ต้นไม้วิวัฒนาการและกฎสำหรับการระบุต้นตอของการลอกเลียนซอร์สโค้ด

Phylogenetic Trees And A Rule For Identifying Original Source Codes

ณัฐนนท์ ลีลาตระกูล[1], สุนิสา ริมเจริญ[2]

Nutthanon Leelathakul[1], Sunisa Rimcharoen[2]

Received: 25 December 2018 ; Revised : 18 January 2019 ; Accepted: 8 February 2019

บทคัดย่อ

การลอกเลียนวรรณกรรม (Plagiarism) ถือเป็นการทำผิดทางจริยธรรมที่ร้ายแรง หากไม่มีการป้องกันหรือตรวจจับที่ดีแล้ว งานสร้างสรรค์ใหม่ ๆ จะเกิดขึ้นน้อย ทำให้เป็นปัญหาต่อการศึกษาและการวิจัยของประเทศ การจัดการเรียนการสอนวิชาการเขียนโปรแกรมก็ประสบปัญหานี้อย่างมาก กล่าวคือ นักเรียนมีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุดคือ การคัดลอกโปรแกรมของเพื่อนแล้วมาส่งอาจารย์ ปัญหานี้จะต้องรีบถูกแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสม ดังนั้น งานวิจัยนี้จึงนำเสนอวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ด เพื่อให้ทราบถึงแหล่งที่มาของการลอกโปรแกรมในกลุ่มนักเรียนนักศึกษา และเมื่ออาจารย์ผู้สอนทราบว่านักเรียนคนใดเป็นต้นฉบับ คนใดที่เป็นคนลอกเพื่อนมาส่ง อาจารย์จะได้วางแนวทางและมาตรการแก้ไข ตักเตือน และแก้ปัญหาได้ตั้งแต่ต้น งานวิจัยนี้ได้นำเสนอขั้นตอนวิธีในการระบุต้นฉบับของซอร์สโค้ดโดยใช้กฎที่สร้างขึ้นจากขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l การหาต้นฉบับเริ่มจากสร้างต้นไม้แสดงวิวัฒนาการของการคัดลอกซอร์สโค้ดด้วยวิธีการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด (Maximum spanning tree) แล้วจึงใช้กฎที่นำเสนอเพื่อระบุทิศทางของลูกศรในต้นไม้เพื่อแสดงลำดับการสืบทอด จากการทดลองพบว่ากฎที่ได้จากขั้นตอนวิธีที่นำเสนอมีความถูกต้องในการระบุต้นฉบับ 90.24%

คำสำคัญ: ขั้นตอนวิธีเชิงวิวัฒนาการ ต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด การลอกเลียนวรรณกรรม ซอร์สโค้ด

Abstract

Plagiarism is deemed a serious ethical offense. Without proper protection and accurate detection algorithms, the number of innovations would significantly dwindle, impeding the advancement of national research and education. Especially in programming courses, plagiarism is extremely common. Most students tend to do their homework in the simplest way: copying the source code of others. Such unpleasant issues must be addressed to preclude this misbehavior before it becomes students' habit, and to promote positive attitudes in our juveniles. In this paper, we, propose an algorithm to identify original source codes, helping instructors to pinpoint whom to monitor and be able to set measures or correctly warn each wrong-doing student. This work introduces a rule, derived from a m+l evolutionary algorithm, for identifying which source codes are original. First, we run the maximum spanning tree algorithm to generate a phylogenetic tree, representing the relationships and sequences of plagiarisms.Then, we apply our proposed rule to identify the source and its copies, whose relations are represented by arrows' directions. The experiment result shows that the proposed rule yields the accuracy of 90.24%.

Keywords: Evolutionary algorithm, Maximum spanning tree, Plagiarism, Source code

[1,2] *ผู้ช่วยศาสตราจารย์ คณะวิทยาการสารสนเทศ มหาวิทยาลัยบูรพา, อำเภอเมือง จังหวัดชลบุรี 20131*

[1,2] *Assisitant Professor, Faculty of Informatics, Burapha university, Muang District, Chonburi20131 , Thailand.*

Email: nutthanon@buu.ac.th, rsunisa@buu.ac.th

บทนำ

ปัญหาการลอกเลียนวรรณกรรม (Plagiarism) เป็นปัญหาสำคัญระดับชาติ ทรัพยากรมนุษย์จะไม่สามารถพัฒนาแบบก้าวไกลได้เลย หากไม่พยายามอย่างที่สุดที่จะคิดสร้างสรรค์นวัตกรรมของตนเอง อีกทั้งยังเป็นปัญหาต่อการศึกษาและการวิจัยที่ฝังรากลึกเป็นระยะเวลายาวนาน โดยเยาวชนรุ่นใหม่มีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด ซึ่งหนึ่งในวิธีเหล่านั้นคือ การคัดลอกผลงานของคนอื่น และนำมาอ้างว่าเป็นของตน ดังตัวอย่างในงานวิจัยของกิตติยา[1] ที่ได้ศึกษาความสำคัญของการป้องกันการโจรกรรมทางวิชาการ ดังนั้น การขโมยความคิดของผู้อื่นมาเป็นของตนเองเป็นปัญหาร้ายแรงที่ส่งผลต่อความเจริญก้าวหน้าและชื่อเสียงของประเทศในระยะยาว โดยเฉพาะอย่างยิ่งถ้าปัญหานี้เกิดขึ้นกับกำลังสำคัญของชาติในอนาคต นั่นคือ นักเรียนนักศึกษา

