Asia-pacific Journal of Convergent Research Interchange
Vol.9, No.11, November 30 (2023), pp.115-125
http://dx.doi.org/10.47116/apjcri.2023.11.11

An Approach to Comparing Java Programs through the Graph Edit Distance with Similar Instruction Matching of Control Flow Graphs

제어 흐름 그래프의 유사 명령어 매칭 기반 그래프 편집 거리를 이용한 자바 프로그램의 유사도 비교 방법

Hyun-il Lim[1]

임현일[1]

[1] *Professor, Department of Computer Engineering, Kyungnam University, South Korea, hilim@kyungnam.ac.kr*

Abstract: As software is essentially used in a wide range of technologies, various analysis techniques are employed to understand the characteristics of software and optimize it. Software similarity analysis is utilized in various fields, including duplicate code detection, plagiarism detection, code optimization, and malicious code detection. Control flow graphs are static data structures that express operational and structural characteristics of software. So, they are widely used for representing and understanding software's features. This paper proposes a method for analyzing similarities between Java programs by applying the edit distance of control flow graphs. To enhance the flexibility of matching instructions within control flow graphs, we have designed a method for classifying similar instructions and matching them based on these classes. This method can improve the reliability of similarity analysis results for programs that have similar control flow structures and instructions. We conducted experiments using Java benchmark programs and evaluated the accuracy and reliability of the proposed method. The results indicated that the method achieved a high accuracy rate of 98% in comparison experiments between similar Java programs. It is expected that the proposed method will be useful for analyzing software similarity and understanding the operational characteristics of software by matching similar instructions in programs.

Keywords: Software Similarity, Control Flow Graph, Graph Edit Distance, Software Analysis

요약: 광범위한 기술 분야에서 소프트웨어가 필수적으로 사용되면서 소프트웨어의 특성을 이해하고 최적화하기 위한 다양한 분석 기술들이 활용되고 있다. 소프트웨어 유사성 분석은 중복 코드 검출, 표절 탐지, 코드 최적화, 악성코드 탐지 등 다양한 분야에 활용된다. 제어 흐름 그래프는 소프트웨어의 동작 특성을 표현할 수 있는 정적인 구조로서 소프트웨어의 구조적 특성을 표현하거나 동작 특성을 이해하는데 널리 활용된다. 본 논문에서는 자바 프로그램의 유사도 분석을 위해 제어 흐름 그래프의 편집 거리를 응용하는 방법을 제안한다. 그래프 편집 거리의 명령어 매칭에 대한 유연성을 개선하기 위해 유사한 명령어에 대한 클래스를 분류하고 명령어의 매칭에 적용하는 방법을 설계한다. 이 방법은 서로 유사한 제어 흐름 구조와 명령어를 가지는 프로그램에 대한 유사도 분석의 신뢰도를 높일 수 있다. 본

Received: July 21, 2023; 1st Review Result: August 26, 2023; 2nd Review Result: September 28, 2023
Accepted: October 25, 2023

논문에서 제안한 방법의 성능을 검증하기 위해 자바 벤치마크 프로그램에 대해 실험을 하고, 결과의 정확도와 신뢰도를 평가하였다. 본 논문에서 제안한 방법은 유사 프로그램의 비교 결과에서 98%의 높은 정확도를 보여주었다. 본 논문에서 제안한 방법은 소프트웨어 유사도 분석 및 소프트웨어의 유사 명령어 매칭을 통해 소프트웨어의 동작 특성을 이해하는데 활용될 수 있을 것이라 기대된다.

핵심어: 소프트웨어 유사도, 제어 흐름 그래프, 그래프 편집 거리, 소프트웨어 분석

1. 서론

소프트웨어의 제어 흐름 그래프는 프로그램 실행에 필요한 연산들의 구조적인 특징을 표현하는 자료 구조이다[1][2]. 제어 흐름 그래프는 프로그램의 실행에 필요한 반복, 조건 분기, 함수 호출 등과 같은 실행 시간의 동적인 동작 특성을 잘 표현할 수 있다. 따라서, 제어 흐름 분석은 프로그램의 동작을 이해하고 분석하기 위해 널리 활용된다. 제어 흐름 분석은 소프트웨어의 특성을 이해하고 코드 최적화 뿐만 아니라 소프트웨어 유사성 분석에 활용될 수 있다. 소프트웨어 유사성 분석은 유사한 코드의 검출, 표절 탐지, 코드 분석 및 리팩토링 등 다양한 분야에 활용할 수 있고, 악성 소프트웨어 탐지를 위해 제어 흐름 그래프와 유사도 분석이 적용되기도 한다[3-5]. 또한, 소프트웨어의 특성을 명령어와 실행 구조 관점에서 상세하게 기술할 수 있는 데이터이기 때문에 빅데이터, 인공지능 등 데이터를 활용하는 응용 분야에서 효과적으로 사용된다.

