제어 흐름 그래프 매칭 기반 자바 버스마크

(A Java Birthmark based on Control Flow Graph Matching)

박 희 완 † 임 현 일 ††
(Heewan Park) (Hyun-il Lim)
최 석 우 ††† 한 태 숙 ††††
(Seokwoo Choi) (Taisook Han)

요 약 소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 내재된 특징이다. 본 논문에서는 제어 흐름 그래프 매칭 기반의 새로운 자바 버스마크를 제안한다. 제어 흐름 그래프 매칭은 노드 매칭과 에지 매칭으로 이루어진다. 두 제어 흐름 그래프의 노드 유사도와 에지 유사도를 구하기 위해서 서열 정렬 알고리즘과 최단거리 알고리즘을 각각 사용하였다. 제안된 버스마크를 평가하기 위해서 다양한 알고리즘을 구현한 자바 프로그램을 대상으로 실험을 하였다. 제안된 버스마크는 이 실험에서 높은 신뢰도와 강인도 뿐만 아니라 빠른 실행 성능을 보여주었다.

키워드 : 소프트웨어 버스마크, 자바 바이트코드, 제어 흐름 그래프, 그래프 매칭

Abstract A software birthmark is inherent characteristics that can be used to identify a program. In this paper, we propose a new Java birthmark based on control flow graph (CFG) matching. The CFG matching consists of node matching and edge matching. To get similarities of nodes and edges of two CFGs, we apply a sequence alignment algorithm and a shortest path algorithm, respectively. To evaluate the proposed birthmark, we perform experiments on Java programs that implement various algorithms. In the experiments, the proposed birthmark shows not only high credibility and resilience but also fast runtime performance.

Key words : Software Birthmark, Java Bytecode, Control Flow Graph, Graph Matching

· 이 논문은 정부(교육과학기술부)의 재원으로 한국과학재단의 지원(No. R01-2008-000-11856-0)과 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2009-C1090-0902-0020)
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '제어 흐름 그래프 매칭 기반 자바 버스마크'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : KAIST 전산학전공
hwpark@pllab.kaist.ac.kr
†† 비 회 원 : KAIST 전산학전공
hilim@pllab.kaist.ac.kr
††† 정 회 원 : KAIST 전산학전공 위촉연구원
swchoi@pllab.kaist.ac.kr
†††† 종신회원 : KAIST 전산학전공 교수
han@cs.kaist.ac.kr

논문접수 : 2009년 8월 13일
심사완료 : 2009년 9월 23일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제11호(2009.11)

1. 서 론

라이선스 위반이나 프로그램 도용에 대한 연구는 소스 표절 검사 기법에서 많이 다루어지고 있다[1]. 만일 도용이 의심되는 프로그램의 소스 코드를 얻을 수 있다면 이 방법이 가장 효과적으로 사용될 수 있다. 그러나 일반적으로 프로그램은 컴파일 되어 배포되기 때문에 소스 코드를 얻을 수 없는 상황에서도 적용할 수 있는 도용 탐지 기법이 필요하다.

소프트웨어 버스마크(Software Birthmark)[2-4]는 프로그램 바이너리로부터 직접 추출이 가능하며 프로그램을 식별하는데 사용될 수 있는 내재된 특징을 말한다. 어떤 두 프로그램으로부터 동일한 버스마크가 추출되었다면 두 프로그램은 서로 도용 관계가 있다고 의심할 수 있다.

본 논문에서는 프로그램의 제어 흐름 그래프(Control Flow Graph) 기반의 새로운 자바 버스마크를 제안하고 기존의 버스마크와의 비교 실험을 통해서 평가한다.

본 논문의 구성은 다음과 같다. 2장에서는 버스마크에 대한 기존 연구에 대해서 설명한다. 3장에서는 소프트웨어 버스마크의 정의를 설명한다. 4장에서는 본 논문에서 제안하는 버스마킹 시스템을 설명한다. 5장에서는 실험 및 평가를 하고, 6장에서 결론을 맺는다.

2. 관련 연구

Tamada는 자바 클래스 파일에 대한 정적 버스마크 기법[2]을 제안하였다. 이 기법은 클래스 상속 구조와 같은 구조적인 특징 위주로 버스마크를 구성하였다. 따라서 다양한 프로그램 변환에 대해서도 버스마크가 쉽게 손상되지 않는다는 장점이 있다. 그러나 많은 클래스가 서로 비슷한 상속 구조를 갖기도 하기 때문에 서로 다른 두 프로그램을 구별하는 신뢰도가 낮다는 단점이 있다.

Myles는 k-gram 버스마크[3]를 제안하였다. 여기서 k-gram이란 연속된 k개의 JVM 명령어 시퀀스 집합을 의미한다. k-gram 버스마크는 JVM 명령어 간 유사도를 비교하기 때문에 서로 다른 프로그램을 구별하는 능

력은 뛰어나지만 난독화를 할 경우에는 JVM 명령어가 유사한 기능을 수행하는 다른 명령어로 대치되거나 명령어 순서가 바뀌는 경우가 생기기 때문에 버스마크가 쉽게 손상되는 단점이 있다.

프로그램의 실행 트레이스를 정적으로 추출하는 자바 명령어 트레이스 버스마크[4]가 제안되기도 했다. 이 방법은 기존의 k-gram 버스마크가 제어 흐름을 고려하지 않는다는 단점을 보완하기 위해서 자바의 제어 흐름을 분석하여 JVM 명령어의 트레이스를 추출한다. 따라서 프로그램 변환에 대해서 강인하지만 분기문의 개수에 대해서 기하급수적으로 트레이스의 개수가 증가하기 때문에 확장성이 떨어진다는 단점이 있다.

3. 소프트웨어 버스마크

Tamada와 Myles는 소프트웨어 버스마크를 다음과 같이 정의하였다[2-3].

정의 1 (버스마크)

프로그램 p와 q에 대한 함수 f가 다음 조건을 만족할 때, $f(p)$를 프로그램 p의 버스마크라고 한다.

조건 1. $f(p)$는 부가적인 코드의 삽입 없이 p 자신으로부터 얻는다.

조건 2. 프로그램 p와 q가 서로 코드 도용 관계에 있다면 $f(p) = f(q)$ 이다.

다음 속성은 버스마크가 만족시켜야 하는 성질이다.

속성 1 (신뢰성: Credibility)

비록 두 프로그램 p 와 q 가 서로 같은 기능을 하더라도, p 와 q 가 서로 소스 코드를 공유하지 않고 독자적으로 개발되었다면, $f(p) \neq f(q)$ 을 만족해야 한다.

속성 2 (강인성: Resilience)

프로그램 p' 이 프로그램 p 로부터 프로그램 최적화나 난독화 기법과 같이 프로그램의 의미를 바꾸지 않는 변환 기법에 의해서 변환되었다고 할 때, $f(p) = f(p')$ 을 만족해야 한다.

4. 제어 흐름 그래프 매칭 기반 버스마크

4.1 제어 흐름 그래프 매칭 기반 버스마킹 시스템

제어 흐름 그래프 매칭 기반 버스마킹 시스템의 전체 구조는 그림 1과 같다. 버스마킹 시스템은 두개의 자바 클래스를 입력으로 받는다. 그래프 매칭은 메소드 단위로 이루어지기 때문에 두 클래스에 포함된 모든 메소드들의 조합에 대한 비교가 이루어진다. 먼저 비교 대상 메소드로부터 제어 흐름 그래프를 생성한다. 그리고 노드 매칭과 에지 매칭을 통해서 노드와 에지의 유사도를 구한다. 이 결과를 이용해서 메소드 유사도와 클래스 유사도를 계산한다.

지금까지 잘 알려진 그래프 비교 알고리즘[5]에는 그래프 동형(Isomorphism), 편집 거리(Edit distance) 계산, 최대 공통 부분 그래프(Maximum common subgraph) 생성, 통계학적 유사도(Statistical Similarity) 비교 방법 등이 있다.

