논문 2024-1-3 http://dx.doi.org/10.29056/jsav.2024.03.03

소스코드의 그래프 변환을 통한 시각화와 그래프 도구 활용에 대한 연구

장석준*, 김수현**, 이임영**†

A Study on Visualization through Graph Conversion of Source Code and Utilization of Graph Tools

Seok-Joon Jang*, Su-hyun Kim**, Im-Yeong Lee**†

요 약

최근 오픈소스 소프트웨어가 증가하면서 오픈소스 라이선스 간 충돌문제와 라이선스 위반이 주요 문제로 떠오르고 있다. 이에 따라 소스코드 분석을 위한 방법 중 하나로 그래프로 변환하여 시각적으로 표현하는 방법이 개발되었다. 하지만 그래프 데이터베이스는 특정 파일 형식만을 지원하기 때문에 전처리 과정이 필수적으로 요구된다. 이에 소스코드를 그래프 변환 시 필요한 전처리 과정을 구현한다. 본 논문에서는 소스코드를 그래프로 변환하여 시각적으로 표현했으며, 그래프 도구 별 분석을 통해 소스코드 분석에 효과적인 도구에 대해 분석했다. 또한, 간단한 예제 소스코드를 그래프 도구로 분석함으로써, 정점과 간선에 표현되는 소스코드 정보에 대해 분석했다.

Abstract

Recently, the increase in open source software has brought to the forefront significant issues, notably the conflict problems among open source licenses and the prevalent challenge of license violations. As a response, a method for source code analysis has been developed, involving the transformation of code into graphs for visual representation. Nonetheless, the limited file format support of graph databases necessitates mandatory preprocessing. This study implements the preprocessing steps required for the graph transformation of source code. In this paper, the source code was converted into a graph and expressed visually, and effective tools for source code analysis were analyzed through analysis of each graph tools. Additionally, by analyzing a simple example source code with a graph tools, I analyzed the source code information expressed in nodes and edges.

한글키워드 : 소스코드 시각화, 그래프 모델링 언어, 그래프 변환, 그래프 도구, 그래프 데이터베이스

keywords : Source code Visualization, GML, Graph transformation, Graph tools, Graph database

* 순천향대학교 소프트웨어융합학과
** 순천향대학교 컴퓨터소프트웨어공학과
† 교신저자: 이임영(email: imylee@sch.ac.kr)
접수일자: 2023.1127. 심사완료: 2023.12.09.
게재확정: 2024.03.20.

1. 서 론

최근 수많은 소스코드가 오픈소스로 공개되면서 오픈소스 활용이 가능한 분야가 확장되고 있

다. 그러나 대다수의 사용자는 오픈소스에 적용된 라이선스를 정확히 이해하기 어렵다. 하나의 소프트웨어에는 여러 오픈소스가 포함되어 있기 때문에 소프트웨어 개발 과정에서 소프트웨어에 포함되는 오픈소스를 분석 및 결합하는 것이 필수적으로 요구된다. 이러한 과정에서 오픈소스 라이선스를 위반하는 사례가 지속적으로 발생하고 있다. 따라서 라이선스 위반 문제를 사전에 방지하는 것이 중요하다. 하지만 소스코드는 외부 참조, 함수 호출 등 다양한 기능을 실행하기 때문에 소스코드만으로는 직관적인 분석이 현실적으로 어렵다.

이러한 문제를 해결하기 위해 다양한 소스코드 분석 도구가 개발되고 있으며, 그 중 하나는 소스코드를 그래프로 시각화하여 표현하는 방법이다. 그래프는 객체와 객체 간의 관계를 시각화하여 도표나 도형 등으로 표현하는 것을 의미하며, 환경이나 목적에 따라 다양한 방식으로 표현이 가능하다. 그래프는 시각화를 통해 특정 데이터 분석 및 다른 데이터와의 관계를 사용자가 이해하기 쉽게 제공할 수 있다. 또한, 그래프 내에 데이터의 세부 정보(변수, 함수 등)를 표현할 수 있기 때문에 하나의 소프트웨어 내부에 존재하는 소스코드 간의 외부 참조나 함수 호출 등의 관계를 직관적으로 표현하여 분석이 가능하다.

하지만, 그래프로 시각화된 소스코드는 해당 시점에만 분석이 가능하다는 문제가 있으며, 지속적인 분석을 위해서는 데이터베이스에 저장하는 것이 필수적으로 요구된다. 그래프로 시각화된 소스코드를 데이터베이스에 저장하기 위해서는 그에 맞는 파일 형식으로 변환하는 과정이 필요하다.

초기 그래프는 다양한 목적에 맞게 다양한 형식으로 표현되었기 때문에, 명확한 표준 형식이 존재하지 않아 상호 간 호환성 문제가 발생했다. 이에 따라 그래프 표준 형식을 만들기 위한 연구가 진행되었고, 1996년 M. Himsolt에 의해 GML(Graph Modelling Language)이라는 표준 형식이 개발되었다. GML은 정점과 간선을 사용하여 간단하게 데이터를 표현하며, 알고리즘 또는 함수를 강조하여 그래프를 시각적으로 나타낼 수 있다는 장점이 존재한다.