본 논문에서는 자바 프로그램의 유사성을 판단하기 위해서 제어 흐름 그래프를 분석하고, 제어 흐름 그래프의 그래프 편집 거리를 응용해서 유사도를 측정하는 방법을 제안한다. 부분적 변화가 있는 유사 프로그램의 비교에서 안정적인 결과를 얻기 위해 유사한 명령어를 클래스로 분류하고 유사 명령어의 편집 거리를 노드 매칭에 반영하는 방법을 제안한다. 본 논문에서 제안한 방법의 성능을 평가하기 위해 오픈소스 Java 벤치마크 프로그램에 대해 비교 실험을 수행하였다. 실험 결과를 통해 제안된 비교 방법은 자바 프로그램의 유사성을 판단하기에 효과적인 방법임을 확인할 수 있다.

본 논문의 구성은 다음과 같다. 제2장에서는 제어 흐름 그래프 유사도 분석에 관한 기존 연구 및 그래프 편집 거리를 이용한 그래프 비교 방법을 소개한다. 제3장에서는 프로그램의 제어 흐름 그래프 특성을 기술하고, 유사 명령어 클래스를 응용한 제어 흐름 그래프 비교 방법을 설계한다. 제4장에서는 본 논문에서 제안한 방법의 성능 검증을 위한 실험 환경과 실험 결과를 보여준다. 제5장에서 이 논문의 결론을 맺는다.

2. 관련 연구

2.1 기존의 소프트웨어 유사성 비교 방법

소프트웨어의 제어 흐름 그래프를 비교하는 방법은 구조적 특성을 비교하기 위해 그래프 매칭 알고리즘, 그래프 편집 거리 및 서브그래프 동형성(Subgraph isomorphism) 등의 방법을 활용할 수 있다. 기존의 연구 방법에서 소프트웨어의 제어 흐름 그래프를 효과적으로 비교하기 위해서는 소프트웨어가 가지는 구조적인 특성을 적극적으로 활용하고 있다. 소프트웨어의 특성을 반영하여 소프트웨어 비교에 적용한 관련 연구로서

Flake 등은 일부 부분에 업데이트 및 수정된 프로그램의 차이를 비교하기 위해 구조적 비교 방법은 제어 흐름 그래프의 노드와 에지가 가지는 특성값을 비교하고 단계적으로 확대 매칭하는 방법을 활용하였다[6]. 또한, 소프트웨어 동작에 중요한 역할을 하는 API 및 함수 호출 정보를 이용하는 방법[7], 제어 흐름 그래프의 기본 블록을 매칭하고 명령어 경로를 비교하는 방법은 소프트웨어의 동작 특성을 나타내는 함수 및 명령어들의 유사성을 비교하고 있다[8]. Kapoor 등은 제어 흐름 그래프의 특징을 비교한 점수와 tf-idf 가중치[9]를 통해 악성 코드를 유사도에 따라 여러 클래스로 분류하는 방법을 제안하였다[3]. Alasmary 등은 제어 흐름 그래프가 가지는 노드와 에지의 수, 노드간의 최단 경로, 부분그래프 등 제어 흐름 그래프의 특성 정보를 비교하고, 악성코드 분석에 적용하였다[4]. Isah 등은 소프트웨어 표절 탐지를 위해 제어 흐름 그래프에 나타나는 구문 및 문장의 구조적 유사성을 비교하는 방법을 제안하였다[10].

2.2 그래프 편집 거리를 이용한 그래프의 비교

그래프 편집 거리는 서로 다른 두 그래프 사이에 차이점 또는 유사성을 정량적으로 측정하는 방법으로 활용된다[11][12]. 이 방법은 두 그래프의 유사성을 정량적으로 측정하기 위해 한 그래프를 다른 그래프로 변환하는데 필요한 편집 연산의 최소 비용을 계산한다. 사용되는 편집 연산은 노드와 에지의 삽입, 삭제, 치환 뿐만 아니라 노드와 에지에 포함된 속성값을 변경하는 연산 등을 포함한다. 따라서, 두 그래프의 유사성은 한 그래프를 다른 그래프로 변경하기 위한 편집 연산의 비용이 얼마나 작은지를 통해 표현할 수 있고 그래프 편집 거리 문제는 변경에 필요한 편집 연산의 총비용을 최소화하는 연산 경로를 찾는 문제로 나타낼 수 있다. 각 편집 연산의 비용은 노드 속성 간의 거리, 그래프의 구조적 특성, 그래프 요소를 표현하는 노드와 에지 변경의 중요도 및 특성 등을 기준으로 정해지고, 그래프의 크기, 노드 또는 에지 간의 의미론적 유사성 등의 요소를 고려한다. 따라서, 두 그래프 G1과 G2 사이의 그래프 편집 거리 GED(G1, G2) 는 다음 수식 (1)과 같이 정의된다.

$$\mathrm{GED}(\mathrm{G1},\mathrm{G2}) = \min_{(e_1,\ldots,e_k)\in\tau(G1,G2)} \sum_{i=1}^{k} c(e_i) \tag{1}$$

여기서, $\tau(G1,G2)$ 는 그래프 G1을 G2로 변환하기 위한 그래프 편집 연산 경로의 집합을 나타내고, $c(e)$ 는 그래프 편집 연산 e 의 비용을 의미하며, $c(e) \geq 0$ 의 조건을 만족한다. 그래프 편집 연산은 노드 삽입(ni), 노드 삭제(nd), 노드 치환(ns), 에지 삽입(ei), 에지 삭제(ed), 에지 치환(es) 등 6개 기본 연산으로 구성되고, 그래프의 특성에 따라 각 편집 연산의 비용은 각각 $c_{ni}(e), c_{nd}(e), c_{ns}(e), c_{ei}(e), c_{ed}(e), c_{es}(e)$ 로 정의한다. 또한, 그래프의 특성에 따라 새로운 노드를 추가하면서 에지를 나누는 에지 분할, 노드를 삭제하면서 두 개의 에지를 합치는 에지 통합 등 복합 연산을 추가로 정의할 수 있다.

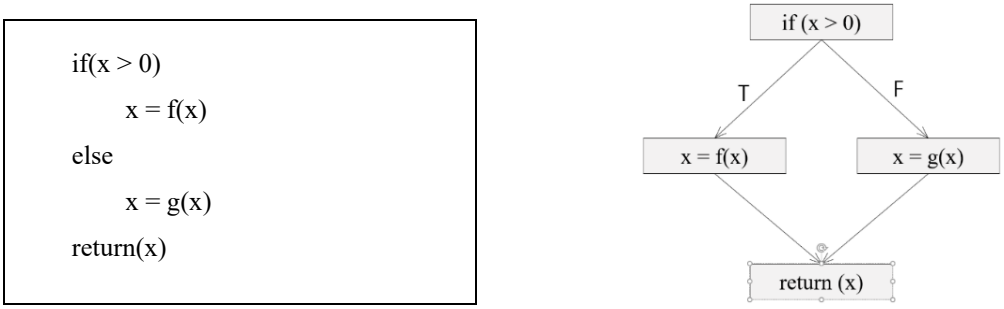
그래프 편집 거리는 최소 비용이 되는 연산 경로를 찾는 최적화 문제를 해결해야 한다. 최적해를 구하는 방법은 최소 경로 탐색 문제를 해결하기 위한 A* 탐색 알고리즘, 그래프 탐색 알고리즘 등으로 접근할 수 있지만 NP 문제로 알려져 있다. 효율적으로 계산하기 위해 헝가리 알고리즘(Hungarian algorithm), 이분 그래프 매칭 알고리즘(Bipartite graph matching algorithm) 등 그래프의 특성 및 매칭을 활용한 근사해를 구하는 방법들이 적용되고 있다[11][12].

그래프 편집 거리는 복잡한 구조로 표현되는 그래프를 비교하기 위해 범용적으로 활용될 수 있다. 그래프 유사성 비교, 그래프 클러스터링, 그래프 구조의 패턴 인식, 그래프 기반 데이터 분석, 이미지 인식, 생물 정보학, 소셜 네트워크 분석 및 패턴 인식 등 다양한 분야에서 분석 기술에 활용될 수 있고, 그래프로 표현되는 다양한 정보들의 구조적 유사성을 이해하는데 효과적이다. 본 논문에서는 그래프 편집 거리를 활용하여 제어 흐름 그래프의 비교에 최적화하기 위해 유사 명령어를 매칭하는 방안을 적용한다.

3. 그래프 편집 거리를 이용한 제어 흐름 그래프의 비교

3.1 제어 흐름 그래프의 특성

소프트웨어의 제어 흐름 그래프는 선형적으로 기술된 소프트웨어에 대해 실행 흐름에 따른 비선형적인 구조를 표현하는 그래프 형태로 표현한다[1][13]. 따라서, 제어 흐름 그래프는 소프트웨어의 정적인 구성 뿐만 아니라 실행 흐름 관점의 동작 특성을 함께 나타낼 수 있기 때문에 소프트웨어의 특성을 이해하고 분석하는데 중요한 역할을 한다. 제어 흐름 그래프 G = (N, E)로 정의되며, N과 E는 각각 제어 흐름 그래프의 노드와 에지의 집합을 나타낸다. 제어 흐름 그래프의 노드는 프로그램 실행에 필요한 구문 정보와 명령어들을 포함하는 기본 블록(Basic block)을 나타내고, 에지는 기본 블록 사이에 실행 가능한 경로를 나타낸다.



[그림 1] 예제 프로그램과 제어 흐름 그래프

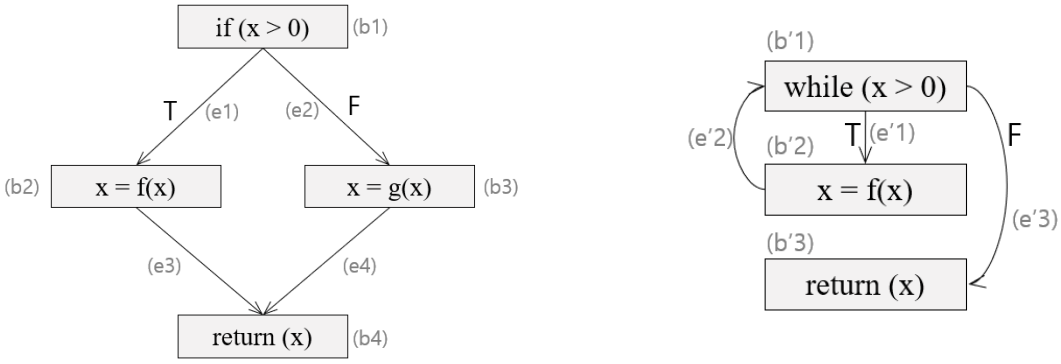
[Fig. 1] An Example Program and its Control Flow Graph

