Journal of The Korea Society of Computer and Information
Vol. 21 No. 12, pp. 1-10, December 2016
www.ksci.re.kr
https://doi.org/10.9708/jksci.2016.21.12.001

Software Similarity Measurement based on Dependency Graph using Harmony Search

Ho Yeong Yun*, Yong Joon Joe**, Byung Ok Jung***, Dong myung Shin****, Hyo Keun Bahng*****

Abstract

In this paper, we attempt to prevent certain cases by tracing a history and making genogram about open source software and its modification using similarity of source code. There are many areas which use open source software actively and widely, and open source software contributes their development. However, there are many unconscious cases like ignoring license or intellectual properties infringe which can lead litigation. To prevent such situation, we analyze source code similarity using program dependence graph which resembles subgraph isomorphism problem, a typical NP-complete problem. To solve subgraph isomorphism problem, we utilized harmony search of metaheuristic algorithm and compared its result with a genetic algorithm. For the future works, we represent open source software as program dependence graph and analyze their similarity.

▸ Keyword : Open Source Software, License, Subgraph Isomorphism Problem, Harmony Search

I. Introduction

오픈소스 소프트웨어는 소스코드가 공개되어 있는 소프트웨어로 일반적으로 자유롭게 사용·복제·배포·수정할 수 있다. 상용을 목적으로 하는 소프트웨어에 반하여 누구나 자유롭게 사용하자는 오픈소스 운동은 공유 경제의 대표적 사례로 꼽힌다. 오픈소스 소프트웨어의 활성화는 임베디드, 모바일, 클라우드 컴퓨팅, 빅 데이터 등 모든 ICT 분야에 기여하고 있다.

이러한 오픈소스 소프트웨어의 자유로운 사용의 배경에는 라이선스 의무 조항을 준수한다는 전제가 선행되어 있다. 즉 라이선스 의무 조항을 반드시 준수해야 하는 조건하에 사용·복제·배포·수정이 가능한 조건부 자유인 것이다. 실제 오픈소스 소프트웨어의 저작권을 무시하거나 잘못 사용하여 법정 분쟁이 발생한 사례도 다수 존재한다. 저작권에 따른 여러 문제를 해결하기 위한 일환으로 소프트웨어의 라이선스 및 저작권 정보를 제공하기 위한 SPDX(Software package Data Exchange) 표준이 재정되기도 했지만 활성화가 이루어지지 않는 등 개발자와 사용자들은 여전히 소프트웨어의 저작권 침해 위험에 노출되어 있다.

오픈소스 소프트웨어는 원(root) 개발자가 최초 배포하면 다른 개발자들이 기능을 수정·추가하여 2차, 3차 배포하게 되는데, 이 과정에서 라이선스 정보가 누락·훼손·변경·충돌하는 경우가 존재한다. 이에 본 연구는 오픈소스 소프트웨어간의 유사도 분석을 통해 해당 오픈소스 소프트웨어가 어떤 오픈소스 소프트웨어로 부터 변경되었는지를 파악하여 해당 오픈소스의 라이선스 정보를 개발자에게 제공하는 것을 목표로 하였다.

소프트웨어간의 유사도를 분석하는 연구는 소프트웨어의 표절을 탐지하는 연구와 동일하다. 소프트웨어의 표절을 탐지하는 기법은 Table 1과 같이 크게 4가지로 분류할 수 있다[1].

• First Author: Ho Yeong Yun, Corresponding Author: Ho Yeong Yun
*Ho Yeong, Yun(allen@lsware.com), LSWare Inc., Korea
**Yong Joon Joe (eugene@lsware.com), LSWare Inc., Korea
***Byung Ok, Jung (luca@lsware.com), LSWare Inc., Korea
****Dong myung, Shin (roland@lsware.com), LSWare Inc., Korea
*****Hyo Keun, Bahng (revoice@skku.edu), SungKyunKwan Uni, Korea
• Received: 2016. 10. 06, Revised: 2016. 10. 17, Accepted: 2016. 12. 15.
• This research project was supported by the Ministry of Culture Sports, and Tourism(MCST) and by the korea Copyright Commission in 2016.

Table 1. Software Plagiarism Methods

Category	Speed	Reliability
String based	Fast	Low
Token based	↑	↓
AST based		
Program Dependency based	Slow	High

해당 기법은 소스 코드가 변하지 않은 경우는 빠르게 탐지할 수 있지만, 변수(Variable)나 식별자(Identifier)등이 조금이라도 바뀌는 경우는 탐지해내지 못하는 단점이 있다. 토큰(Token) 기반의 표절 탐지 기법은 국내외 공개된 많은 표절 탐지 도구에 적용된 기법으로, 주석, 공백과 같은 유사도에 영향을 주지 않는 요소를 제외한 소스코드에 대해 토큰 시퀀스를 생성하여 유사도를 측정하는 기법이다. 이는 인위적인 공백 삽입, 식별자 변경, 코드 형식 변경 등을 탐지할 수 있는 장점이 있지만 함수 호출 순서 변경, 중간 코드 삽입 등의 표절 기법은 탐지하기 힘든 단점이 있다. 추상 구문 트리(Abstract Syntax Tree) 기반의 표절 탐지 기법은 소스코드로부터 추상 구문 트리를 생성하여 유사도를 측정하는 기법이다. 문자열 기반과 토큰 기반과는 달리 구조 정보를 가지기 때문에 함수의 위치 변경이나 병합, 소스코드 수정 등을 탐지하는데 장점이 있지만 문장 재정렬에는 취약하다는 단점이 존재한다. 프로그램 종속성 그래프(Program Dependence Graph, PDG) 기반의 표절 탐지 기법은 소스코드의 제어 흐름과 데이터 간의 의존성 정보를 그래프로 표현하여 분석하는 기법이다. 원본 소스코드와 표절 검사의 대상이 되는 소스코드를 그래프로 표현하여 이의 유사도를 비교하는 검사를 그래프 동형(Graph Isomorphism) 문제라고 한다. 본 연구는 신뢰성에 중점을 두어 프로그램 종속성 그래프 기반의 표절 탐지 기법을 통해 오픈소스 소프트웨어의 유사도를 분석하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 표절 탐지에 대한 국내외 연구동향과 그래프 동형 문제에 대해 서술하였다. 3장에서는 그래프 동형 문제를 해결하기 위해 설계한 알고리즘을 기술하고 실험결과를 분석하였다. 4장에서는 본 논문의 결론과 향후 연구에 대하여 서술하였다.

II. Preliminaries

1. Literature Reviews

지적재산권에 대한 중요성이 높아지면서 교육기관의 레포트, 언론사의 뉴스기사, 프로그램 코드 등의 표절을 탐지하는 연구는 현재까지도 이어지고 있다. 본 연구에서는 프로그램 표절 탐지에 대한 선행연구만 다루기로 한다.

김신형[1]은 웹에서 가장 많이 사용되는 스크립트 언어인 자바 스크립트로 작성된 프로그램을 대상으로 표절을 탐지하는 연구를 진행한 바 있다. 표절 탐지 기법으로는 프로그램 종속성 그래프와 이를 사용한 탐지 기법을 제안하였다. 지정훈 외[2]는 자바 프로그램에 대해 표절검사기법을 제시하였다. 소스코드를 이용한 표절검사는 보안문제가 발생할 수 있기 때문에 바이트코드를 이용해 표절검사를 수행하였는데, 실제 소스코드와 바이트코드의 유사도는 비슷한 분포를 보였다. 표절 기법으로는 토큰 시퀀스(Token sequence)를 생성하여 이를 유사도 측정하였다. 김영철 외[3]는 구문 트리를 이용하여 서로 다른 두 개의 C 프로그램의 유사도를 평가하는 알고리즘을 제시하였다. 또한 유사성이 높은 집단을 클러스터링하여 다수의 프로그램을 적은 횟수로 비교할 수 있게 하는 알고리즘을 함께 제안한 바 있다.

Koschke 외[4]는 추상화 구문 트리를 분석하는 기법을 통해 C언어로 작성된 소프트웨어의 표절을 탐지하였다. Liu 외[5]는 프로그램 종속성 그래프를 기반으로 표절을 탐지 하였으며, 이를 그래프 동형문제로 변환하고 소스코드를 전처리하여 해결하는 과정을 서술하였다.

부산대학교에서 개발한 프로그램 표절 탐색 시스템인 SoVAC(Software Verification and Analysis Center)는 토큰 시퀀스를 DNA로 변경한 뒤 지역 정렬(Local alignment) 알고리즘을 이용하여 DNA를 비교하는 것으로 표절을 판단하고 있다[6]. Java, C#, C, C++, Scheme 뿐만 아니라 자연 언어 텍스트의 표절 탐지를 지원하는 JPlag와 스탠포드에서 개발한 MOSS(Measure of Software Similarity), 그 외 SIM, YAP3 또한 토큰 기반 표절 탐지 기법을 적용하였다[7-9].

2. Graph Isomorphism Problem

그래프 이론은 노드들의 집합과 노드 사이의 관계를 표현하여 분석하는 이론으로 물리, 화학, 커뮤니케이션 과학, 컴퓨터 기술, 전기 및 토목 공학, 건축, 유전학, 심리학, 사회학, 경제학, 언어학 등 여러 분야에 응용되고 있다[10].

그래프 동형 문제는 두 그래프가 근본적으로 같은 그래프인지 아닌지를 판별하는 문제이다. Fig. 1에서 두 그래프의 형태는 다른 모습을 띠지만 좌측 그래프의 노드 {1, 2, 3, 4}는 우측 그래프의 노드 {D, B, A. C}에 각각 대응되며, 이때 두 그래프는 동형관계라고 한다.

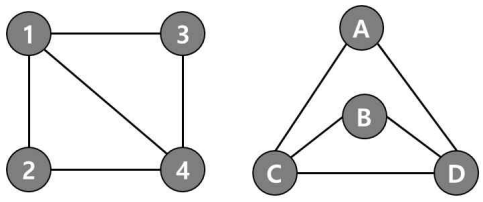


Fig. 1. Examples of Graph Isomorphism

그래프 G와 그래프 H의 노드(Node)의 집합을 각각 $V(G)$, $V(H)$이고, 노드를 잇는 변인 엣지(Edge)의 집합을 각각 $E(G)$, $E(H)$라고 할 때, 그래프가 동형관계이면 아래의 조건을 만족하며, $G \cong H$로 표현한다.

i) $|V(G)| = |V(H)|$이고, $|E(G)| = |E(H)|$이다.

ii) 각각의 연결성분의 개수가 동일하다.
iii) 노드의 차수(Degree Sequence)도 동일하다.

오픈소스 소프트웨어를 표절, 복제, 변경한 경우에는 완벽하게 일치하는 경우는 거의 없기 때문에 본 연구에서는 그래프 동형 문제에서 파생된 부분 그래프 동형(Subgraph Isomorphism) 문제가 더 적합하다. 부분 그래프 동형 문제는 그래프 G와 그래프 H가 $V(G) \subset V(H)$와 $E(G) \subset E(H)$를 만족할 때 그래프 G는 그래프 H의 부분 그래프라고 한다. Fig. 2의 예시를 보면 좌측의 그래프의 노드 {1, 2, 3}은 우측 그래프의 노드 {A, B, C, D, E, F} 중 {A, B, D}에 대응된다. 이때 좌측 그래프는 우측 그래프의 부분 그래프에 해당한다고 할 수 있다.

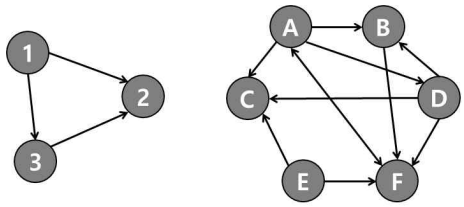


Fig. 2. Examples of Subgraph Isomorphism

부분 그래프 동형 문제는 화학정보학(Cheminformatics) 분야에서 화학 구조를 비교하는데 활용되고 있으며, 그 외에도 데이터베이스의 패턴 분석, 단백질 네트워크 분석(Protein-Protein Interaction Networks), 사회관계망(Social network) 분석 등에 적용되고 있다.

III. Experiments

부분 그래프 동형 문제는 전형적인 NP-Complete 문제로 다항시간 내에 풀 수 있는 알고리즘은 현존하지 않는다. 그렇기 때문에 문제의 크기가 커지는 경우 복잡도 또한 기하급수적(Exponential)으로 커지기 때문에 전역검사를 수행하는 것은 현실적으로 불가능하다. 이러한 경우 문제를 완화(Relaxation)하여 접근하는 방법과 근사값을 구하는 방법이 있는데 전자의 경우는 문제를 완화할 수 있는 조건이 제한적이기 때문에 본 연구에서는 후자의 경우를 채택하였다.

1. Harmony Search

본 연구에서는 근사값을 구하기 위해 메타휴리스틱 알고리즘을 활용하였다. 메타휴리스틱 알고리즘은 자연 현상을 모방한 알고리즘으로 지역해(Local Optimal)를 벗어나 전역해(Global Optimal)로 접근해 나가는 것을 목적으로 하는 알고리즘이다. 유전 알고리즘(Genetic Algorithm), 시뮬레이티드 어닐링(Simulated Annealing), 타부 서치(Tabu Search), 개미 군집 최적화(Ant Colony Optimization), 개체 군집 최적화(Particle Swarm Optimization) 등 현재까지도 매우 다양한 메타휴리스틱 알고리즘이 발표되고 있다. 메타휴리스틱 알고리즘은 어떤 문제에도 디자인이 가능하다는 장점이 있기 때문에 부분 그래프 동형 문제에도 적용 가능하며, 유전 알고리즘과 시뮬레이티드 어닐링으로 접근한 연구들이 존재한다[11-13]. 본 연구에서는 즉흥 연주자들이 화음을 맞춰나가는 과정을 모방한 알고리즘인 화음 탐색법(Harmony Search)을 적용하였다.

화음 탐색법은 해집단 기반(Population based)의 알고리즘으로 5가지의 연산자(Operator)가 존재하지만 실제로 사용되는 연산자는 3개다. 기존에 사용한 음을 그대로 참고하는 Memory considering과 기존에 사용한 음을 조금 변경하여 참고하는 Pitch adjustment, 임의의 값을 참고하는 Randomization이 화음 탐색법의 연산자이다. Fig. 3은 화음 탐색법의 의사코드이다.

2. Experiment Design

2.1. Representation

$V(G) \leq V(H)$일 때, 그래프 G를 하나의 배열로 표현한다. 배열의 인덱스는 그래프 G의 노드를 뜻하며, 배열이 갖는 인덱스의 값은 그래프 H의 노드를 뜻한다. 하나의 배열은 그래프 G가 그래프 V에 어떻게 대응되는지에 대한 정보를 가지고 있다.

2.2. Fitness Function

```
Pseudo Code : Harmony Search
begin
    Objective function f(x), x = (x_1, x_2, ..., x_d)^T
    Generate initial harmonics (real number arrays)
    Define pitch adjusting rate (P_PAR), pitch limits and bandwidth
    Define harmony memory accepting rate (P_HMCR)
    while (not_termination)
        Generate new harmonics by accepting best harmonics
        Adjust pitch to get new harmonics
        if (rand() < P_HMCR)
            if (rand() < P_PAR)
                Adjust the pitch randomly within limits
            else
                Choose an existing harmonic randomly
            end if
        else
            Generate new harmonics via randomization
        end if
        Accept the new harmonics if better
    end while
    Find the current best solutions
end
```

Fig. 3. Pseudo code of the Harmony Search

부분 동형 여부를 판단하는 척도로는 그래프가 대응됐을 때, 간선 개수의 차이와 대응 여부를 다중 목적(Multi-objective)함

수로 정의하였다. $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$가 있고, $|V_1| \le |V_2|$일 때, G_1의 부분그래프인 $G_S=(V_s,E_s)$는 G_2의 부분 그래프라고 정의할 수 있다. 이 때, 첫 번째 목적함수(f_1)인 간선 개수의 차이는 (1)같이 정의된다.