การลอกเลียนวรรณกรรมในกลุ่มนักเรียนนักศึกษาจึงเป็นเรื่องเร่งด่วนที่ต้องมีมาตรการมาแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสมซึ่งอาจก่อให้เกิดปัญหาใหญ่ที่ร้ายแรงตามมาได้ การแก้ปัญหาที่จุดเริ่มต้นในวัยเยาว์ จะเป็นการปลูกฝังทัศนคติที่ถูกต้องให้กับเยาวชนของชาติซึ่งจะเติบโตไปเป็นกำลังสำคัญของประเทศและจะนำไปสู่การพัฒนาประเทศอย่างยั่งยืน

การผลิตบัณฑิตทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์เพื่อตอบสนองอุตสาหกรรมการผลิตซอฟต์แวร์เป็นหนึ่งในกลไกสำคัญในการขับเคลื่อนยุทธศาสตร์และนโยบายของชาติในการสร้างนวัตกรรม การจัดการเรียนการสอนวิชาการเขียนโปรแกรมจึงเป็นหัวใจสำคัญของการสร้างบัณฑิตเพื่อให้จบออกไปประกอบอาชีพทางการผลิตซอฟต์แวร์ แต่ดังที่ได้กล่าวไปก่อนหน้า ปัญหาของเยาวชนรุ่นใหม่ คือ นักเรียนนักศึกษามีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด โดยการคัดลอกโปรแกรมของคนอื่นมาส่งอาจารย์ ปัญหานี้จะต้องรีบถูกแก้ไข ดังนั้น งานวิจัยนี้จึงนำเสนอวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ด (Source code) เพื่อให้ทราบถึงแหล่งที่มาของการลอกโปรแกรมในกลุ่มนักเรียนนักศึกษา และเมื่ออาจารย์ผู้สอนทราบว่านักเรียนคนใดเป็นต้นฉบับ คนใดที่เป็นคนลอกเพื่อนมาส่ง จะได้วางแนวทางและมาตรการแก้ไข ตักเตือน และแก้ปัญหาได้ตั้งแต่ต้น

ในบทความวิจัยนี้ ผู้วิจัยได้นำเสนอวิธีการในการระบุว่าซอร์สโค้ดใดเป็นต้นฉบับ และได้สร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน โดยต้นไม้นี้ถูกสร้างขึ้นจากการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด และผู้วิจัยได้ระบุทิศทางของลูกศรเพื่อแสดงว่าซอร์สโค้ดใดเป็นต้นฉบับโดยอาศัยกฎที่ได้จากขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l ซึ่งจะกล่าวรายละเอียดต่อไป

การคัดลอกซอร์สโค้ด

ซอร์สโค้ด (Source code) คือ ชุดคำสั่งทางซอฟต์แวร์ที่ถูกเขียนขึ้นเพื่อเรียบเรียงขั้นตอนการแก้ปัญหาให้คอมพิวเตอร์ประมวลผลตามคำสั่งที่ระบุ วิชาการเขียนโปรแกรมถือเป็นวิชาพื้นฐานที่สอนให้ผู้เรียนเข้าใจวิธีการเขียนชุดคำสั่งเหล่านี้ นิสิตนักศึกษาที่เรียนทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์จะต้องเรียนวิชาการเขียนโปรแกรมเพื่อจบออกไปประกอบอาชีพในอนาคต ในการเรียนการสอนวิชานี้ ผู้เรียนจำเป็นจะต้องฝึกหัดเขียนโปรแกรมเพื่อแก้ปัญหาที่หลายหลาย เป็นการฝึกทักษะกระบวนการคิดแก้ปัญหา การเรียบเรียงคำสั่งสำหรับเขียนโปรแกรมออกมาเป็นลำดับขั้นตอน ซึ่งกระบวนการฝึกฝนนี้เป็นหัวใจสำคัญที่จะทำให้ผู้เรียนมีความรู้ความสามารถในการประกอบอาชีพโปรแกรมเมอร์เมื่อจบการศึกษา

แต่ในปัจจุบัน มีผู้เรียนจำนวนหนึ่งที่ไม่พยายามใช้ความสามารถของตัวเอง แต่จะพยายามหาวิธีที่ง่าย ๆ ที่ไม่ถูกต้องเพื่อที่จะทำให้ตนเองมีงานส่งอาจารย์ นั่นคือการลอกเลียนโปรแกรมของคนอื่นมาส่ง การทำเช่นนี้อาจทำให้ติดเป็นนิสัยและอาจเกิดผลเสียหายร้ายแรงได้ในระยาว ดังนั้น จึงต้องมีมาตรการในการแก้ไขตั้งแต่ในตอนเริ่มแรก เพื่อเป็นการปลูกฝังจิตสำนึกให้กับเยาวชน และเป็นการเสริมสร้างแนวความคิดในการสร้างสรรค์ผลงานด้วยตนเอง

ที่ผ่านได้มีความพยายามในการคิดค้นวิธีการในการตรวจจับความคล้ายกันของซอร์สโค้ด Agrawal และ Sharma[2] ได้ศึกษาวิธีการตรวจจับการลอกเลียนซอร์สโค้ดจากงานวิจัยต่าง ