3$, $m=5$일 때 함수 유사도 행렬 W의 확장

만약 $n=m$이라면 행렬 W에 변화를 주지 않고 그대로 사용한다.

(1)에서의 유사도 정의에서 W_{ij}의 분모는 두 함수의 인접행렬에 포함된 성분의 개수의 합과 같으므로, 각 행과 열에서 하나씩의 성분만 선택했을 때 분모의 합은 일정하다.

따라서 분모를 무시하고 분자만으로 유사도를 계산할 수 있다. W의 각 행과 열에서 하나씩의 성분만 선택했을 때, 그 분자의 합의 최대값을 프로그램 A와 프로그램 B의 유사도로 정의한다.

5. 그래프의 매칭을 통한 유사도의 결정

프로그램의 전체 유사도를 결정하기 위한 매칭은 두 단계로 나누어 수행하였다.

첫 번째 단계는 두 프로그램을 함수 단위로 분리하여 서로 비교하는 것으로, 전체 프로그램의 유사도를 구하기 위해 필요한 유사도 행렬을 만들었다.

두 번째 단계는 첫 번째 단계에서 만들어진 행렬을 사용하여 전체 프로그램의 유사도를 구하는 것으로, 최대 매칭 알고리즘을 사용하여 인접행렬의 일치하는 성분의 개수를 최대화하였다.

(1) 각 함수 사이의 유사도 계산

첫 번째 단계는 프로그램에 포함된 함수 사이의 유사도를 계산하는 것이다. 프로그램은 특정 기능을 수행하는 함수 단위로 구성되어 있기 때문에, 이 함수들 사이의 유사도를 계산한 후에, 전체 프로그램을 함수 단위로 대응시킬 수 있다. 이 때 각 함수의 그래프의 인접행렬 중 얼마나 많은 부분이 서로 일치하는가를 유사도를 계산하는 기준으로 사용하였다.

그래프의 동형 여부를 판별하는 Graph isomorphism 문제는 현재까지의 기술로는 다항시간 해결 알고리즘이 존재하지 않는 NP-Hard 문제이다. 그렇기

때문에 그래프가 얼마나 닮았는지를 판별하기 위해서 두 그래프의 정점들에 대해 최적의 일대일 대응을 찾는 Local search를 사용하였다.

Local search를 사용하기 위해 첫 번째 함수의 요소에서 연결 가능한 두 번째 함수의 요소를 임의로 지정하여 초기해를 설정하였다. 이후 대응된 원소들을 바꿔 가면서, 일치하는 성분의 개수를 최대화하는 방향으로 Local search를 진행하였다.

일반적인 그래프의 동형 문제와 비교해 볼 때, 이 문제는 정점들이 가지고 있는 특징을 이용할 수 있어 탐색 공간을 상당히 줄일 수 있다. 원소 사이의 대응은 각 원소의 기능을 기준으로 하여, 같은 기능을 가진 원소 사이에만 이루어지도록 하였다. 이는 함수 사이의 유사도가 더욱 현실적인 수치를 갖도록 하기 위함이다.

예를 들어, 변수 x와 상수 1은 프로그램 상에서 하는 역할이 다르다. 그렇기 때문에 x와 1의 대응이 아무리 인접행렬의 일치도를 증가시킨다 하더라도 이를 서로 대응시키는 것은 바람직하지 않은 결과를 가져올 수 있다. 그리하여 변수는 변수끼리, 상수는 상수끼리 대응할 수 있도록 하였고, 마찬가지로 반복문은 반복문끼리, 조건문은 조건문끼리 대응시켰다.

(2) 두 프로그램의 유사도 계산

두 번째 단계는 함수끼리의 대응을 통한 두 프로그램의 유사도를 구하는 것이다. 이는 그래프의 동형 문제를 근사적으로 해결하기 위한 방법이다. 만약 두 그래프의 동형 여부를 조사하고자 한다면 그래프의 각 정점을 일대일 대응시켜 그래프가 완전히 일치하는지를 확인하는 방법을 사용할 수 있다. 그러나 프로그램이 복제되는 과정에서 그래프의 형태가 변했을 것으로 가정하기 때문에 두 그래프를 최대한 많은 부분에서 일치하는 대응을 구하는 것으로 동형 문제를 변형시켰다.