[그림 1]은 예제 프로그램과 제어 흐름 그래프를 보여주고 있다. 예제 프로그램은 if 분기문과 실행 경로에 포함된 함수 호출을 보여주고 있다. 제어 흐름 그래프에서는 선형적인 구조를 가지는 프로그램에 대해 분기문의 실행 경로를 그래프로 보여주고 있다. 이와 같이 프로그램에서 선형적으로 기술된 명령어의 구조는 제어 흐름 그래프를 통해 기본 블록과 에지의 비선형적인 실행 경로로 표현될 수 있기 때문에 제어 흐름 분석은 프로그램의 동작을 기술하고 이해하기 위한 분석 방법으로 널리 활용된다.

3.2 유사 명령어 분류 및 매칭을 통한 제어 흐름 그래프 비교 방법 설계

그래프 편집 거리는 두 그래프의 유사성 및 차이점을 분석하기 위한 방법으로 그래프 형태의 데이터 구조에 범용적으로 적용할 수 있다. 하지만, 일반적인 추가, 삭제, 수정 등의 기본 편집 연산을 그대로 적용한다면 소프트웨어의 동작 특성을 표현하도록 설계된

제어 흐름 그래프의 유사성 분석에서 안정적인 결과를 도출하는데 한계가 있다. 따라서, 소프트웨어가 가지는 제어 흐름 그래프의 특성에 맞춰서 편집 연산의 비용 모델을 설계하고, 편집 거리 탐색 모델을 구성할 필요가 있다.



[그림 2] 두 제어 흐름 그래프에서 그래프 편집 거리 예제

[Fig. 2] Example of Graph Edit Distance for Two Control Flow Graphs

[그림 2]는 그래프 편집 거리를 위한 두 제어 흐름 그래프의 예제를 보여주고 있다. 두 프로그램의 코드상 선형적인 명령어 구성은 유사하지만 두 제어 흐름 그래프는 각각 if 분기 구문과 while 반복 구문을 포함하고, 구조 특성상 분기와 반복이라는 상이한 동작 구조를 가지고 있다. 따라서, [그림 2(b)] 그래프의 반복 구조는 상위 노드로 되돌아 가는 $(e'2)$ 에지가 포함되어 있는 것을 확인할 수 있다.

노드 b와 에지 e, e'에 대해 노드 삽입, 삭제, 치환 및 에지 삽입, 삭제, 치환 등 6개 기본 편집 연산의 비용을 순서대로 각각 $c_{ni}(b), c_{nd}(b), c_{ns}(b,b'), c_{ei}(e), c_{ed}(e), c_{es}(e,e')$라고 하면, 치환 연산의 비용은 해당 요소를 삭제 후 새로 삽입하는 연산 비용의 합보다 작다$(c_{ns}(b,b') < c_{nd}(b) + c_{ni}(b'))$. 그러면, [그림 2(a)]의 제어 흐름 그래프를 [그림 2(b)]로 변경하기 위해서 필요한 그래프 편집 연산의 최소 비용은 수식 (2)와 같이 노드 및 에지의 치환 연산을 포함하는 6개 편집 연산 비용의 합으로 구할 수 있다.

$$\mathrm{GED}(\mathrm{G1},\mathrm{G2}) = c_{ns}(b1,b'1) + c_{es}(e3,e'2) + c_{es}(e2,e'3) + c_{nd}(b3) + c_{ed}(e2) + c_{ed}(e4) \quad (2)$$

유사한 정도와 상관없이 그래프 편집 연산에서 기본 블록에 포함된 명령어가 일치하지 않으면 다르다고 판단하기 때문에 유사한 동작 특성을 가지는 명령어를 포함한 경우에 안정적인 비교를 하기 어렵다. 예를 들어, 서로 다른 기본 블록에 대해 치환 연산을 수행할 때 기본 블록 명령어의 유사한 정도는 고려하지 않고 일치 여부만을 반영한다면 유사한 구조를 가지는 프로그램과 다른 구조를 가지는 프로그램 사이에 유사성을 효과적으로 변별하기 어려워진다. 따라서 노드 및 에지의 치환 연산을 수행하기 위해서 서로 유사한 명령어로 구성되는 기본 블록 및 에지를 치환하는 경우는 그렇지 않은 경우에 비해 편집 비용을 보정해주는 것이 유사한 프로그램을 분석하는데 효과적인 비교 모델이 된다.