또한, GML을 활용 가능한 다양한 그래프 도구가 존재한다. 소스코드를 변환한 그래프는 이러한 도구들을 사용하여 분석이 가능하다. 하지만 대부분의 그래프 도구는 소스코드의 그래프 분석을 목적으로 개발된 것이 아니기 때문에, 소스코드의 목적과 상황에 따라 그래프 도구를 선택하고 분석하는 것이 효과적이다.

본 논문에서는 그래프로 표현된 소스코드를 그래프 데이터베이스에 맞는 파일 형식으로 변환하고 전처리 과정을 통해 효과적인 그래프 저장 및 표현이 가능한 방법을 구현한다. 이를 통해 직관적인 분석이 어려운 소스코드 및 프로젝트 파일을 그래프로 시각화하고, 이를 그래프 데이터베이스에 저장함으로써 사용자의 지속적인 소스코드 분석, 수정 및 변경에 편의성을 제공한다. 추가적으로, 사용자들이 주로 사용하는 그래프 도구인 iGraph, NetworkX, Gephi를 비교하여 각 도구의 특징을 분석한다. 또한, 그래프 도구 중 하나인 iGraph를 사용하여 목적 별로 분류된 소스코드를 그래프로 표현하는 과정에서의 효율성에 대해 분석한다.

2. 관련 연구

2.1 그래프 전처리

2.1.1 GML

GML은 1996년 M. Himsolt가 고안한 그래프 파일 형식으로, 데이터 구조를 그래프로 표현할 수 있다[1-2]. GML은 정점과 간선으로 표현되는

간단한 구조와 이를 확장하기 쉽다는 특징을 가지고 있다.

정점(Node)은 특정 객체를 나타내며, 임의의 길이와 높이를 가진 사각형, 원 등의 도형으로 표현이 가능하다. 또한, 텍스트로 구성된 라벨을 통해 해당 정점에 대한 설명의 추가가 가능하다.

간선(Edge)은 정점과 정점 간의 관계를 나타내는 역할을 하며, 두 정점을 잇는 선으로 표현된다. 간선도 정점과 마찬가지로 텍스트로 구성된 라벨을 통해 해당 간선에 대한 설명을 추가할 수 있다.

GML은 임의의 데이터를 다양한 환경에서 표현하는 것을 주목적으로 하고 있기 때문에 세부적인 항목을 표현하기보다는 데이터들을 통일된 형식으로 표현하는 것이 더 중요하다[3]. GML은 다음과 같이 표현할 수 있다.

- 그래프: $G=(V,E)$
- 정점: $V=(N_V,A,L_V,E)$
- 간선: $E=(N_V,L_E)$

정점은 숫자(N), 속성(A), 라벨(L) 그리고 연결된 간선(E)을 포함한다. 숫자는 그래프 내의 정점에 대한 유일한 값으로, 식별자로 작용한다.

간선은 숫자(N)와 라벨(L)을 포함하며, 숫자는 해당 간선이 연결하는 두 정점의 숫자를 의미하며, 라벨을 통해 두 정점 간의 연결에 대한 설명을 제공할 수 있다.

2.1.2 Code Graph

Code Graph는 통합 개발 환경(IDE, Integrated Development Environment)인 Visual Studio에서 동작하는 외부 도구로, Visual Studio의 마켓플레이스에서 이를 추가적으로 설치하여 사용 가능하다[4]. Code Graph는 C, C++ 등의 프로그래밍 언어로 작성된 소스코드를 그래프로 변환해주는 기능을 제공한다. Code Graph는 다양한 기능을 제공하며, 주요 기능은 표 1과 같다.

표 1. Code Graph 주요 기능

Table 1. main function of Code Graph

번호	기능
1	호출 그래프 시각화
2	클래스 계층 구조 시각화
3	클래스 멤버 시각화
4	변수 시각화
5	프로젝트 구조 시각화
6	파일 등의 그래프 시각화
7	프로젝트 종속성 분석

2.1.3 Neo4J

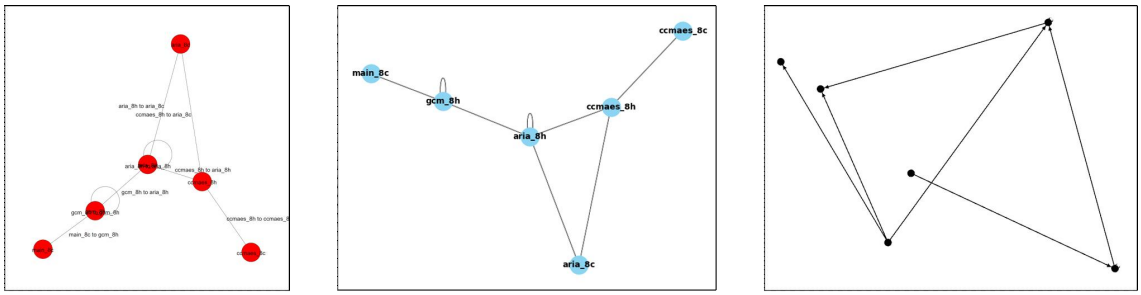
Neo4J는 Neo4J사가 개발한 그래프 데이터베이스 관리 시스템으로, 현재 상용화되어 있는 그래프 데이터베이스 시스템 중에서 가장 널리 사용되고 있다[5]. Neo4J는 간단하면서도 효과적으로 그래프를 표현할 수 있는 Cypher 언어를 사용하여 데이터베이스 내의 데이터를 그래프 형태로 나타낸다.

Neo4J는 Desktop, 브라우저, Bloom, AuraDB 등의 다양한 기능에 따른 인터페이스를 제공하고 있으며, 데이터베이스 내의 그래프를 시각화하기 위한 주요 도구로 Neo4J 브라우저를 활용하고 있다.

2.2 그래프 도구

2.2.1 iGraph

iGraph는 그래프의 생성, 조작 및 네트워크 분석을 위한 라이브러리로, C 언어로 개발되었으며, Python 및 R 패키지를 통해 사용이 가능하다[6, 7]. iGraph는 네트워크 과학 및 관련 분야에서 널리 사용되며, 무료로 이용이 가능하다.



a. iGraph b. NetworkX c. Gephi

그림 1. 그래프 도구 예시
Fig. 1. graph tools example

iGraph는 대규모 네트워크에 적합하며, 대화형과 비대화형 환경을 모두 지원한다는 특징이 있다. 또한, iGraph는 오픈소스로 제공되며, Python 패키지를 활용한 다양한 오픈소스 소프트웨어 패키지도 활용이 가능하다.

2.2.2 NetworkX

NetworkX는 그래프와 네트워크 연구를 위해 개발된 Python 라이브러리이다[8, 9]. 이 라이브러리는 Python 데이터 구조인 '사전' 구조에 높은 의존성을 가지고 있어 네트워크 및 소셜 네트워크 분석에 뛰어난 효율성과 확장성을 제공한다. NetworkX는 다양한 그래프 언어와 파일 형식과 호환되도록 설계되었기 때문에 대규모 그래프 작업에서 사용되지만, 많은 시간이 소요된다는 특징이 있다. 따라서 NetworkX는 주로 소규모 및 중규모 그래프 작업에 적합하다는 평가를 받고 있다.

2.2.3 Gephi

Gephi는 그래프 분석 및 시각화를 목적으로 하는 오픈소스 소프트웨어 도구이다[10, 11]. 독립적인 소프트웨어로 작동하며, 그래프 데이터를 다양한 분야에서 시각화하고 분석하는 데 사용된다. 시각화, 네트워크 분석, 비즈니스 분야에서 활용되며, 다양한 기본 시각화 도구를 제공한다.

각 그래프 도구에 대한 예시는 그림 1과 같다.

3. 소스코드 그래프 변환에서의 전처리

C, C++로 작성된 임의의 소스코드를 Code Graph를 활용하여 그래프로 시각화했을 때, 그래프는 Visual Studio 상에 출력된다. 해당 그래프는 프로젝트 파일 내의 소스코드와 소스코드 내의 함수 및 변수를 정점으로 나타내며, 각 호출 관계는 간선으로 연결된다. 소스코드 내의 모든 함수 및 변수가 시각적으로 표현되며, 호출될 때마다 추가적으로 정점이 생성되기 때문에 동일한 소스코드라도 여러 번 출력되는 특징이 있다. 또한, 시각화된 그래프를 저장할 때는 주로 JSON, XML(Extensible Markup Language) 등의 파일 형식으로 저장된다. 이 때, 호출된 시점에 따라 동일한 변수를 구분하기 위한 문자열이 뒤에 추가되어 저장됨을 그림 2에서 확인할 수 있다.

또한, 결과 파일을 전처리 없이 그래프 데이터베이스에서 호출할 경우 그림 3과 같이 무수히 많은 정점을 가진 그래프가 생성된다.

따라서, 그래프 데이터베이스 상에서는 소스코드와 관련된 핵심 정보만을 표현하기 위한 전처

그림 2. 문자열이 추가된 정점 및 간선
Fig. 2. Vertices and edges with strings added

리 과정이 필수적으로 요구된다. 이 과정은 주로 소스코드를 제외한 변수, 함수 등을 삭제, 소스코드 명 뒤에 생성된 특정 문자열 제거 및 중복된 정점을 제거하는 단계로 이루어지며, 변환되는 전체 구조는 그림 4와 같다.