매칭 단계에서는 함수 사이의 대응을 고려하여 유사도를 최대로 하는 대응을 찾는다. 이 때 첫 번째 단계에서 구한 각 함수 사이의 유사도 행렬을 이용하였다. 가능한 모든 함수의 대응을 고려하는 알고리즘은 시간복잡도가 $O(n!)$

으로, 프로그램에 포함된 함수가 15개만 넘어도 현실적인 시간 내에 답이 나오지 않는다는 단점이 있다. 따라서 짧은 시간 안에 답을 낼 수 있는 알고리즘이 필요하였다.

시간문제를 해결하기 위해 사용한 알고리즘은 최대 매칭 알고리즘이다. 최대 매칭 알고리즘은 다항 시간에 해결 가능하며, 시간복잡도는 평균 $O(n^3)$으로, $O(n!)$ 알고리즘보다 훨씬 효율적이다.[1, 2]

n이 자연수이고, 행렬 $W=(w_{ij})$와 $X=(x_{ij})$가 $n\times n$ 행렬이라고 하자. 단, W는 위에서 정의한 방법으로 만든 함수 유사도 행렬이다.

최대 매칭 알고리즘은 $Z=\sum_{i=1}^{n}\sum_{j=1}^{n}W_{ij}x_{ij}$을 최대화하는 알고리즘으로, $\sum_{i=1}^{n}x_{ij}=1(j=1,\cdots,n)$, $\sum_{j=1}^{n}x_{ij}=1(i=1,\cdots,n)$, $x_{ij}\geqq 0$을 만족하는 행렬 X를 구하는 알고리즘이다.[3]

즉, 위의 조건을 만족하기만 하면 어떠한 대응도 상관없다. 따라서 함수 f와 함수 g가 유사도가 높다고 하더라도, f와 g를 대응시키지 않는 다른 방법으로 전체 프로그램의 유사도가 더 높아질 수 있다면, f와 g는 대응되지 않는다.

Ⅲ. 실험결과 및 분석

1. 복제 여부를 알고 있는 샘플에 대한 유사도 계산

알고리즘의 정확성을 평가하기 위해 복제 여부가 이미 판정된 프로그램들을 대상으로 실행해 보았다.

첫 번째 샘플 sort1.java와 sort2.java는 다음과 같은 결과를 나타내었다. 이때 함수 사이의 매칭에 각각 0.3초의 시간을 사용하였다.

표 Ⅱ sort1.java와 sort2.java의 판정 결과

샘플 이름	sort1.java	sort2.java
샘플의 크기	16.4KB	9.99KB
함수의 개수	20개	21개
정점의 개수	299개	222개
계산된 유사도	4212/12897 = 32.66%	

위의 샘플은 실제로 복제가 판정된 프로그램임에도 불구하고 유사도가 32.66%로 낮게 계산되었다. 그 이유는 변수와 함수의 이름이 수정되었고, 명령의 순서도 바뀌었으며, 함수가 분리되고 결합되는 과정을 통해 변경되었기 때문인 것으로 생각된다(Level 5~6에 해당하는 복제이다[4]).

두 번째 샘플인 sort3.java와 sort4.java는 아래와 같은 결과를 나타내었다. 함수 사이의 매칭에는 역시 0.3초를 사용하였다.

표 Ⅲ sort3.java와 sort4.java의 판정 결과

샘플 이름	sort3.java	sort4.java
샘플의 크기	34.2KB	13.8KB
함수의 개수	20개	15개
정점의 개수	380개	323개
계산된 유사도	19266/22137 = 87.03%	

이 샘플 역시 복제되었음이 이미 판정된 것으로, 함수의 개수가 5개나 차이가 나지만 많은 수정이 가해지지 않았기 때문에 유사도가 87.03%로 높게 계산되었다. 특히 대응된 15쌍의 함수 중 13쌍의 함수가 서로 같은 기능을 하는 것으로 나타났다.