본 논문에서는 편집 연산의 비용을 설계할 때, 기본 블록에 포함된 명령어의 동작 특성의 유사성을 고려하여 유사한 명령어에 대한 편집 비용을 조정할 수 있도록

설계하였다. 이를 위해 본 연구에서는 자바 프로그램의 제어 흐름 그래프에 포함된 총 202개의 자바 바이트코드 명령어[14]를 동작 특성의 유사성에 따라 66개의 유사 클래스로 분류하고, 유사 클래스에 속하는 바이트코드 명령어에 대한 치환 연산을 우선 매칭할 수 있도록 비교 모델을 설계하였다. 이 과정을 통해 자바 바이트코드 b의 유사 클래스를 $class(b)$라고 할때, 제어 흐름 그래프 G1과 G2의 유사 명령어 매칭 기반 편집 거리 $\mathrm{GED}_{CFG}(\mathrm{G1},\mathrm{G2})$는 다음 수식 (3)과 같이 설계하였다.

$$\mathrm{GED}_{CFG}(\mathrm{G1},\mathrm{G2}) = \min_{(e_1,\ldots,e_k)\in\tau(G1,G2)} \sum_{i=1}^{k} c(e_i) \tag{3}$$

$$c_{ni}(b) = c_{nd}(b) = c_{ei}(e) = c_{ed}(e) = 1.0$$

$$c_{ns}(b,b') = \begin{cases} 0.6, & \text{if } class(b) = class(b') \\ 1.0, & \text{if } class(b) \neq class(b') \end{cases}$$

$$c_{es}(e,e') = \begin{cases} 0.4, & \text{if } class(n_1) = class(n_1') \text{ and } class(n_2)) = class(n_2') \\ 0.6, & \text{if } class(n_1) = class(n_1') \;\; or \;\; class(n_2)) = class(n_2') \\ 1.0, & \text{if } class(n_1) \neq class(n_1') \text{ and } class(n_2)) \neq class(n_2'), \\ & \text{where } e = (n_1, n_2) \text{ and } e' = (n_1', n_2') \end{cases}$$

두 제어 흐름 그래프의 유사 명령어 매칭을 통한 그래프 편집 거리는 편집 연산 경로의 비용으로 계산되고, 노드와 에지의 추가 및 삭제를 위한 연산의 비용은 기본값 1로 정의한다. 유사 명령어를 매칭하기 위해 노드의 치환 연산은 유사 클래스의 명령어가 치환되는 경우에 편집 비용은 0.6으로 조정하였다. 또한, 에지 치환 연산은 에지가 연결하는 노드의 유사도에 따라 0.4 또는 0.6의 비용으로 조정한다. 두 제어 흐름 그래프의 편집 거리로 부터 프로그램의 유사도 $Sim_{GED}(\mathrm{G1},\mathrm{G2})$ 는 수식 (4)와 같이 정의한다.

$$Sim_{GED}(\mathrm{G1},\mathrm{G2}) = 1 - \frac{\mathrm{GED}_{CFG}(\mathrm{G1},\mathrm{G2})}{\max\big(len(G1), len(G2)\big)} \tag{4}$$

여기서 $len(G1)$ 은 제어 흐름 그래프 G1에 포함된 노드와 에지의 수를 의미한다. 그래프 편집 거리에 대해 그래프 노드와 에지의 크기로 나누어줌으로써 최대값 100%가 될 수 있도록 정규화하였다. 따라서, 편집 거리의 크기에 따라 프로그램 유사도는 0% - 100% 범위에서 결정된다.

[그림 2]의 예제에서 기존의 그래프 편집 거리는 각 편집 연산의 비용을 기본값 1로 정할 때 $\mathrm{GED}(\mathrm{G1},\mathrm{G2}) = 6$이 되고, 프로그램 유사도 $Sim(G1,G2) = 1 - {}^{6}/_{8} = 0.25$가 된다. 본 논문에서 제안한 유사 명령어 매칭을 그래프 편집 거리에 적용하면 노드 $(\mathrm{b1})$과 $(\mathrm{b'1})$이 동일 클래스에 속하는 유사 명령어로 매칭되고, 치환 연산의 비용이 각각

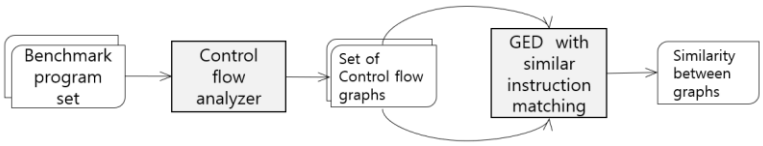
$$c_{ns}(b1,b'1) = 0.6,\ c_{es}(e3,e'2) = 0.6,\ c_{es}(e2,e'3) = 0.4$$

이므로 유사한 명령어에 대한 편집 비용이 조정된다. 따라서, 그래프 편집거리 $\mathrm{GED}_{CFG}(\mathrm{G1},\mathrm{G2}) = 4.6$ 이 되고, 유사 명령어 매칭을 통한 프로그램의 유사도는 $Sim_{GED}(\mathrm{G1},\mathrm{G2}) = 1 - {}^{4.6}/_{8} = 0.425$ 가 된다. 예제에서와 같이 유사한 특성의 명령어는 유연하게 매칭할 수 있기 때문에 유사한 구조의 프로그램에 대한 유사도를 서로 다른 프로그램과 변별력있게 구분할 수 있다.

4. 실험 및 평가

4.1 실험 환경 구성

본 절에서는 본 논문에서 제안한 유사도 분석 방법의 결과를 평가하기 위한 실험을 수행한다. 자바 프로그램의 제어 흐름을 분석하고 제어 흐름 그래프를 생성하기 위한 분석기를 C언어로 구현하고, 제어 흐름 그래프의 편집 거리를 측정하고, 유사도를 분석하는 분석기는 파이썬으로 구현하였다. 실험 대상 프로그램은 자바 오픈소스 프로그램 사이트[15]로 부터 Apache Derby, McKoi, 그리고 SmallSQL 데이터베이스 프로그램에 포함된 클래스파일들을 벤치마크 프로그램으로 이용하였다.