$$f_1 = \sum_{e \in E_1} I(e,E_s) + \sum_{e \in E_S} I(e,E_1)$$
$$where\, I(e,E) = \begin{cases} 0, & if\, e \in E \\ 1, & otehrwise \end{cases} \quad (1)$$

두 번째 목적함수(f_2)인 그래프간 대응 여부는 (2)와 같이 정의된다. $e_{\text{in}-\text{out}}$은 각 그래프의 진입(incoming) 간선과 진출(outcoming) 간선을 뜻한다.

$$f_2 = \sum_{v \in E_1} S(v)$$
$$S(v) = \begin{cases} 0, & \text{if } e_{\text{in}-\text{out}}(G_2(v)) \ge e_{\text{in}-\text{out}}(G_1(v)) \\ 1, & otherwise \end{cases} \quad (2)$$

목적함수 f_1과 f_2는 50%의 비중을 갖는다. 즉 다중 목적함수 $f(w_1,w_2)=w_1f_1+w_2f_2$에서 w_1과 w_2의 값은 0.5가 된다[14].

2.3. Experimental Setup

실험에 사용한 그래프 데이터는 노드의 개수와 엣지의 밀도(density, η)를 고려하여, 난수 데이터를 생성하였다. 노드의 크기가 150개를 갖는 그래프를 기준 그래프로 하였고, 이를 기반으로 10개, 25개, 50개, 75개, 100개의 노드를 갖는 그래프를 생성하였다. 이는 기준 그래프를 축소한 그래프들이기 때문에 모두 부분 그래프이다. 즉 최적해의 값이 0인 데이터이다. 엣지의 밀도는 0.01, 0.05, 0.1, 0.2의 기준으로 150개의 노드를 갖는 그래프를 세분화시켰다. 실험은 총 120개의 데이터 셋으로 진행하였다.

반복 실험은 통계적 유의성을 위해 30회 진행하였다. 화음 탐색법의 파라미터는 HMCR은 0.85, 0.9, 0.95로 PAR은 0.5와 0.7로 총 6가지 경우이며, Pitch Adjustments의 대역폭(Bandwidth)은 부분 그래프의 노드 사이즈의 ±20%, HMS는 10개 ~ 100개 사이에서 노드의 사이즈별로 상이하게 진행하였고 한 실험당 반복횟수(iteration)는 10,000번으로 고정하였다.

부분 그래프 동형 알고리즘은 Visual Studio 2012에서 C# 언어로 구현하였으며, 8GB RAM, i5-6600 3.3GHz의 CPU 환경에서 실험을 진행하였다.

3. Experiment Results

3.1. Experiment Harmony Search

Table 2 ~ Table 6을 통해 상이한 노드와 밀도 그리고 화음 탐색법의 파라미터를 구분하여 노드별로 각각 24개씩, 총 120가지 경우의 실험결과를 나타낸다. 표에서 Best는 반복실험 중에서 목적함수가 가장 낮은 값, Worst는 목적함수가 가장 높은 값, Average는 반복실험의 평균값, STDEV는 반복실험의 표준편차, n of Optimal은 30번의 반복실험 중에 최적값(Global Optimal)인 0에 도달한 횟수를 의미한다. 밀도별로 가장 좋은 수치에 대해서는 강조와 밑줄로 표시하였다.

노드의 크기가 10개일 때는 밀도에 상관없이 모두 최적값에 도달하였는데, 밀도가 0.02와 0.05, 0.1에 대해서는 반복실험의 대부분이 최적값에 도달하였다. 다만 밀도가 0.2일 때는 최적값에 도달한 횟수가 현저히 적어졌고, 최대값과 평균값, 표준편차 또한 다른 밀도에 비해 증가한 결과를 얻었다.

노드의 크기가 25일 때는 밀도가 0.02일 때 모두 최적값에 도달하였고, 0.05일 때 일부의 파라미터에 대해서만 최적값을 만족하였다. 그 외의 밀도에 대해서는 최적값에 도달하지는 못하였지만 최적값과 근사한 값을 보였다.

노드의 크기가 50개, 75개, 100개일 때는 모두 최적값을 찾지 못하였다. 마찬가지로 밀도가 클수록 결과값의 수치가 높았는데 노드의 크기와 밀도가 높을수록 그래프가 복잡해지기 때문에 해의 탐색 영역 또한 증가하여 최적해를 찾지 못하는 것으로 결론지을수 있다. 노드의 사이즈에 비례하게 해의 개수를 늘리고, 반복횟수를 다르게 실험한 결과 해(Solution)가 개선되기도 하였지만, 여전히 사이즈가 큰 경우에 대해서는 지역해(Local Optimal)영역을 벗어나지 못하는 결과를 얻었다. 이는 알고리즘을 보완할 필요가 있음을 시사한다.

3.2. Comparative Experiment

본 연구에서 제시하는 화음 탐색법의 검증을 위해 유전 알고리즘과의 비교실험을 진행하였다. 유전 알고리즘은 확률적 최적화 분야의 대표적인 방법으로 생물의 진화 과정을 모방한 알고리즘이다[16]. 부분 그래프 동형 문제에 적용된 유전 알고리즘은 최자은 외가 진행한 알고리즘을 그대로 구현하였으며, 해당 연구에서 제시한 알고리즘의 설정은 Table 2와 같다.

Table 2. Genetic Framework

Function	Contents
Fitness Function	Same fitness function in this study
Initialization	Random Generation
Selection	Rouletted wheel based proportional selection
Crossover	Cycle crossover
Mutation	Two genes at random and exchanges them
Replacement	worst members of the population
Local Optimization	Two-Vertex Exchange Heuristic
Stopping Condition	find a solution with f_1= 0

위의 설정을 토대로 구현한 유전 알고리즘의 유사코드는 Fig. 4.와 같다. 일반적인 유전 알고리즘에 지역해를 개선하는 함수를 추가한 하이브리드 유전 알고리즘 형태이다. 비교실험에서는 기존 연구의 종료조건을 본 연구와 동일하게 반복횟수 10,000번으로 수정하고, 해를 보정해주는 역할을 하는 Two-Vertex Exchange Heuristic기반의 지역해 개선 알고리즘은 화음 탐색법에 적용하지 않았기 때문에 유전 알고리즘에서도 제외하였다. 기존 연구는 실험

데이터를 랜덤하게 발생시켰는데, 이를 확보할 수 있는 방안이 없어 본 연구의 실험 데이터로 대체하였다. 화음 탐색법의 파라미터와 유전 알고리즘의 파라미터는 개념이 다르기 때문에 정확한 비교가 불

```
Pseudo Code : Hybrid Genetic Algorithm
begin
    Create initial population of fixed size
    while (not_termination)
        for i= 0 to n do
        select parent_1 and parent_2 from population
        offspring_i ← crossover(parent_1,parent_2)
        mutation(offspring_i)
        end for
        replace n chromosomes with n offspring
    end while
return the best chromosome
```

Fig. 4. Pseudo code of the Hybrid Genetic Algorithm

가하다. 이는 유전 알고리즘의 파라미터는 기존연구 그대로 설정하고, 화음 탐색법은 파라미터를 무시하고 밀도 내에서 가장 값이 좋은 Best값, Worst값을 유전 알고리즘의 결과값과 비교하였다.

Table 8. Comparison result of our algorithm with the results of Choi (2012)

		Genetic Algorithm		Harmony Search	
Node	Density	Best	Worst	Best	Worst
10	0.02	0	2.5	**0**	**0**
	0.05	0	1	**0**	**0**
	0.1	0	1	**0**	**0**
	0.2	0	3	**0**	**1.5**
25	0.02	0.5	4.5	**0**	**1**
	0.05	0.5	3.5	**0**	**1.5**
	0.1	1.5	6	**0.5**	**3**
	0.2	3.5	6.5	**1.5**	**4**
50	0.02	2.5	6.5	4.5	8.5
	0.05	4.5	10	6	**9**
	0.1	5	11.5	8.5	**10**
	0.2	8	12	8.5	**11**
75	0.02	5	10.5	14.5	17.5
	0.05	11.5	15.5	14.5	18.5
	0.1	10	14.5	15	20
	0.2	10	16.5	18	21
100	0.02	9.5	16.5	20	24.5
	0.05	12.5	17	22	25
	0.1	13.5	19.5	29	34
	0.2	16.5	21	31	35

유전 알고리즘과 화음 탐색법을 비교분석한 결과값은 Table 8에 기술하였다. 노드의 크기가 10개, 25개일 때는 화음 탐색법의 결과가 우수했지만, 50개, 75개, 100개일 때는 유전 알고리즘의 결과가 더 우수하였다. 화음탐색법이 더 좋은 결과를 보이는 경우에 대해서 강조와 밑줄로 표시하였다. 이러한 결과를 보이는 이유는 각 알고리즘마다 해를 생성하는 매커니즘에 따른 것으로 보인다. 유전 알고리즘은 한 번의 반복마다 다수의 해를 교차하여 여러 해를 생성하는 반면, 화음 탐색법은 한 번의 반복마다 하나의 해를 만들고, 그마저도 좋은 해가 아니면 받아들이지 않는다. 화음탐색법의 이러한 특징은 사이즈가 작은 경우에는 탐색해의 영역이 넓지 않아 속도가 빠르다는 장점이 될 수 있지만, 탐색해의 영역이 큰 경우에는 지역해를 탈출하는 전략이 약하다는 단점이 될 수 있다. 그렇기 때문에 지역해를 탈출할 수 있는 전략을 별도로 수립해야 함을 시사한다. 또한 기존 연구에서 제시한 Two-Vertex Exchange Heuristic 기법과 같은 해를 보정해나가는 알고리즘을 추가한다면 보다 나은 결과값을 도출할 수 있을 것으로 보인다.