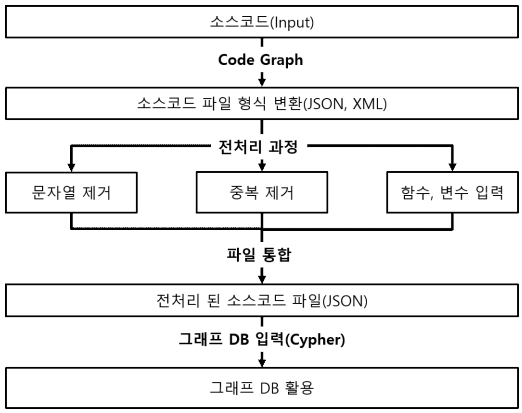


그림 4. 그래프 전처리 전체 구조
Fig. 4. graph preprocessing overall structure

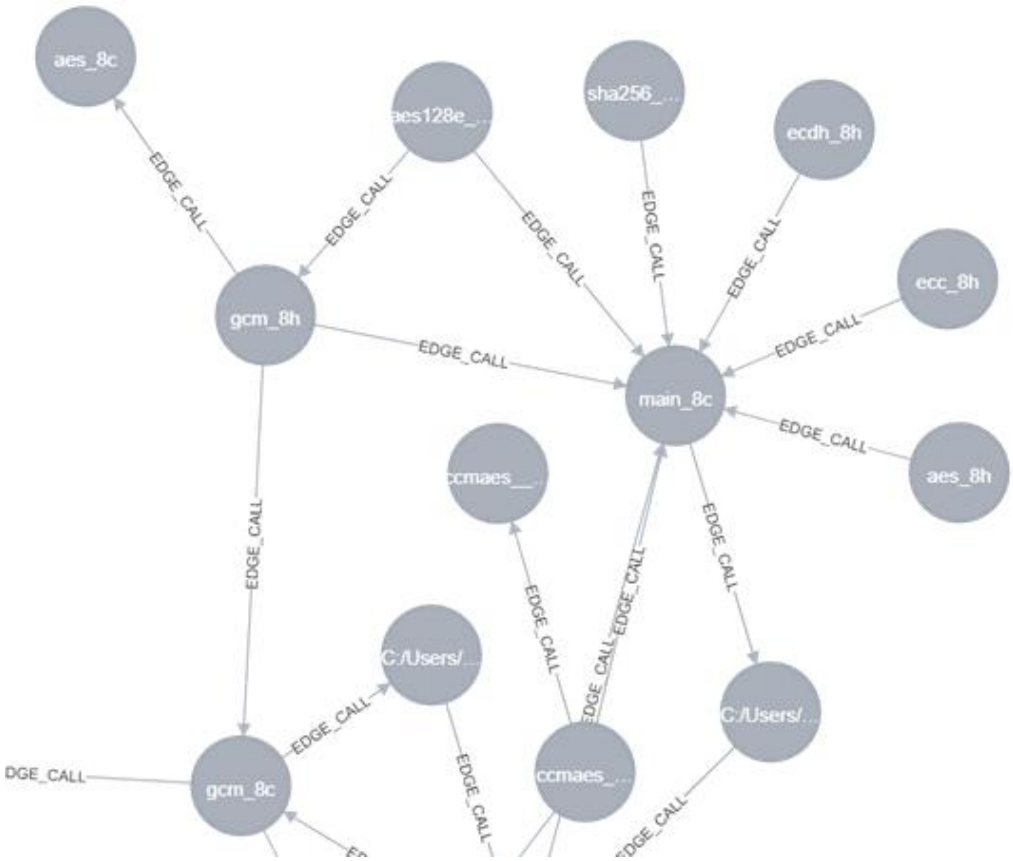


그림 3. 전처리 전의 그래프 결과 중 일부
Fig. 3. some of the graph results before preprocessing

소스코드의 그래프 변환 결과에 대해 이루어지는 전처리는 크게 특정 문자열 제거, 중복 정점 및 간선 제거, 그리고 함수 입력의 3가지로 이루어진다.

3.1 특정 문자열 제거

먼저, 특정 문자열을 제거하는 과정이다. 특정 문자열은 소스코드를 그래프로 변환하는 과정에서 생성된다. 이는 소스코드뿐만 아니라 함수, 변수 등 내부에서 호출이 일어날 때마다 정점이 생성되기 때문에 이를 구분하기 위해서 생성된다. 소스코드를 그래프로 변환한 파일을 분석한 결과, 정점은 다음과 같은 형식의 이름을 갖는다.

'소스코드명_구분_특정 문자열'

여기서 구분은 소스 파일과 헤더 파일을 구분하는 문자이다. 따라서 특정 문자열을 제거하기 위해서는 다음과 같이 '_'를 구분자로 갖는 슬라이싱 기법을 사용할 수 있다.

'["uname"].split('_')'

이를 통해, 구분자인 '_'를 기준으로 문자열을 나눌 수 있으며, 마지막 슬라이싱 된 부분을 삭제할 경우, 특정 문자열의 삭제가 가능하다.

3.2 중복 정점 및 간선 제거

두 번째로, 중복 정점 및 간선을 제거한다. 특정 문자열은 각각 다른 값을 갖기 때문에, 제거하기 전의 식별자는 같은 소스코드여도 다른 정점으로 인식된다. 하지만, 특정 문자열을 제거한 후, 중복되는 정점 및 간선의 구분이 가능하며, 이를 제거하는 작업이 요구된다.

중복 제거의 경우 Python의 자료구조 중 'set'의 특징인 중복이 없다는 점을 이용하여 쉽게 제거할 수 있다. 이를 다음과 같이 list 자료형과 함께 활용함으로써 형식은 유지하면서 중복된 데이터 삭제가 가능하다.

'list(set(map(tuple, data['ItemName'])))'

특정 문자열과 중복 정점 및 간선이 제거된 결과는 그림 5와 같다.

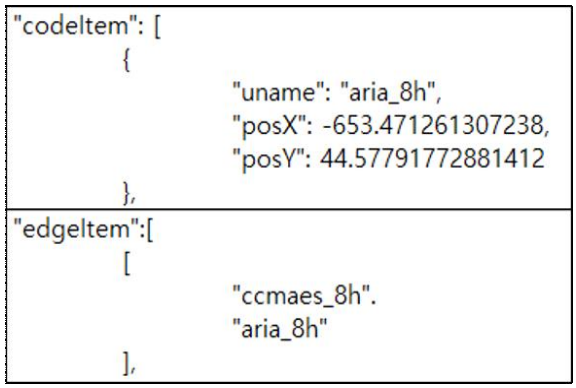


그림 5. 특정 문자열 및 중복 제거 결과
Fig. 5. result of removing specific string and duplicates

3.3 소스코드 내부 함수 입력

마지막으로 각 소스코드에 포함된 함수를 입력하는 과정이다. 소스코드에 포함된 함수는 각각 XML 파일에 별도로 저장된다. 따라서 XML 파일을 파싱하여 JSON에 입력하는 과정을 수행해야 한다. 이는 특정 문자열이 제거된 정점과 XML 파일명을 비교하여 일치할 경우 해당 XML 파일 내부에 존재하는 함수를 JSON 파일의 정점에 추가하는 과정으로 진행된다.

XML 파일 내부에서 함수, 변수 등을 구분하는 구분자는 'memberdef'로, 해당 소스코드의 함수를 구분하기 위해 다음과 같이 memberdef의 function을 찾는다.

'name.attrib['kind'] == "function"

이후, 찾은 함수들을 모두 리스트에 저장, 해당

리스트를 정점에 입력하는 순서로 진행한다. 함수가 추가된 JSON 파일은 그림 6과 같다.

그림 6. 함수가 추가된 JSON 파일
Fig. 6. JSON file with added function

이러한 전처리 과정을 거친 뒤, 해당 파일을 그래프 데이터베이스에 입력한다. Neo4J 그래프 데이터베이스에 입력하며, 입력은 Neo4J에서 사용하는 쿼리 언어인 Cypher 언어를 사용하여 입력한다. 우선 각 정점을 입력한 뒤, 해당 정점에 대해 간선에 저장된 식별자를 정점과 비교하여 일치할 경우 간선을 생성하는 형태로 진행한다. 그래프 데이터베이스에 입력한 결과는 그림 7과 같다.

4. 그래프 도구 비교

위 과정을 통해 도출된, 소스코드를 변환한 그래프는 다양한 그래프 도구를 활용하여 분석이 가능하다[12]. 하지만 대부분의 그래프 도구는 소스코드의 그래프 분석을 목적으로 개발되지 않았기 때문에, 소스코드의 목적과 상황에 따라 적절히 선택하여 분석하는 것이 효과적이다. 본 장에서는 다양한 그래프 도구들 중 많이 사용되고 있는 iGraph, NetworkX, 그리고 Gephi의 3가지 그래프 도구를 비교분석한다.

iGraph는 그래프 이론 및 네트워크 분석에 특

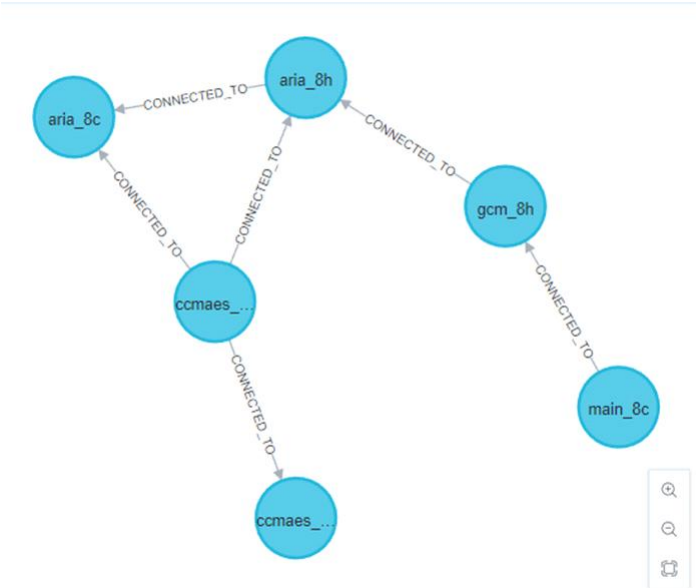


그림 7. 소스코드 그래프의 전처리 결과
Fig. 7. results after preprocessing of source code graph

화된 라이브러리로, C로 개발되어 속도가 빠르고 Python과 R에서 사용 가능하다는 특징이 있다. 그래프 및 네트워크 분석, 복잡한 네트워크 모델링 등의 목적으로 개발되었다. 또한, 그래프 생성 및 조작, 그래프 시각화의 기능을 지원하며, 성능이 좋기 때문에 대규모 그래프에 효과적이다. 라이브러리 형태의 오픈소스로 개발되었기 때문에, 이를 사용한 많은 소프트웨어가 개발, 다양한 함수를 지원한다.

NetworkX는 Python 오픈소스 라이브러리로, 다양한 그래프 언어와 파일 형식과 호환되도록 설계되었기 때문에 속도가 iGraph에 비해 상대적으로 느려 작거나 중간 규모의 그래프 분석에 적합하다. 그래프 및 네트워크 분석, 알고리즘 개발 등의 목적으로 설계되었으며, 그래프 생성 및 조작, 시각화 등의 기능을 제공한다. iGraph와 마찬가지로 라이브러리 형태이기 때문에 다양한 함수의 지원이 가능하다.

마지막으로 Gephi는 Java 기반의 소프트웨어로 네트워크 시각화 및 상호작용에 특화되어 있다. 그래프 및 네트워크를 시각화 하는 것이 가장 큰 목적이기 때문에, 기본적인 조작 기능만을 지원한다.

iGraph, NetworkX, Gephi 세 가지의 그래프 도구를 비교하였으며, 각각에 대한 분석 결과는 표 2와 같다[13-15].

5. 소스코드 그래프 효율성 분석

본 논문에서는 소스코드를 다양한 기준으로 분류하고, 그래프로 시각화하여 효율성에 대한 분석을 진행한다. 이를 위해 그림 8과 같이 기존 전처리된 JSON 형식의 파일을 GML 형식으로 변환 후 그래프 도구에 입력하여 다양한 통계치를 분석한다. 분석 기준에는 그래프 변환 및 분석의 실시간성, 그래프 변환의 정확성 등이 포함된다. 또한, 소스코드가 변환된 그래프와 기존의

표 2. 그래프 도구 비교 결과
Table 2. graph tool comparison results

	iGraph	NetworkX	Gephi
타입	라이브러리	라이브러리	소프트웨어
구현 언어	C	Python	Java
입력 형식	GML, GraphML 등 그래프 언어	GML, GraphML, JSON 등 그래프 언어 및 파일	GML, GraphML 등 그래프 언어
개발 목적	대규모 그래프 분석 및 계산	소규모 그래프 분석 및 계산	시각화 및 상호작용 중심
그래프 기능	많음 (다양한 함수)	많음 (다양한 함수)	적음 (기본적 기능)
라이선스	GPL (General Public License), LGPL(Lesser GPL)	3-Clause BSD (Berkeley Software Distribution)	GPL
소요시간	빠름	느림	- (고려 대상 아님)

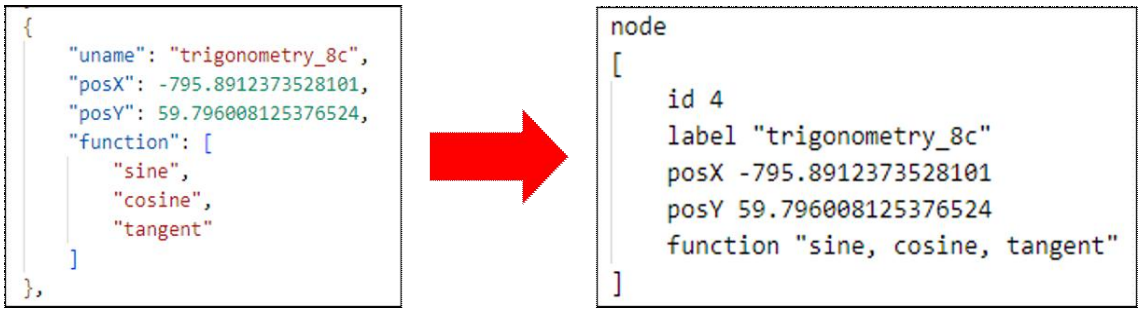


그림 8. JSON 형식의 GML 형식으로의 변환
Fig. 8. conversion of JSON format to GML format

소스코드를 비교하여 소스코드 및 내부 함수의 표현력을 분석한다. 이 과정에서 소요되는 시간 및 메모리 사용량 등의 비교를 통해 그래프 변환의 성능을 평가할 수 있다. 이를 통해 소스코드를 그래프로 표현했을 때의 효율성에 대한 분석이 가능하다.

소스코드는 실시간성이 중요한 주식, CCTV 예제와, 정확성이 중요한 의료 및 간단히 구현된 계산기 예제를 활용했으며, 각 분야 별로 소스코드들의 통계치를 분석한다. 마지막으로, 그래프와 원본 소스코드를 비교하여 그래프의 효율성에 대해 분석한다. 예제 소스코드는 오픈소스 예제들을 활용했으며, 각 분야 별 기본적인 기능만이 구현된 소스코드를 활용했다.

5.1 시스템 환경

소스코드 그래프 효율성 분석은 13th Gen Intel(R) Core(TM) i5-13400F 2.50GHz, 32.0GB RAM, NVIDIA GeForce RTX 3050 환경에서 수행되었다.

5.2 효율성 분석 결과

분석 결과, 소스코드를 그래프로 변환하면 모두 정점으로 나타낼 수 있다. 또한, 소스코드에 포함된 구조체도 일부 정점으로 표현되어, 전체 소스코드 수보다 많은 정점이 생성된다. 소스코드 간의 관계는 간선으로 시각화되어, 간단한 주식 거래 소스코드를 그래프로 표현한 그림 9와 같이 한 눈에 알아보기 쉽게 표현되었다. 이에 따라, 소스코드 간의 관계를 시각적으로 표현함으로써, 기존 소스코드를 그대로 분석하는 것에 비해 소스코드, 소스코드에 포함된 함수, 그리고 소스코드 간의 관계 분석을 효과적으로 수행할 수 있다. 그래프 효율성을 분석한 결과는 표 3과 같으며, 이를 그래프로 표현한 결과는 그림 10과 같다.

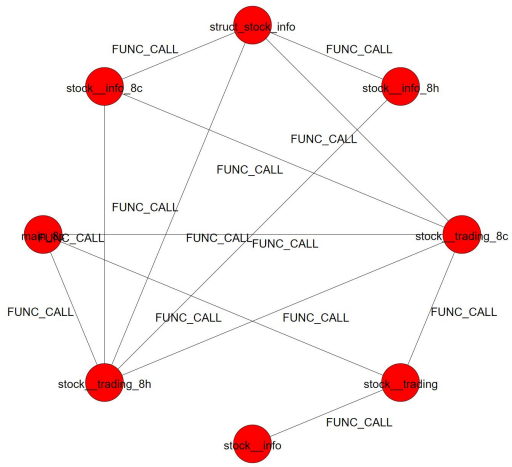


그림 9. 그래프 표현 결과
Fig. 9. graph representation result

표 3. 그래프 효율성 분석 결과
Table 3. graph efficiency analysis results

	주식 거래	CCTV	의료 데이터	계산기
소스코드 합계(개)	12	12	12	12
함수 합계(개)	25	20	18	21
정점 합계(개)	14	12	12	12
간선 합계(개)	32	22	22	18
소요 시간 평균(s)	0.138	0.114	0.022	0.114
메모리 사용량 평균(MB)	0.05	0.036	0.047	0.046

하지만, 소스코드를 그래프로 변환하는 과정에서 소스코드의 모든 내용이 그래프로 변환되는 것은 아니다. 소스코드와 각각의 소스코드 간의 관계는 모두 그래프로 표현되지만, 소스코드 내부에 존재하는 변수, 함수 등의 세부적인 내용은 표현되지 않았다. 따라서 표면적인 부분만을 그래프로 표현하기 때문에 소요 시간은 평균 0.15초를 넘지 않아 거의 실시간성을 보장할 수 있었고, 메모리 사용량 역시 0.05MB를 넘지 않아 경량 환경(IoT 등)에서도 사용 가능할 것으로 판단된다.

6. 결 론

최근 많은 소스코드가 오픈소스로 공개되면서 활용 가능한 분야가 증가하고 있다. 그러나 오픈소스에 적용된 라이선스를 이해하는 것은 어려운 일이며, 소프트웨어 개발 과정에서 라이선스 위반 문제가 지속적으로 발생하고 있다. 이를 방지하기 위해 다양한 소스코드 분석 도구가 개발되었고, 그 중 하나는 소스코드를 그래프로 시각화

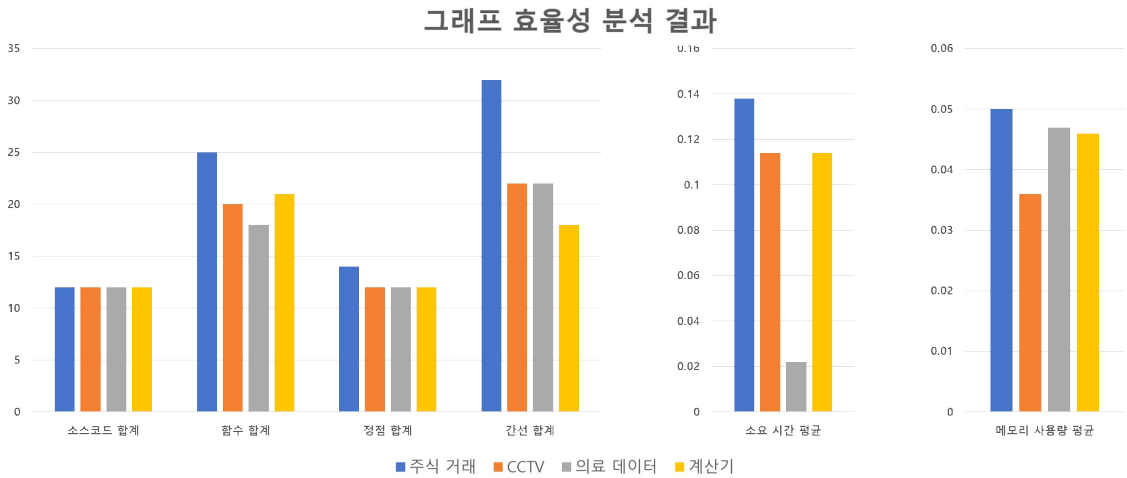


그림 10. 그래프 효율성 분석 결과
Fig. 10. results of graph efficiency analysis

하는 방법이다. 이를 통해 소프트웨어 내부의 관계를 직관적으로 이해할 수 있다. 하지만, 그래프로 시각화된 소스코드는 해당 시점에만 분석이 가능하며, 지속적인 분석을 위해서는 데이터베이스로의 저장이 필수적으로 요구된다. 하지만 데이터베이스에 저장을 위해서는 특정 형식으로의 변환이 요구된다.

또한, 소스코드의 그래프 분석을 목적으로 개발된 그래프 도구는 존재하지 않기 때문에, 소스코드의 환경과 목적에 따라 효과적인 그래프 도구가 다르다. 따라서 상황에 따라 효과적인 그래프를 선택하는 과정이 요구된다.

본 논문에서는 그래프 데이터베이스로의 저장을 위해 GML 형식을 이용하여 전처리 과정을 구현하였다. 특정 문자열 제거, 중복 제거, 그리고 함수 입력의 과정을 통해 그래프 데이터베이스에 입력 가능한 형식으로의 변환을 수행했다.

또한, 소스코드의 그래프 변환과 분석 과정에서 효과적인 그래프 도구의 판단을 위해 상용화되어 있는 그래프 도구들 중 일부(iGraph, NetworkX, Gephi)의 특징과 장단점을 비교분석했다.

이를 통해 소스코드를 그래프 변환 및 저장하는 과정에서 요구되는 전처리 및 형식 지정을 구현하였으며, 환경과 목적에 따라 그래프 도구를 비교분석했다.