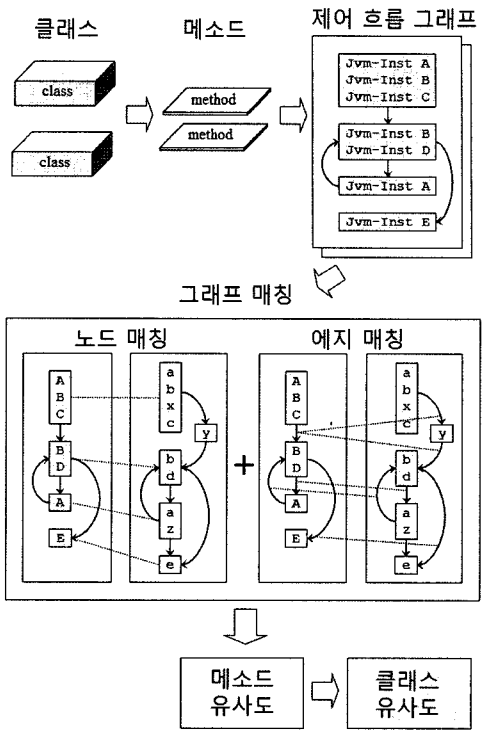

그림 1 그래프 매칭 기반 버스마킹 시스템

프로그램의 고유한 제어 흐름의 특성을 내포하고 있는 제어 흐름 그래프를 버스마크의 용도로 사용하기 위해서는 너무 엄밀하게 두 그래프를 비교하면 안 된다. 왜냐하면 지금까지 알려진 다양한 제어 흐름 난독화 기법을 적용하면 그래프가 변경될 수 있기 때문이다. 또한 너무 함축된 통계 정보를 이용해서도 안 된다. 서로 다른 알고리즘을 표현하고 있는 제어 흐름 그래프를 구별할 정도의 신뢰도가 보장되어야 하기 때문이다. 즉, 제어 흐름 그래프를 버스마크로 사용하기 위해서는 신뢰도와 강인도 두 가지 요소를 모두 충족시키는 그래프 비교 방법이 필요하다. 본 논문에서는 노드 매칭과 에지 매칭을 통한 새로운 그래프 유사도 비교 방법을 제안한다.

4.2 제어 흐름 그래프 매칭 알고리즘

제어 흐름 그래프 매칭은 크게 두 단계로 나뉘어진다. 먼저 비교 대상인 두 제어 흐름 그래프의 노드들에 대한 매칭을 수행하고 그 다음 에지에 대해서 매칭을 수행한다.

4.2.1 노드 매칭(Node Matching)

노드 매칭을 위해서 본 논문에서는 제어 흐름 그래프를 구성하는 노드를 다음과 같이 두 가지로 구분하여 2

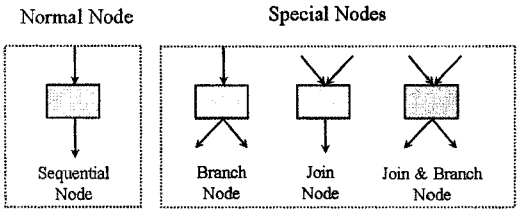

그림 2 제어 흐름 그래프를 구성하는 노드의 종류

단계 노드 매칭을 수행한다.

노드는 그림 2와 같이 일반(Normal) 노드와 특별(Special) 노드로 구분하였다. 일반 노드는 진입 에지와 진출 에지가 각각 1개 이하인 순차(Sequential) 노드를 의미한다. 특별 노드는 진출 에지가 두 개 이상인 분기(Branch) 노드이거나, 진입 에지가 두 개 이상인 합류(Join) 노드, 또는 이 두 가지가 혼합(Join & Branch) 노드를 의미한다.

특별 노드는 제어 흐름 그래프 유사도 비교에서 중요한 매칭 포인트가 된다. 따라서 일반 노드보다 우선하여 1순위 매칭 대상이 된다. 특별 노드의 매칭이 끝나면 특별 노드 사이에 위치한 일반 노드들을 대상으로 2순위 매칭을 한다. 예를 들어 그림 1의 노드 매칭에서 노드 BD는 특별 노드이다. 따라서 먼저 1순위 매칭을 하고, 노드 ABC, A, E는 일반 노드이므로 1순위 매칭이 끝나고 나서 2순위 매칭을 한다. 이와 같은 2단계 노드 매칭 기법을 사용하면 노드 사이의 매칭 대상을 줄일 수 있기 때문에 빠른 매칭이 가능하고, 서로 연관된 노드들 사이의 매칭을 유도하기 때문에 연관성이 없는 무분별한 노드 매칭을 배제시키는 효과가 있다.

두 노드 사이의 유사도 점수 계산은 생물 정보학에서 널리 사용되고 있는 서열 정렬 알고리즘을 사용하였다. 대표적인 서열 정렬 알고리즘에는 전체 정렬(Global Alignment)[6], 지역 정렬(Local Alignment)[7], 준 전체 정렬(Semi-global Alignment)[8]이 있다. 전체 정렬은 두 서열의 길이가 유사한 경우에 적합한 비교 방법이고, 지역 정렬은 두 서열 사이의 부분 유사도를 고려하고자 할 때 사용할 수 있는 방법이고, 준 전체 정렬은 길이 차이가 큰 두 서열의 정렬에 주로 활용되는 방법이다. 본 논문에서는 나열된 3가지 서열 정렬 방법 중에서 전체 정렬을 사용하였다. 그 이유는 크기가 다양한 노드들 사이에서 서로 비슷한 크기의 노드 쌍을 우선 매치시키기 위해서이다. 만일 노드가 난독화와 같은 변환에 의해서 두 개로 분리되었을 경우를 고려하여 인접한 두 노드를 결합시킨 후 결합된 노드와도 유사도를 계산하였다. 그리고 노드 결합 전과 후에 대해서 노드 유사도가 높은 값을 선택하였다. 이 방법을 사용하면 노드를 분리시키는 난독화와 같은 변환을 수행했더라도 분리된 두 노드를 한 개로 결합하여 비교하기 때문에 강인한 버스마크가 될 수 있다.

노드 매칭 순서는 노드 간 유사도 점수가 높은 순서를 따른다. 여기서 노드 간 유사도 점수는 두 노드 사이의 전체 정렬의 결과를 노드 크기로 나눈 결과를 이용했다. 두 노드 X와 Y의 유사도 점수를 구하는 계산식은 다음과 같다.

$$NodeSim(X,Y)=\frac{GlobalAlignment(X,Y)}{MIN(|X|,|Y|)}.$$

여기서 *GlobalAlignment*는 비교 대상 노드 X, Y의 전체 정렬 점수를 의미한다. 노드는 JVM 명령어의 시퀀스이기 때문에 서열 정렬을 이용하면 노드가 완전하게 일치하지 않아도 부분 일치에 해당하는 유사도를 얻을 수 있다. 전체 정렬에서 원소의 일치 점수는 1, 불일치와 공백 삽입에 대한 감점은 -1로 정하였다. 예를 들어, 원본 노드 'ABC'와 난독화에 의해서 x가 삽입된 'abxc' 노드 사이의 유사도는 다음과 같이 구할 수 있다.

$$NodeSim('ABC','abxc')$$

$$=\frac{GlobalAlignment('ABC','abxc')}{MIN(|'ABC'|,|'abxc'|)}=\frac{2}{MIN(3,4)}=2/3.$$

4.2.2 에지 매칭(Edge Matching)

에지 매칭은 원본 메소드에 포함된 에지가 비교 대상 메소드에도 존재하는지를 체크하는 것이다. 노드 매칭 단계에서 노드 A와 a가 매칭되고, 노드 B와 b가 매칭되었다고 가정하자. 만일 두 노드 A와 B 사이에 에지 (A,B)가 존재한다면, 비교 대상 메소드에서도 에지 (a,b)가 존재할 경우에 에지 유사도가 1이 된다. 그러나 난독화에 의해서 기존의 에지 사이에 새로운 노드가 추가될 수 있기 때문에 이것을 고려하여 에지 유사도는 다음과 같이 구한다.

$$EdgeSim((A,B),(a,b))$$

$$=\max\left(0,1-\frac{|ShortestPath(a,b)|-1}{threshold}\right).$$

여기서 *ShortestPath*는 두 노드 사이의 최단거리를 의미한다. 최단 거리를 계산하면 난독화에 의해서 새로운 노드가 추가되었을 경우에도 두 노드 사이의 거리를 고려하여 유사도를 구할 수 있다. 그리고 *threshold*는 난독화나 프로그램 변환에 의해서 노드가 삽입되는 것을 허용하는 임계값을 의미한다. 본 논문에서는 이 값을 5로 정하였다. 즉, 에지 유사도는 매칭된 노드 사이의 최단 거리가 1인 경우 최대 유사도 값인 1로 계산되고, 최단 거리가 증가함에 따라서 0.8, 0.6, 0.4, 0.2, 0 사이의 값을 갖는다. 만일 최단거리가 5를 초과할 경우는 에지 사이의 유사성이 없다고 판단하여 0으로 계산된다. 예를 들어, 그림 1의 에지 매칭 그림에서 (ABC,BD)는 (abxc,

bd)와 매칭되었는데 노드 abxc와 bd 사이에 y가 삽입되었기 때문에 최단 거리가 2로 증가하였고, 그 결과 에지 유사도는 다음과 같이 계산된다.

$$EdgeSim((ABC,BD),(abxc,bd))$$
$$=\max\left(0,1-\frac{|ShortestPath(abxc,bd)|-1}{5}\right)$$
$$=\max\left(0,1-\frac{1}{5}\right)=4/5.$$

4.3 메소드 유사도와 클래스 유사도의 계산

노드 매칭과 에지 매칭이 끝나면 메소드 유사도를 계산할 수 있다. 메소드 유사도는 메소드에 포함된 모든 노드의 유사도의 합계와 모든 에지의 유사도의 합계를 이용하여 구한다. 두 가지 유사도를 모두 고려하기 위해서 본 논문에서는 조화 평균을 이용하였다. 따라서 노드의 개수가 N이고 에지의 개수가 E인 메소드의 유사도는 다음과 같이 정의된다.

$$MethodSim=2\times\left(\frac{\frac{\sum NodeSim}{|N|}\times\frac{\sum EdgeSim}{|E|}}{\frac{\sum NodeSim}{|N|}+\frac{\sum EdgeSim}{|E|}}\right).$$

클래스 유사도는 클래스에 포함된 모든 메소드의 유사도 합으로 계산할 수 있다. 클래스에 포함된 메소드의 개수가 m개이고 각각의 메소드를 $M[i]$라고 할 때 클래스 유사도는 다음과 같이 정의된다.

$$ClassSim=\frac{\sum_{i=1}^{m}MethodSim(M[i])}{m}.$$

5. 실험 및 평가

본 논문에서 제안한 버스마크를 평가하기 위해서 대표적인 정적 버스마크 기법인 Tamada 버스마크, k-gram 버스마크, 트레이스 버스마크와 비교 실험을 하였다. 이 실험을 위해서 Stigmata[1]를 사용하였고, k-gram에서의 k 값은 3으로 정하였다.

그래프 매칭 기반 버스마크의 구현은 C 언어를 사용하였고, Intel® Xeon™ CPU 3.2GHz, RAM 4GB의 Linux 시스템에서 실험하였다. 실험 대상은 공개된 자바 클래스 71개[9]를 선택하였다.

5.1 신뢰도 평가

신뢰도 실험에서는 71개 예제 자바 클래스에 대한 서로 다른 조합의 수인 $_{71}C_2$ = 2485번의 비교 실험을 하였다. 그리고 클래스 유사도의 평균을 구하는 방법을 통해서 신뢰도를 측정하였다.