IV. Conclusions

본 연구에서는 오픈소스 소프트웨어 간의 유사도를 프로그램 종속성 그래프 기반으로 측정하기 위해, 부분 그래프 동형 문제를 풀이하였다. NP-Complete인 부분 그래프 동형 문제는 다항시간 내에 최적해를 찾는 알고리즘은 존재하지 않으며, 본 연구에서는 효율적인 방식으로 근사해를 찾는 메타휴리스틱 알고리즘을 제시하였다. 실험에 사용된 데이터는 노드가 150개인 기준 그래프와 이를 기반으로 크기가 다른 5개의 부분 그래프를 생성하였다. 노드의 사이즈가 작은 문제에 대해서는 최적해를 찾았지만, 사이즈가 큰 문제에 대해서는 최적해를 찾지 못하였다. 또한 밀도가 높을수록 결과값이 높은 것을 확인할 수 있었다. 기존 연구에서 제시된 유전 알고리즘과의 비교실험에서는 사이즈가 작은 경우에 대해서는 우수한 결과를 보였지만, 노드의 크기가 큰 문제에 대해서는 유전 알고리즘이 더 좋은 결과를 나타냈다. 이러한 결과는 각 알고리즘마다 해를 생성해나가는 방식에서 기인한 것으로 파악된다. 추후에 새로운 해를 취하는 방식을 변경하거나 지역해를 탈출하는 휴리스틱 알고리즘을 추가하는 기법을 활용하여 알고리즘을 보완할 예정이다.

향후 연구로는 더미 데이터가 아닌 실제 오픈소스 소프트웨어의 소스코드를 그래프로 표현하여 이를 알고리즘 적용할 예정이다. 프로그램 개발은 개발자마다 다른 성향을 보이기 때문에 소스코드를 그래프로 표현하기 위해서는 전처리과정이 필수적으로 이루어져야 한다. 또한 알고리즘의 성능을 개선하여 오픈소스 소프트웨어 간의 유사도를 측정하고 이를 기반으로 라이선스가 누락·훼손·변경·충돌되는 경우에 대해 정확한 라이선스 정보를 복원시켜 사용자에게 제공하는 시스템을 구축할 예정이다.

REFERENCES

[1] S.H.Kim, “Plagiarism Detection using Dependence Graph Analysis Specialized for Javascript”, Master

Thesis, Korea Advanced Institute of Science Technology, 2011.

[2] J.H.Ji, G.Woo, H.G.Cho, “A Plagiarism Detection Technique for Java Program Using Bytecode Analysis”, Journal of Korean Institute of Information Scientists and Engineers, Vol. 35, No. 7, pp.442-451, 2008. 07.

[3] Y.C.Kim, S.C.Hwang, J.Y.Choi, “A Program Similarity Evaluation Algorithm”, Journal of Korean Society for Information, Vol. 6, No. 1, pp.51-64, 2005. 02.

[4] Koschke, R., Falke, R, Frenzel, P, "Clone Detection Using Abstract Syntax Suffix Trees", Proceedings of the 13th Working Conference on Reverse Engineering. IEEE, pp. 253-262, 2006.

[5] Liu, C., Chen, C., Han, J., Yu, P. S., "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis" Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 872-881, 2006.

[6] Y.E.Kim, J.S.Cheon, S.W.Byun, G.Woo, “A Parallel Performance Comparison of Haskell Using a Plagiarism Detection Method”, Korea Computer Congress, pp.1724-1726, 2016.

[7] Lutz Prechelt, Guido Malpohl, Michael Philippsen, “Finding Plagiarism among a Set of Programs with JPlag”, Journal of Universal Computer Science, Vol. 8, No. 11, pp.1016-1038, 2002

[8] Gitchell, David, and Nicholas Tran, "Sim : a Utility for Detecting Similarity in Computer Programs." ACM SIGCSE Bulletin. Vol. 31, No. 1. ACM, 1999

[9] Wise, Michael J. "YAP3 : Improved Detection of Similarities in Computer Program and Other Texts." ACM SIGCSE Bulletin Vol. 28, No.1, pp.130-134, 1996

[10] Suresh G. Singh, “Graph Theory”, PHI Learning, 2010.

[11] J.E.Choi, Y.R.Yoon, B.R.Moon. "An efficient genetic algorithm for subgraph isomorphism." Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp.361-368, 2012.

[12] Farahani, M. M., Chaharsoughi, S. K., "A Genetic and Iterative Local Search Algorithm for Solving Subgraph Isomorphism Problem.", Industrial Engineering and Operations Management (IEOM), International Conference on. IEEE, pp.1-6, 2015.

[13] Li, Z., Chen, B., Che, D, "Solving the Subgraph Isomorphism Problem Using Simulated Annealing and Evolutionary Algorithms." Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), pp.293-299, 2016.

[14] J.E.Choi, Y.R.Yoon, B.R.Moon, "An efficient genetic algorithm for subgraph isomorphism.", Proceedings of the 14th annual conference on Genetic and evolutionary computation, 2012.

[15] H.R.Lee, S.H.Shin, K.H.Choi, G.H.Jung, S.K.Park, “Constructing Software Structure Graph through Progressive Execution”, Journal of The Korea Society of Computer and Information, Vol.18, No.7, 2013. 07

[16] J.H.Kim, H.K.Kim, K.H.Jang, J.M.Lee, Y.S.Moon, “Object Classification Method Using Dynamic Random Forests and Genetic Optimization”, Journal of The Korea Society of Computer and Information, Vol.21, No.5, 2016. 05.