또한, 다양한 분야의 소스코드 예제를 그래프 도구에 입력, 소스코드가 그래프로 변환될 때의 데이터 정확성, 소요 시간 및 메모리 사용량을 분석했다. 대부분의 경우 소스코드 명 또는 내부 함수 등 핵심 내용만을 간단하게 그래프로 표현하기 때문에, 그래프 변환 과정에서 시간과 메모리 소모에는 큰 차이가 없다는 결과를 얻었다.

본 연구를 통해 소프트웨어(소스코드, 라이선스 등)의 시각적 분석이 가능하며, 이후 소스코드의 디버깅에서의 활용, 정점을 라이선스로 설정함으로써 라이선스 간 관계를 파악하여 라이선스 충돌 예방 등의 기술 개발에 기초 연구로 활용될 수 있다.

향후 소스코드의 세부적인 내용을 그래프에 표현하는 방법에 대한 연구, 그래프로 시각화된 데이터에 대한 보안적인 조치 또는 적용 가능성에 대한 연구가 필요할 것으로 사료된다. 특히, 소스코드를 그래프로 변환하여 시각화할 경우, 소스코드 내부에 존재하는 데이터가 그대로 표현되기 때문에, 기업 비밀이나 개인정보 등 민감한 데이터가 드러날 수 있어 유출될 경우 심각한 피해를 입을 수 있다. 따라서 그래프에 적용 가능한 보안적인 조치에 대한 연구가 특히 요구된다.

본 연구는 문화체육관광부 및
한국콘텐츠진흥원의 2023년도 SW저작권
생태계 조성 기술개발 사업으로 수행되었음
(과제명 : 클라우드 서비스 활용 구축 형태별
대규모 소프트웨어 라이선스 검증 기술개발,
과제번호 : RS-2023-00224818, 기여율: 100%)

참 고 문 헌

[1] M. Himsolt, GML: Graph modelling language. University of Passau, 1997
[2] M. Himsolt, GML: A portable graph file format. Technical report. University of Passau, 1997
[3] A. Messina, Overview of Standard Graph File Formats, Tech. Rep. RT-ICAR-PA-2018-06.
DOI: 10.13140/RG.2.2.11144.88324
[4] Y. Auyeung, Code Graph, Visual Studio Marketplace, 2021,
https://marketplace.visualstudio.com/items?itemName=YaobinOuyang.CodeAtlas

[5] Neo4J, Neo4J Documentation, Neo4J, 2023, https://neo4j.com/docs/
[6] igraph, python-igraph stable, igraph, 2023, https://python.igraph.org/en/stable/
[7] G. Csardi, T. Nepusz, The igraph software package for complex network research, International Journal of Complex Systems, 1695(5), pp.1-9, 2006,
[8] NetworkX, NetworkX Documentation, NetworkX, 2023, https://networkx.org/documentation/stable/
[9] A. Hagberg, P. Swart, D. S Chult, Exploring network structure, dynamics, and function using NetworkX(No. LA-UR-08-05495; LA-UR-08-5495), United States : Los Alamos National Lab(LANL) & Los Alamos, NM, 2008
[10] Gephi, Gephi-Learn, Gephi, 2023, https://gephi.org/developers/
[11] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating networks, In Proceedings of the international AAAI conference on web and social media, 3(1), pp.361-362, 2009, DOI : 10.1609/icwsm.v3i1.13937
[12] N. Akhtar, Social network analysis tools, In 2014 fourth international conference on communication systems and network technologies, IEEE, pp.388-392, 2014, DOI: https://doi.org/10.1109/csnt.2014.83
[13] S. Wandelt, X. Sun, Complex network analysis: The need for speed, In 2016 35th Chinese Control Conference, IEEE, pp.1213-1218, 2016, DOI: https://doi.org/10.1109/ChiCC.2016.7553252
[14] T. Bonald, N. De Lara, Q. Lutz, B. Charpentier, Scikit-network: Graph analysis in python, The Journal of Machine Learning Research, 21(1), pp.7543-7548, 2020, ISSN: 1532-4435
[15] A. Chaudhary, N. Jain, A. Kumar, Tools for Social Network Analysis and Mining, In 2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART), IEEE, pp.1063-1067, 2022, DOI: https://doi.org/10.1109/SMART55829.2022.10046935

저 자 소 개

장 석 준(Seok-Joon Jang)

2023.2 순천향대학교
컴퓨터소프트웨어공학과 졸업
2023.3-현재 : 순천향대학교
소프트웨어융합학과 석사과정
<주관심분야> 정보보호, 암호학, GML, 그래프 변환

김 수 현(Su-hyun Kim)

2010.2 순천향대학교 정보기술공학부 졸업
2012.2 순천향대학교 컴퓨터학과 석사
2016.2 순천향대학교 컴퓨터학과 박사
2016.3-2018.9 순천향대학교
IoT보안연구센터 연구교수
2018.10-2023.2 정보통신산업진흥원 책임
2023.3-현재 : 순천향대학교
컴퓨터소프트웨어공학과 교수
<주관심분야> 암호 프로토콜, IoT 보안, 클라우드 컴퓨팅 보안

이 임 영(Im-Yeong Lee)

1981.2 홍익대학교 전자공학과 졸업
1986.2 오사카대학 통신공학전공 석사
1989.2 오사카대학 통신공학전공 박사
1989-1994 한국전자통신연구원 선임연구원
1994-현재 : 순천향대학교
컴퓨터소프트웨어공학과 교수
<주관심분야> 암호이론, 암호 프로토콜, 컴퓨터보안