[그림 3] 제안 방법의 평가 실험 환경의 구성

[Fig. 3] The Structure of Experimental Environments for Evaluating the Proposed Approach

벤치마크 데이터로 사용된 자바 클래스파일에서 바이트코드를 분석하고 제어 흐름 그래프를 생성하였다. 사이즈가 작은 클래스파일은 공유 수행 패턴을 포함하는 유사한 패턴의 프로그램을 포함하고 있기 때문에 결과의 신뢰성을 얻기 위해 제어 흐름 그래프의 기본 블록 및 에지의 개수가 40개 이상인 제어 흐름 그래프를 대상으로 벤치마크 프로그램 집합을 구성하였다. 벤치마크 집합에 포함된 그래프는 총 197개이고, 기본 블록의 개수는 최대 590개 평균 122개이며, 에지의 개수는 최대 506개 평균 91개이다. 그리고, 유사한 프로그램의 비교 결과를 평가하기 위한 유사 프로그램의 비교 실험에 사용할 대상 프로그램을 확보할 필요가 있다. 이를 위해 동일한 실행 결과를 가지지만 명령어 및 구조를 난독화하고 구조를 변경하는 난독화 도구 Smokescreen과 Zelix KlassMaster[16]를 적용한 프로그램을 원본 프로그램에 대한 유사 프로그램 집합에 포함하였다.

[그림 3]은 본 실험 환경의 시스템 구조를 보여주고 있다. 제어 흐름 분석기(Control flow analyzer)는 벤치마크 프로그램으로부터 바이트코드를 분석하고 각 프로그램에 대한 제어 흐름 그래프를 생성한다. 생성된 제어 흐름 그래프는 유사 명령어 매칭 기반 편집 거리를 비교하고 두 프로그램 사이의 유사도를 분석한다. 난독화 도구를 통해 생성된 수정된 프로그램과 원본 프로그램의 유사도를 비교하고, 유사한 프로그램에 대한 비교 결과를 측정하였다. 또한, 서로 상이한 프로그램간의 비교 결과를 얻기 위해 벤치마크에 포함된 서로 다른 프로그램을 비교하고 결과를 통해 제안된 방법의 신뢰도를 평가하였다.

4.2 유사 명령어 매칭 기반 제어 흐름 그래프 비교 실험 결과

본 실험에서 유사 명령어 매칭을 통해 프로그램 유사도 분석 결과를 평가하기위해 유사한 프로그램과 서로 다른 프로그램의 벤치마크 집합에서 제어 흐름 그래프를 함께

비교하였다. 수식 (2)와 (3)에서 설계한 방법으로 벤치마크 프로그램 사이의 유사도를 측정하였으며 편집 거리에 따라 각 유사도는 0%에서 100%사이에서 결정된다. 자바 프로그램에 대한 유사도 여부를 판단하는 임계값을 설정하기 위해 사전 실험 결과를 분석하고 위양성과 위음성을 줄일 수 있는 값으로 설정하였다. 임계값이 커지면 위음성은 줄어들지만 위양성이 증가하기 때문에 균형을 이루는 값이 필요하다. 자바 프로그램에 대한 사전 실험을 통해 임계값 0.1 (유사도 90%)를 기준으로 자바 프로그램의 유사도 여부를 판단하고 본 실험 결과를 통해 제안된 방법의 성능을 평가한다.

[표 1] 벤치마크 프로그램의 제어 흐름 그래프 비교 실험 결과

[Table 1] Experimental Results of the Comparisons of Control Flow Graphs between Benchmark Programs

Input	Analysis results								
	Positives		Negatives		Total	Output range	Mean	Median	Accuracy
Positive	True Pos.	362	False Neg.	32	394	[0.7672 - 1.0]	0.9624	0.9741	0.9188
Negative	False Pos.	362	True Neg.	18,944	19,306	[0.0033 - 1.0]	0.5274	0.5258	0.9812

[표 1]은 본 논문에서 제안한 방법의 실험 결과를 보여주고 있다. 유사 프로그램에 대한 비교 실험에서 총 394쌍에 대해 비교 실험을 수행하였고, 362개는 유사한 프로그램으로 검출하였고 32개는 90% 미만의 유사도를 보여주었다. 전체 유사도의 범위는 최소값 76.7%, 평균 96.2%, 중간값 97.4% 이고, 분석 정확도는 91.9% 였다. 반대로, 서로 다른 프로그램 사이의 비교에서는 총 19,306쌍의 비교에 대해 18,944쌍은 서로 다른 프로그램으로 인식하였으며 362쌍의 프로그램은 90% 이상의 유사도를 보여주었다. 서로 다른 프로그램의 유사도 평균은 52.7%이고 중간값은 52.6%, 분석 정확도는 98.1% 였다. 서로 다른 제어 흐름 그래프에 대해서 100%의 유사도를 가진 경우는 벤치마크 프로그램중 하나의 파일 내에 정의된 클래스중 동일한 구조를 가지는 클래스가 중복 정의된 경우 각각의 클래스가 벤치마크 집합에 중복 포함되었기 때문이다. 따라서, 이런 경우는 오탐지이기 보다는 프로그램내에 중복된 코드가 검출되었기 때문으로 분석된다.