세 번째 샘플은 avltree1.java, avltree2.java, avltree3.java로, 세 개의 프로그램이 모두 복제된 프로그램으로 이미 판정된 것이었다. 실행 결과는 아래 표와 같았다. 함수 사이의 유사도 계산 시 사용한 시간은 한 쌍의 함수 당 0.2초였다.

표 Ⅳ avltree1.java, avltree2.java, avltree3.java의 판정 결과

샘플 이름	avltree1.java	avltree2.java	avltree3.java
샘플의 크기	4.18KB	6.56KB	8.24KB
함수의 개수	7개	7개	7개
정점의 개수	108개	108개	108개
계산된 유사도	avltree1, avltree2 : 4028/4028 = 100.00% avltree1, avltree3 : 4028/4028 = 100.00% avltree2, avltree3 : 4028/4028 = 100.00%		

세 번째 샘플은 함수의 개수가 모두 7개이고, 그래프에 포함된 정점의 개수도 108개로 모두 같았다. 그러나 샘플의 크기는 서로 다른데, 그 중 avltree1.java와 avltree3.java는 그 크기에 있어서 약 4.06KB의 차이가 있었다. 이는 샘플에 포함된 주석 때문인데, 그럼에도 불구하고 유사도는 100%로 계산되었다. 이로부터 프로그램의 주석은 그래프를 만드는 과정에 전혀 영향을 미치지 않음을

알 수 있었다.

네 번째 샘플은 avltree1.java와 avltree6.java로, 양자는 같은 기능을 하는 프로그램이지만 실제로는 복제되지 않은 프로그램이었다. 실행 결과는 아래와 같았다.

표 V avltree1.java와 avltree6.java의 판정 결과

샘플 이름	avltree1.java	avltree6.java
샘플의 크기	4.18KB	10.6KB
함수의 개수	7개	10개
정점의 개수	108개	64개
계산된 유사도	1382/3178 = 43.49%	

이 결과로부터, 기능이 같은 프로그램이라도 내부 구조가 다른 경우에는 그래프의 형태가 서로 다르며, 그로부터 계산된 유사도 역시 낮게 나타나는 것을 알 수 있었다.

다섯 번째 샘플 matching1.java와 matching3.java는 복제되지 않은 두 프로그램이지만 유사도가 비교적 높게 판정된 경우이다. 실행 결과는 아래 표와 같았다.

표 VI matching1.java와 matching3.java의 판정 결과

샘플 이름	matching1.java	matching3.java
샘플의 크기	5.01KB	5.45KB
함수의 개수	13개	3개
정점의 개수	147개	94개
계산된 유사도	8630/11983 = 72.02%	

두 샘플의 함수의 개수는 10개나 차이가 나지만 72.02%의 높은 유사도를 보

였다. 이는 두 프로그램의 대부분의 정점들이 main 함수에 집중되어있었기 때문이다. main 함수끼리의 유사도는 $\frac{8306}{10756} \approx 77.2\%$이고, 전체 대응된 인접행렬의 성분들 중 main 함수에서 대응된 성분의 비율은 $\frac{8306}{8630} \approx 96.2\%$로, 대부분의 성분들이 모두 main 함수에서 대응되었음을 알 수 있었다. 이로부터 복제되지 않은 프로그램이라도 특정 함수의 규모가 매우 크고 이들 사이의 유사도가 높을 때, 전체 유사도가 높게 계산될 수 있다는 결론을 얻었다.

2. 복제 프로그램을 판정할 수 있는 기준

(1) 복제된 프로그램에 대한 실험 결과 분석

전체 자료들 중에서 복제된 프로그램은 총 66쌍이었으며, 각각의 쌍에 대해 계산된 유사도는 아래와 같았다. 함수 간 유사도 계산 시간은 0.03초였다.