표 1 버스마크의 신뢰도 실험

	Tamada	k-gram	트레이스	그래프매칭
평균	**0.48**	**0.10**	**0.13**	**0.15**
최소값	0.10	0.00	0.00	0.00
최대값	1.00	1.00	1.00	1.00
표준편차	0.14	0.13	0.17	0.18
총 시간(초)	-	-	363.0	3809.2
평균시간(초)	-	-	0.15	1.53

표 1은 버스마크의 신뢰도 실험 결과이다. 신뢰도 실험에서는 서로 다른 알고리즘의 유사도를 비교하기 때문에 유사도 평균값이 낮을수록 신뢰도가 높은 것을 의미한다. 측정 결과 k-gram, 트레이스, 그래프 매칭 버스마크는 10%에서 15% 정도의 평균값을 얻어서 비슷한 성능을 보여주었고 Tamada 버스마크는 48%로서 가장 좋지 않은 결과를 얻었다. 소요 시간은 Tamada 버스마크와 k-gram의 경우는 버스마크 툴킷인 Stigmata 시스템이 소요 시간을 측정해주지 않기 때문에 제외하였고, 트레이스 버스마크와 그래프 매칭 버스마크에 대해서만 측정하였다. 신뢰도 실험에서 그래프 매칭 버스마크는 클래스 당 평균 1.53초의 성능을 보여주었고, 트레이스 버스마크는 이보다 10배 정도 빠른 성능을 보여주었다.

5.2 강인도 평가

강인도를 측정하기 위해서 난독화 도구인 Smokescreen[2]과 ZKM[3]을 사용하여 실험하였다. 실험은 원본 클래스과 난독화 클래스 파일과의 유사도를 비교하는 방법으로 수행하였다.

표 2, 3은 각각 Smokescreen과 ZKM에 대한 강인도

표 2 Smokescreen에 대한 버스마크의 강인도 실험

	Tamada	k-gram	트레이스	그래프매칭
평균	**0.73**	**0.70**	**0.95**	**0.97**
최소값	0.66	0.48	0.81	0.85
최대값	0.96	0.84	1.00	1.00
표준편차	0.06	0.08	0.05	0.03
총 시간(초)	-	-	337.1	109.2
평균시간(초)	-	-	4.75	1.54

표 3 ZKM에 대한 버스마크의 강인도 실험

	Tamada	k-gram	트레이스	그래프매칭
평균	**0.99**	**0.91**	**0.96**	**0.95**
최소값	0.91	0.75	0.88	0.81
최대값	1.00	1.00	1.00	1.00
표준편차	0.01	0.07	0.03	0.04
총 시간(초)	-	-	30940.3	109.3
평균시간(초)	-	-	435.78	1.54

1) http://stigmata.sourceforge.jp

2) http://www.leesw.com/smokescreen/

3) http://www.zelix.com/klassmaster/

실험 결과이다. 강인도는 난독화나 프로그램 변환에도 변하지 않는 성질을 의미하기 때문에 유사도가 높을수록 강인도가 높은 것을 의미한다. Smokescreen을 사용한 실험에서는 트레이스 버스마크와 그래프 매칭 버스마크는 90%가 넘는 우수한 결과를 보였고, Tamada, k-gram 버스마크가 70% 정도의 평균값을 보였다. 소요시간은 트레이스 버스마크에 비하여 그래프 매칭 버스마크가 3배정도 빠르게 동작하였다.

ZKM을 사용한 실험에서는 네 가지 버스마크가 모두 90% 이상의 평균값을 보였다. 다만 소요 시간은 트레이스 버스마크가 그래프 매칭 버스마크보다 282배 정도 느린 성능을 보여주었다. ZKM에 대해서 트레이스 버스마크의 수행 성능이 크게 떨어지는 이유는 다음과 같다. 트레이스 버스마크는 제어 흐름 그래프로부터 실행 가능한 모든 트레이스를 추출하여 비교하는 방법을 사용한다. ZKM은 제어 흐름 구조를 복잡하게 하기 위해서 노드를 분리하고 에지를 추가하는 방법을 사용하기 때문에 제어 흐름 그래프가 매우 복잡해진다. 특히 분기문의 개수에 대해서 기하급수적으로 추출되는 트레이스 개수가 증가하기 때문에 프로그램의 제어 흐름 그래프의 복잡도에 대해서 트레이스 버스마크는 매우 큰 성능 저하를 보인다는 문제가 있다.

본 논문에서 제안한 그래프 매칭을 이용한 버스마크는 난독화 실험에서 트레이스 버스마크보다 빠른 실행 성능을 보여주었다. 물론 난독화에 의해서 CFG의 노드와 에지 개수가 증가하기 때문에 난독화 전과 비교하면 그래프 매칭에 시간이 더 많이 소요된다.

표 4는 난독화에 의한 그래프 매칭 소요 시간의 변화를 보여준다. 총 소요 시간을 71번의 비교 회수로 나눌 경우에 난독화 전은 평균 1.53초로 계산되고 SmokeScreen과 ZKM 난독화 후에 평균 1.54초로 계산되기 때문에 난독화에 의한 그래프 매칭 소요 시간이 증가하지 않고 일정한 것처럼 보인다. 그러나 파일 입출력과 CFG 생성에 소요되는 시간을 빼고 순수하게 그래프 매칭에만 소요되는 시간을 비교해보면 SmokeScreen과 ZKM에 의해서 각각 20.1%, 55.2%의 매칭 시간이 증가된 것을 확인할 수 있다.

표 4 난독화에 의한 그래프 매칭 소요 시간 변화(초)

	난독화 전	SmokeScreen 난독화 후	ZKM 난독화 후
총 소요 시간	108.83	109.15	109.30
입출력, CFG 생성 시간	108.44	108.68	108.69
순수 그래프 매칭 시간	0.39	0.47	0.61
난독화전 대비 매칭 시간 변화	-	20.1% 증가	55.2% 증가

6. 결 론

본 논문에서는 프로그램의 제어 흐름 그래프 매칭을 기반으로 하는 자바 버스마크를 제안하였고 신뢰도와 강인도 실험을 하였다. 실험 결과로부터 본 논문에서 제안하는 그래프 매칭 기반 버스마크가 기존의 Tamada, k-gram 버스마크보다 난독화에 대한 강인도가 높다는 것을 확인하였고, 트레이스 버스마크에 버금가는 신뢰도와 강인도를 보여주는 것을 확인하였다. 또한, 소요 시간 면에서도 난독화의 여부와 상관없이 일정한 성능을 보여주었다.

참 고 문 헌

[1] Prechelt, L., Malpohl, G. and Philippsen, M. "Finding plagiarisms among a set of programs with jPlag," *Journal of Universal Computer Science*, vol.8, no.11, pp.1016-1038, 2002.

[2] Tamada, H., Nakamura, M., Monden, A., Matsumoto, K., "Java birthmark Detecting the software theft," *IEICE Transactions on Information and Systems*, E88-D, 9, pp.2148-2158, Sept. 2005.

[3] Ginger Myles and Christian Collberg, "k-gram Based Software Birthmarks," In *Proceeding of the 2005 ACM Symposium on Applied Computing*, pp.314-318. Santa Fe, New Mexico, USA, 2005.

[4] Heewan Park, Seokwoo Choi, Hyun-il Lim, and Taisook Han. "Detecting Code Theft via a Static Instruction Trace Birthmark for Java Methods," *The 6th IEEE International Conference on Industrial Informatics (INDIN)*, July 2008.

[5] Laura Zager, "Graph similarity and matching," MS Thesis, EECS, MIT, 2005.

[6] Saul Needleman and Christian Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, 48(3):443-453, 1970.

[7] Temple Smith and Michael Waterman. "Identification of Common Molecular Subsequences," *Journal of Molecular Biology*, 147: 195-197, 1981.

[8] Brudno M, Malde S, Poliakov A, Do CB, Couronne O, Dubchak I, Batzoglou S., "Glocal alignment: finding rearrangements during alignment," *Bioinformatics*, vol.19 Suppl.1:i54-62, 2003.

[9] Mark Allen Weiss, "Data Structures and Algorithm Analysis in Java," Addison-Wesley, http://users.cs.fiu.edu/~weiss/dsaajava/code/.