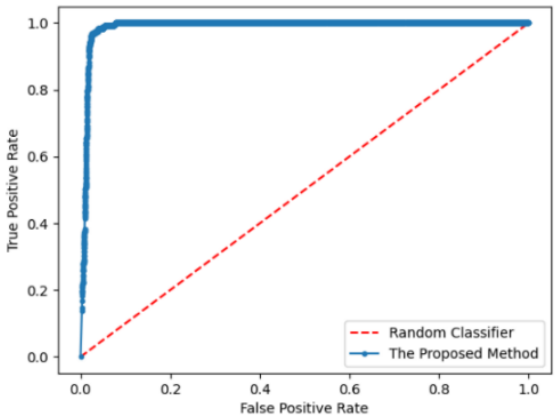
[표 2] ROC 곡선과 정밀도-재현율 분석 결과 (임계값 = 0.1)

[Table 2]] ROC and Precision Recall Analysis Results of the Experiments with Threshold = 0.1

ROC Analysis		Precision Recall Analysis		Overall accuracy
True Positive Rate	0.9188	Precision	0.5000	0.9800
False Positive Rate	0.0188	Recall	0.9188	
AUC	0.989	AUC	0.521	

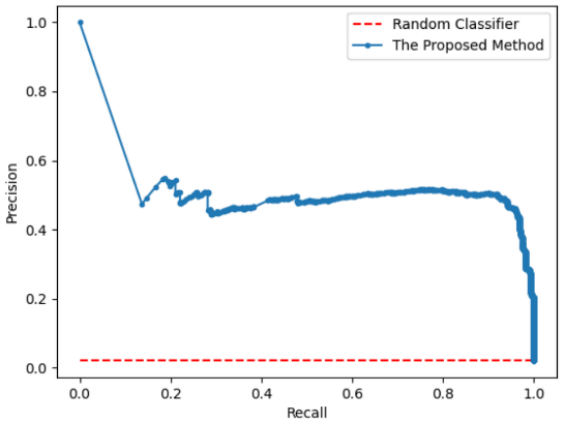
유사도의 이진 분류 모델을 평가하기 위해 ROC(Receiver Operating Characteristic) 곡선과 정밀도와 재현율(Precision and recall, PR) 곡선은 임계값의 변화에 따른 이진 분류의 예측 정확도를 평가하는 척도로 적용된다[17]. [표 2]와 [그림 4]는 각각 ROC 분석과 정밀도-재현율 분석 실험 결과와 실험 결과의 ROC 곡선을 보여주고 있다. 실험 결과 분석에서 분류 임계값 0.1일 때 참양성율(True positive rate) 91.9%, 위양성율(False positive rate)은 0.19%로 분석되었다. 정밀도와 재현율은 각각 50.0%, 91.9%이고, 유사 프로그램 분석에 대한 전체 정확도는 98.0%의 정확도를 보여주고 있다. ROC 곡선의 AUC(Area Under Curve)는 0.989로 상당히 우수한 결과를 보여주었고, 정밀도-재현율(PR) 곡선의 AUC는 0.521로 평가되었다. [그림 5]는 정밀도-재현율(PR) 곡선을 보여주고 있다. PR 곡선에서 높은 임계값 범위의 분류 정확도가 낮아지는 것은 벤치마크 프로그램내에 유사 중복 코드로 인해 위

양성 결과가 영향을 준 것으로 분석되고, 이로 인해 AUC 값이 ROC 곡선에 비해서 낮게 나타난 것을 확인할 수 있다. 또한, PR 곡선에서 0.7이상의 높은 임계값을 가지는 영역에서 정확도가 낮아지고 임계값 0.1 정도에서 정확도가 곡선의 최대치에 도달하는 것을 확인할 수 있다. 따라서, 임계값 0.1은 자바 프로그램의 유사성을 판단하는 임계값으로 적절함을 확인할 수 있다. 실험 결과로 부터 유사 코드 매칭 기반 그래프 편집 거리는 제어 흐름 그래프를 통한 유사도 측정에서 높은 신뢰도를 보여주는 것을 확인할 수 있다.



[그림 4] 자바 프로그램 비교 실험의 ROC 곡선

[Fig. 4] ROC Curve of Comparison Experiments between Java Programs



[그림 5] 자바 프로그램 비교 실험의 정밀도-재현율 곡선

[Fig. 5] Precision-recall Curve of Comparison Experiments between Java Programs

5. 결론

프로그램의 제어 흐름 그래프는 프로그램이 가지는 다양한 특성들을 효과적으로 표현할 수 있는 자료 구조이다. 유사한 프로그램을 비교하기 위해 제어 흐름 그래프는 프로그램이 가지는 동작 특성들을 활용하여 신뢰성 있는 결과를 얻을 수 있다. 본 논문에서는 자바 프로그램의 유사도를 분석하기 위한 방법으로 제어 흐름 그래프의 편집 거리를 응용하는 방법을 제안하였다. 그래프 편집 거리를 유사도 분석에 반영하기 위해

유사 명령어 클래스를 분류하고 편집 거리 매칭에 반영함으로써 유사한 프로그램의 동작 과정에서 공유되는 실행 구조와 명령어의 패턴을 비교에 반영할 수 있었다.

본 논문에서 제안한 유사 명령어 매칭 기반 비교 모델을 평가하기 위해 자바 벤치마크 프로그램에 대해 실험하고 결과의 정확도와 신뢰도를 평가하였다. ROC 분석 및 PR 분석 결과를 통해 유사 명령어에 대한 편집 거리 매칭은 유사한 프로그램의 분류 및 탐지에 효과적인 결과를 보여주었으며, 전체 분류 정확도 98.0%의 높은 정확도를 보여주었다. 향후 본 연구 결과의 정확도를 개선하기 위해 유사 명령어 패턴과 편집 거리에 대한 연구를 수행할 계획이다. 또한, 유사한 구조의 명령어 패턴에 대한 데이터를 분석하고, 다양한 언어에 대한 분석을 통해 유사도 분석 결과를 최적화하는 방법에 대해 연구할 계획이다. 본 논문에서 제안한 방법은 명령어의 편집 거리를 통해 프로그램의 특성과 유사도를 분석하는 방법으로 활용될 수 있을 것이라 기대되고, 향후 소프트웨어 분석에서 개별적인 명령어의 특성을 고려한 소프트웨어 비교 분석에 활용할 수 있을 것이라 기대된다.

6. 감사의 글

이 연구결과물은 2022학년도 경남대학교 학술진흥연구비 지원에 의한 것임

References

[1] Renhard Wilhelm and Dieter Maurer, Compiler Design, Addison-Wesley, (1995)

[2] R. Smelik, A. Rensink, and H. Kastenberg, Specification and Construction of Control Flow Semantics, Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, pp.65-72, (2006)
DOI: http://doi.org/10.1109/VLHCC.2006.45

[3] A. Kapoor and S. Dhavale, Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation, Defence Science Journal, (2016), Vol.66. No.2, pp.138-145.
DOI: https://doi.org/10.14429/dsj.66.9701

[4] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, Analyzing and Detecting Emerging Internet of Things Malware: A Graph-Based Approach, IEEE Internet of Things Journal, Oct. (2019), Vol.6, No.5, pp.8977-8988.
DOI: http://doi.org/10.1109/JIOT.2019.2925929

[5] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion, Control Flow Graphs as Malware Signatures, International Workshop on the Theory of Computer Viruses, Nancy, France (2007)
Available from: https://inria.hal.science/inria-00176235

[6] Halvar Flake, Structural Comparison of Executable Objects, Proceedings of the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), pp.161-173, (2004)
DOI: http://doi.org/10.17877/DE290R-2007

[7] V. Nagarajan, R. Gupta, X. Zhang, M. Madou, and B. de Sutter, Matching Control Flow of Program Versions, IEEE International Conference on Software Maintenance, (2007)
DOI: http://doi.org/10.1109/ICSM.2007.4362621

[8] D. Qiu, J. Sun, and H. Li, Improving Similarity Measure for Java Programs Based on Optimal Matching of Control Flow Graphs, International Journal of Software Engineering and Knowledge Engineering, (2015), Vol.25, No.7, pp.1171-1197.
DOI: http://doi.org/10.1142/S0218194015500229

[9] Bijoyan Das and Sarit Chakraborty, An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation, Computing Research Repository(CoRR), (2018)
DOI: http://doi.org/10.48550/arXiv.1806.06407

[10] A. Isah, A. I. Obasa, and M. Hussaini, Impact of Algorithm Plagiarism Detection using Structural, Block and Sentence Similarities of Control Flow Graphs, Journal of Vocational and Technical Education, November (2015), Vol.6, No.1, pp.109-122.

[11] X. Chen, H. Huo, J. Huan, and J. S. Vitter, An efficient algorithm for graph edit distance computation, Knowledge-Based Systems, (2019), Vol.163, pp.762-775.
DOI: http://doi.org/10.1016/j.knosys.2018.10.002

[12] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, Comparing Stars: On Approximating Graph Edit Distance, International Conference on Very Large Data Bases (VLDB), pp.25-36, (2009)
DOI: http://doi.org/10.14778/1687627.1687631

[13] P. P. F. Chan and C. Collberg, A Method to Evaluate CFG Comparison Algorithms, The 14th International Conference on Quality Software, pp.95-104, (2014)
DOI: http://doi.org/10.1109/QSIC.2014.28

[14] https://docs.oracle.com/javase/specs/, The Java Virtual Machine Specification, Sept 1 (2022)

[15] Open Source Software in Java, https://java-source.net/, Mar 1 (2022)

[16] Zelix KlassMaster, http://www.zelix.com/klassmaster/, Mar 1 (2022)

[17] Jesse Davis and Mark Goadrich, The Relationship Between Precision-Recall and ROC Curves, The 23rd International Conference on Machine Learning, pp.233-240, (2006)
DOI: http://doi.org/10.1145/1143844.1143874