표 Ⅶ 복제된 프로그램 66쌍에 대한 실험 결과

계산된 유사도(%)	샘플의 개수(쌍)
~10	0
10~20	0
20~30	0
30~40	1
40~50	0
50~60	2
60~70	3
70~80	6
80~90	5
90~	49
평균 유사도	91.93%
표준편차	13.91

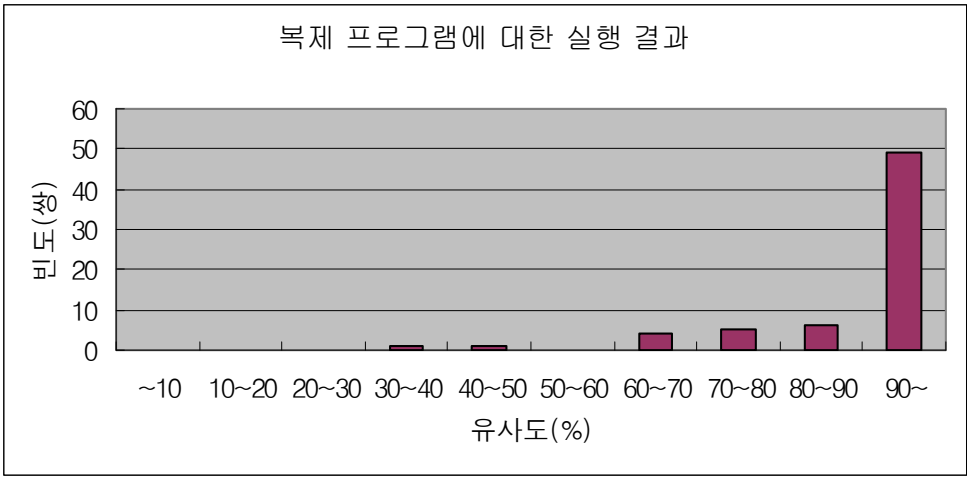

그림 10 : 복제된 프로그램 66쌍에 대한 실험 결과 차트

실험 결과 복제 프로그램의 유사도는 평균 91.93%로 계산되었고, 가장 낮게 판정된 유사도는 32.66%였다. 함수 사이의 유사도를 계산하는데 비교적 짧은 시간인 0.03초만을 사용했지만, 전체 복제 프로그램 샘플 중 81.81%인 54쌍이 80% 이상의 유사도를 보였다.

비교적 낮은 유사도를 보였던 프로그램들은 프로그램 복제 스펙트럼 중 높은 단계의 복제인 Level 5(program statements의 변형)와 Level 6(control logic의 변형)에 속하는 것들이었다.[4] 특히 함수의 결합과 분리를 거쳐 복제된 프로그램의 경우에는 함수 사이의 유사도도 낮게 계산되었으며, 전체 프로그램의 유사도도 낮게 계산되었다.

90% 이상의 높은 유사도를 보인 샘플은 대부분 가장 간단한 복제인 Level 1(comments의 변형)에 해당하는 경우였고, 드물게 Level 2(identifiers의 변형)와 Level 3(variable position의 변형)에 해당하는 경우가 발견되었다. 특히 Level 1에 해당하는 샘플은 대다수가 100%의 유사도를 보였다. 이는 파싱 단계에서 소스 코드의 주석은 고려되지 않았기 때문이다.

(2) 복제되지 않은 프로그램에 대한 실험 결과 분석

이번에는 전체 자료들 중에서 복제된 프로그램을 제외한 나머지 프로그램들

로 실험을 해 보았다. 실험에 사용한 샘플의 개수는 총 1026쌍이며, 샘플들에 대해 유사도를 계산하여 다음과 같은 결과를 얻을 수 있었다. 함수 간 유사도 계산 시간은 0.03초였다.

표 Ⅷ 복제되지 않은 프로그램 1026쌍에 대한 실험 결과

계산된 유사도(%)	샘플의 개수(쌍)
~10	18
10~20	46
20~30	111
30~40	167
40~50	241
50~60	246
60~70	167
70~80	27
80~90	3
90~	0
평균 유사도	45.73%
표준편차	15.43

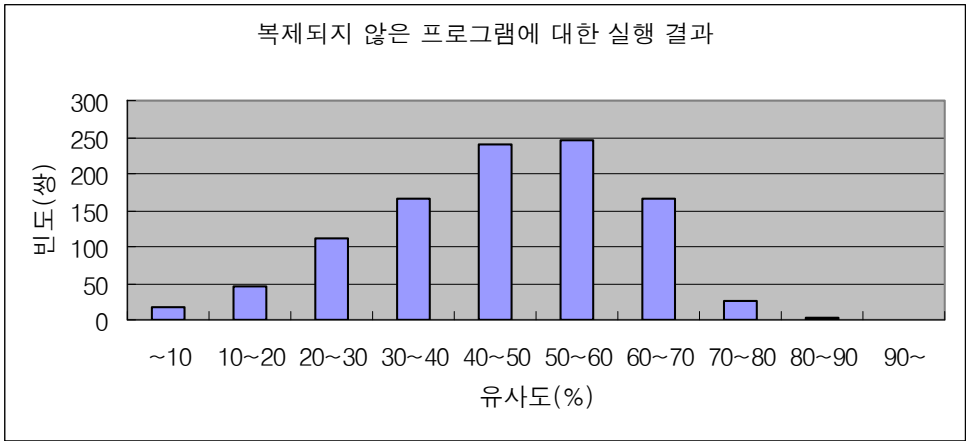

그림 11 : 복제되지 않은 프로그램 1026쌍에 대한 실험 결과 차트

실험 결과 가장 높게 판정된 유사도는 86.24%였으며, 가장 낮게 판정된 유사도는 1.68%였고, 평균 유사도는 45.73%였다. 복제되지 않은 프로그램 쌍 중에서 90% 이상의 유사도를 가진 쌍은 하나도 없었다.

위의 실행 결과를 정규분포로 가정한다면, 99%의 신뢰도로 복제 프로그램을 판단할 수 있는 기준은 $m+2.33\times\sigma=81.68\%$이다. 즉, 81.68% 이상의 유사도를 가진 프로그램이 복제 프로그램일 확률은 99%이다.

(3) 유사도 분포와 복제 판정선

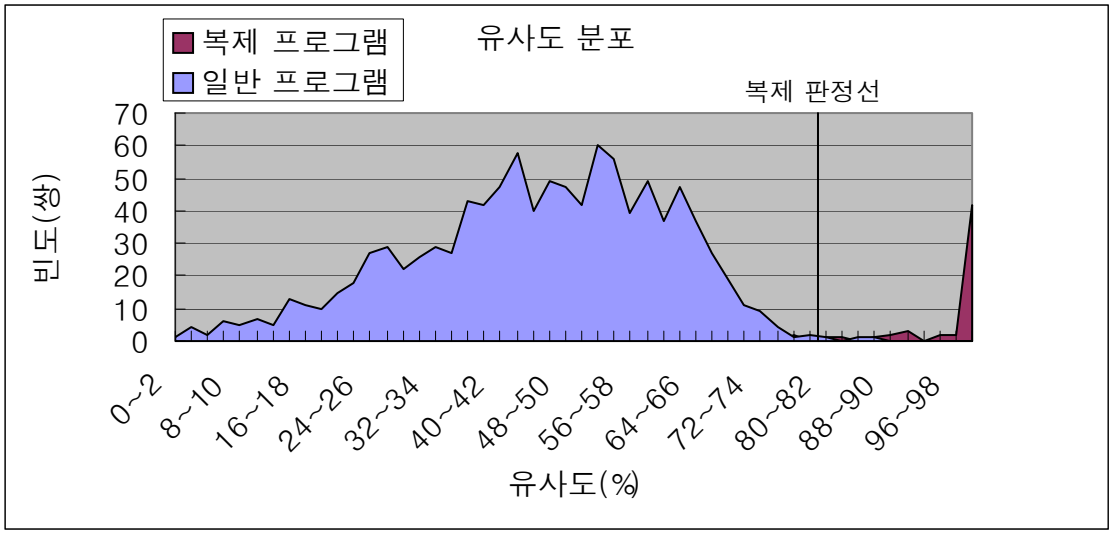

그림 12 : 유사도 분포 그림과 복제 판정선(81.68%)

그림 12는 복제가 아닌 프로그램들 쌍과 복제 프로그램들 쌍의 계산된 유사도 분포를 그려본 것이다. 99%의 신뢰도로 복제 프로그램을 판단할 수 있는 기준인 81.68% 복제 판정선을 함께 표시하였다. 복제 판정선을 기준으로 복제 프로그램들이 오른쪽으로 현저히 편중되어 있고, 일반 프로그램들은 왼쪽으로 현저히 편중되어 있음을 알 수 있다.

위에서 복제 판정선을 제시하였지만 대부분의 상황에서 최종적인 판단은 사람이 하게 되므로 이 복제 판정선은 판정 작업의 정밀도와 판정에 사용할 시

간을 고려하여 설정할 수 있다.

예를 들어, 복제 판정선을 오른쪽으로 이동한다면 판정에 소요되는 시간은 줄어들지만 판정 기준은 비교적 느슨해진다. 반면에 복제 판정선을 왼쪽으로 이동한다면 판정 기준은 더 엄밀해지지만 판정을 위한 확인 작업에 더 많은 시간이 소요된다. 따라서 복제 판정선은 판정의 정밀도 및 소요 시간에 맞추어 정하면 된다.

3. 여러 가지 복제 유형에 대한 알고리즘의 성능 분석

알고리즘이 계산한 유사도를 여러 가지 복제 유형별로 분석한 결과 아래와 같은 결론을 얻을 수 있었다.

(1) 주석의 삽입 또는 제거

주석은 프로그램의 실행에 아무런 영향을 미치지 않고, 그래프의 구성 요소로 포함되지도 않는다. 따라서 주석이 삽입 또는 제거된 경우에는 거의 완벽하게 복제 프로그램을 찾아낼 수 있었다.

(2) 불필요한 변수의 선언

파싱 과정에서 그래프를 구성할 때, 프로그램에서 한 번이라도 연산에 참여한 요소만이 그래프에 추가되었다. 그렇기 때문에, 아무 기능을 하지 않는 변수는 그래프의 구성 요소로 포함되지 않아, 복제 여부의 판별이 비교적 용이하였다. 하지만 동일 기능을 가진 두 개의 변수가 선언된 경우에는 판별이 어려웠다. 이는 그 변수가 불필요함에도 불구하고 그래프의 구성 요소로 추가되었기 때문이다.

(3) 함수나 변수의 이름 바꾸기

매칭 단계에서 함수나 변수의 이름의 유사성에 대한 고려는 전혀 이루어지지 않고, 그래프의 형태만을 기준으로 하였다. 따라서 함수나 변수의 이름만을

바꾼 복제 프로그램의 경우에는 거의 완벽하게 판별할 수 있었다.

(4) 불필요한 연산

어떤 연산이 실제로 필요한지 아닌지를 판단하는 것은 쉽지 않다. 따라서 그래프의 구성 요소에서 제외되지 않았다. 그 결과 불필요한 연산이 포함된 복제 프로그램은 판정이 쉽지 않았다. 그러나 0을 더하거나 1을 곱하는 것과 같이 간단한 경우에 대해서는 파싱 프로그램의 성능이 향상되면 쉽게 판별할 수 있을 것이라 예상된다.

(5) 명령의 순서 바꾸기

변수(또는 상수)의 집합으로 그래프의 정점을 만들었기 때문에, 생성된 그래프가 명령들의 시간 순서를 거의 반영하지 않게 된다. 따라서 명령들의 시간 순서가 바뀐 복제 프로그램은 잘 찾아냈다. 다만, 복제되지 않은 프로그램도 명령의 순서를 바꾸어서 그 형태가 원본 프로그램과 유사할 경우 유사도가 높게 계산되기도 했다.

(6) 불필요한 분기

불필요한 연산을 검사하는 것과 마찬가지로, 파싱 과정에서 어떤 조건문이 필요한지 아닌지를 판단하는 것이 쉽지 않다. 따라서 불필요한 분기는 그래프의 구성 요소에서 제외되지 않았으므로, 이를 판정하는 것은 어려웠다. 그러나 조건의 만족 여부와 상관없이 같은 내용을 실행하도록 하는 조건문의 경우, 파싱 프로그램을 더욱 강화시킨다면 판별이 가능할 것으로 예상된다.