Table 3. Experiment Results (Node : 10, Density : 0.02 ~ 0.2, Best, Worst, Average, STDEV, n of Optimal)

Node	Density	HMCR	PAR	Best	Worst	Average	STDEV	n of Optimal
10	0.02	0.95	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	0.5	0.017	0.091	29
		0.9	0.7	**0**	1	0.033	0.183	29
			0.5	**0**	**0**	**0.000**	**0.000**	**30**
		0.85	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	**0**	**0.000**	**0.000**	**30**
	0.05	0.95	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	0.5	0.017	0.091	29
		0.9	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	0.5	0.017	0.091	29
		0.85	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	**0**	**0.000**	**0.000**	**30**
	0.1	0.95	0.7	**0**	**0**	**0.000**	**0.000**	**30**
			0.5	**0**	0.5	0.033	0.127	28
		0.9	0.7	**0**	0.5	0.050	0.153	27
			0.5	**0**	**0**	**0.000**	**0.000**	**30**
		0.85	0.7	**0**	0.5	0.033	0.127	28
			0.5	**0**	0.5	0.033	0.127	28
	0.2	0.95	0.7	**0**	**1.5**	0.617	0.387	4
			0.5	**0**	**1.5**	0.783	**0.364**	1
		0.9	0.7	**0**	**1.5**	0.633	0.370	3
			0.5	**0**	2	0.650	0.375	2
		0.85	0.7	**0**	**1.5**	**0.567**	0.469	8
			0.5	**0**	**1.5**	0.600	0.403	6

Table 4. Experiment Results (Node : 25, Density : 0.02 ~ 0.2, Best, Worst, Average, STDEV, n of Optimal)

Node	Density	HMCR	PAR	Best	Worst	Average	STDEV	n of Optimal
25	0.02	0.95	0.7	**0**	**1**	0.550	0.442	**10**
			0.5	**0**	1.5	0.583	0.437	7
		0.9	0.7	**0**	1.5	**0.467**	**0.346**	7
			0.5	**0**	2	0.617	0.503	8
		0.85	0.7	**0**	1.5	0.567	0.469	8
			0.5	**0**	1.5	0.550	0.461	9
	0.05	0.95	0.7	**0**	2	0.867	**0.307**	2
			0.5	0.5	**1.5**	1.017	0.454	0
		0.9	0.7	**0**	**1.5**	0.817	0.334	2
			0.5	**0**	2	**0.767**	0.504	**4**
		0.85	0.7	0.5	**1.5**	1.017	0.359	0
			0.5	**0**	2	0.883	0.429	1
	0.1	0.95	0.7	1.5	3.5	2.517	0.517	0
			0.5	1.5	4	2.650	0.528	0
		0.9	0.7	1.5	3.5	2.617	**0.370**	0
			0.5	1.5	**3**	**2.383**	0.387	0
		0.85	0.7	**0.5**	3.5	2.433	0.553	0
			0.5	1.5	3.5	2.500	0.509	0
	0.2	0.95	0.7	2	**4**	3.283	0.468	0
			0.5	2.5	4.5	3.517	**0.464**	0
		0.9	0.7	2.5	4.5	3.283	0.552	0
			0.5	2.5	4.5	3.600	0.548	0
		0.85	0.7	2.5	4.5	3.483	0.594	0
			0.5	**1.5**	5	**3.267**	0.666	0

Table 5. Experiment Results (Node : 50, Density : 0.02 ~ 0.2, Best, Worst, Average, STDEV, n of Optimal)

Node	Density	HMCR	PAR	Best	Worst	Average	STDEV	n of Optimal
50	0.02	0.95	0.7	5	**8.5**	6.800	0.837	0
			0.5	5.5	**8.5**	7.017	**0.580**	0
		0.9	0.7	5	**8.5**	7.033	0.830	0
			0.5	**4.5**	**8.5**	6.867	0.850	0
		0.85	0.7	5	9	6.917	0.948	0
			0.5	5	**8.5**	**6.617**	0.944	0
	0.05	0.95	0.7	7	**9**	7.950	0.547	0
			0.5	7.5	9.5	8.067	**0.487**	0
		0.9	0.7	**6**	**9**	**7.783**	0.665	0
			0.5	7	**9**	7.933	0.626	0
		0.85	0.7	6.5	**9**	7.950	0.592	0
			0.5	**6**	**9**	**7.783**	0.762	0
	0.1	0.95	0.7	8	**10**	9.267	0.537	0
			0.5	8.5	10.5	9.483	0.500	0
		0.9	0.7	**7.5**	10.5	**9.183**	0.636	0
			0.5	8	**10**	**9.183**	0.676	0
		0.85	0.7	8	**10**	9.400	**0.498**	0
			0.5	**7.5**	10.5	9.217	0.652	0
	0.2	0.95	0.7	9.5	**11**	10.467	**0.540**	0
			0.5	**8.5**	11.5	10.517	0.663	0
		0.9	0.7	**8.5**	11.5	**10.433**	0.626	0
			0.5	9	12	10.633	0.694	0
		0.85	0.7	9.5	11.5	10.467	0.601	0
			0.5	9.5	11.5	10.583	0.558	0

Table 6. Experiment Results (Node : 75, Density : 0.02 ~ 0.2, Best, Worst, Average, STDEV, n of Optimal)

Node	Density	HMCR	PAR	Best	Worst	Average	STDEV	n of Optimal
75	0.02	0.95	0.7	**14.5**	**17.5**	16.383	0.625	0
			0.5	**14.5**	**17.5**	16.483	0.748	0
		0.9	0.7	**14.5**	**17.5**	16.400	0.700	0
			0.5	**14.5**	**17.5**	**16.333**	0.699	0
		0.85	0.7	15	**17.5**	16.383	**0.486**	0
			0.5	15	**17.5**	16.467	0.656	0
	0.05	0.95	0.7	16.5	**18.5**	17.317	**0.517**	0
			0.5	**14.5**	**18.5**	17.367	0.919	0
		0.9	0.7	15.5	**18.5**	17.367	0.681	0
			0.5	15.5	**18.5**	17.200	0.690	0
		0.85	0.7	15	**18.5**	**17.100**	0.865	0
			0.5	16	**18.5**	17.433	0.763	0
	0.1	0.95	0.7	16	20.5	18.600	1.110	0
			0.5	16.5	**20**	**18.567**	1.023	0
		0.9	0.7	17	21	19.117	**0.907**	0
			0.5	16	20.5	18.917	1.026	0
		0.85	0.7	**15**	20.5	18.783	1.271	0
			0.5	15.5	20.5	18.750	0.998	0
	0.2	0.95	0.7	18.5	21.5	20.283	0.773	0
			0.5	19	**21**	**19.900**	0.578	0
		0.9	0.7	18.5	**21**	20.083	0.617	0
			0.5	18.5	**21**	19.983	0.623	0
		0.85	0.7	**18**	21.5	19.933	0.868	0
			0.5	19	**21**	20.150	**0.528**	0

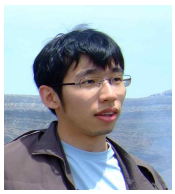
Table 7. Experiment Results (Node : 100, Density : 0.02 ~ 0.2, Best, Worst, Average, STDEV, n of Optimal)

Node	Density	HMCR	PAR	Best	Worst	Average	STDEV	n of Optimal
100	0.02	0.95	0.7	**20**	25.5	23.867	0.964	0
			0.5	23	25	24.150	**0.511**	0
		0.9	0.7	21	25.5	23.883	0.971	0
			0.5	22	25	24.000	0.754	0
		0.85	0.7	22	25	**23.650**	0.778	0
			0.5	22.5	**24.5**	**23.650**	0.658	0
	0.05	0.95	0.7	**22**	**25**	**24.000**	1.035	0
			0.5	29.5	32	30.833	**0.648**	0
		0.9	0.7	29	32	30.567	0.704	0
			0.5	28	32	30.317	0.996	0
		0.85	0.7	29	32	30.400	0.914	0
			0.5	27.5	32	30.467	1.074	0
	0.1	0.95	0.7	30.5	35	32.867	1.033	0
			0.5	30	**34**	32.733	**0.989**	0
		0.9	0.7	**29**	**34**	32.450	1.302	0
			0.5	29.5	**34**	**32.350**	1.138	0
		0.85	0.7	29.5	**34**	32.433	1.165	0
			0.5	30.5	34.5	33.150	1.010	0
	0.2	0.95	0.7	**31**	**35**	33.783	0.926	0
			0.5	**31**	35.5	33.967	0.960	0
		0.9	0.7	33	35.5	34.083	**0.732**	0
			0.5	31.5	36	33.983	0.905	0
		0.85	0.7	32	**35**	33.700	0.890	0
			0.5	31.5	**35**	**33.633**	0.826	0

Authors

Ho Yeong Yun received the B.S. degrees in Industrial Management Engineering from Hansung University, 2012 and Ph.D candidate in Information Industrial Engineering from Yonsei university.

He is currently a special exception for researchers, in the LSWare Inc. for substitute of mandatory military service. His research interests include the optimization, simulation, complexity theory, etc.



Yong Joon Joe received the B.S., M.S. degrees in Information Science from Kyushu University, Japan in 2013, and Ph. D candidate in Information Science from Department of Informatics at Kyushu University.

He is currently a special exception for researchers, in the LSWare Inc. for substitute of mandatory military service. He is interested in game theory for repeated game, distributed constraint optimization problem, AI simulation, etc.

Jung Byung Ok received the B.S., M.S. degrees in computer engineering from Deajon university, Korea, in 2004, 2007 respectively. He did research and development in the DigiCAP Inc by 2016 from 2006. He is currently working for LSWare Inc. His research interests include the BigData Analytics, Cloud Security Service, Network System Security.

Dong Myung Shin received his B.A, M.S and Ph.D degrees in computer engineering from Daejeon university, Korea, in 1997, 2000, 2003, respectively. In 2001 he joined KISA(Korea Information Security Agency), where he worked on Applied Security Technology Team as a member of senior researcher. and he worked on Software Appraisal Team in PDMC(Computer Program Deliberation & Mediation Committee) as a senior researcher. In 2009, He researched on contents filtering of copyright works at the KCC(Korea Copyright Committee) and researched on smart grid security at the KSGI(Korea Smart Grid Institute) in 2014. and Now, he worked on LSWare inc. as a CTO of SW research institute.

His research interests include the system and network security, analysis of software vulnerability and various ICT(Internet Communication Technology) convergence.

Hyo keun Bahng received his B.E. degree in Computer Engineering from Daejeon University, Korea in 2001 and M.E. degree in Computer Engineering from SungKyunKwan University, Korea in 2004. He started his software engineering career at KCC(Korea Copyright Commission) in 2003, where he worked on projects measuring source code similarity(exEyes), managing open source license compliance(CodeEye, OLIS), monitoring and tracing online piracy and developing disk forensic tool. He has been working for KCOPA(Korea Copyright Protection Agency) since transferred from KCC in Sep 2016. He is interested in Computer System and Network Security, Database System, Digital Forensics and Tools, eDiscovery and so on.