(7) 함수의 분리 또는 결합

이러한 복제 방법은 프로그램의 구조에 대한 이해가 필요하다. 왜냐하면 프로그램의 함수를 분리하거나 결합하는 과정에서 논리적으로 오류가 없어야 하기 때문이다. 매칭 단계에서는 함수 단위로 프로그램을 비교하기 때문에 함수가 분리되었거나 결합된 경우에는 복제 여부를 판별하기 어려웠다.

Ⅳ. 결론

본 논문에서는 두 프로그램의 복제 여부를 판별하기 위하여, 프로그램의 구조를 최대한 반영하는 새로운 알고리즘, 즉 그래프 모델링을 이용한 알고리즘을 제시하였다. 이 알고리즘은 프로그램의 소스 코드를 분석하여 프로그램에 포함된 요소들 사이의 관계를 그래프로 나타낸 이후, 그래프의 매칭을 통하여 유사도를 계산한다. 프로그램을 그래프로 표현하기 때문에 구성 요소와 그들 사이의 관계를 쉽게 알 수 있다는 장점이 있다.

연구 결과 이 알고리즘을 사용하여 81.68%보다 더 높은 유사도를 가지는 프로그램 쌍은 99%의 정확도로 복제 프로그램임을 판정할 수 있다는 점을 밝혔다. 그리고 그래프 모델링을 이용한 매칭 알고리즘이 간단하게 이루어진 복제에 대해 좋은 성능을 가진다는 것을 보였으며, 비교적 복잡한 복제 프로그램에 대해서도 높은 유사도로 판정을 해 준다는 것을 보였다.

앞으로 컴퓨터 프로그램은 그 수가 더욱 많아져 모든 분야에 걸쳐 적용될 것이고, 그와 함께 프로그램의 복제에 대한 문제도 더욱 심각해질 것이다. 본 연구는 이러한 프로그램의 복제를 판별할 수 있는 새로운 방법을 제시하였다는 데 그 의의가 있다.

이번 연구가 더욱 심화되면 큰 프로젝트에 대해서도 복제의 판별이 가능해질 것이며, 전체 프로그램 중 일부에 대해서만 이 알고리즘을 사용하여 부분적으로 복제된 소스 코드를 판별하는 것도 시도할 수 있을 것이다. 특히, 파싱 프로그램의 성능을 향상시키면 낮은 유사도로 판정되었던 복제 프로그램의 경우에도 더욱 높은 유사도로 판정이 가능할 것으로 기대된다. 또한 본 논문의 그래프 매칭 알고리즘이 이미 프로그램 복제판별에 이용되고 있는 스트링 매칭 또는 신경망 네트워크 알고리즘 등과 함께 적용된다면 더욱 정확한 결과를 낼 수 있을 것이다.

Ⅴ. 참고 문헌

[1] H. Kuhn, "The Hungarian Method for the Assignment Problem", Naval Res Logistics Quarterly, Vol. 2, pp. 83-97, 1955.

[2] J. Munkres, "Algorithms for the Assignment and Transportation Problems", Journal of the Society of Industrial and Applied Mathematics, Vol. 5, No. 1, pp. 32-38, 1957.

[3] Katta G. Murty, "An Algorithm for Ranking all the Assignments in Order of Increasing Cost", Operations Research, Vol. 16, No. 3, pp. 682-687, 1968. 5.

[4] Alan Parker, James O. Hamblen, "Computer algorithms for plagiarism detection", Education, IEEE Transactions on, Volume 32, Issue 2, pp. 94-99, 1989. 5.

[5] Seung-jin Yang, Yong-Geon Kim, Yung-Keun Kwon, Byung-Ro Moon, "Assignment Copy Detection Using Neuro-Genetic Hybrids", Genetic and Evolutionary Computation Conference, pp. 2426-2427, 2003.

[6] Jean Mayrand, Claude Leblanc, Ettore M. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", Software Maintenance 1996, Proceedings., International Conference on, pp. 243-253, 1996. 11.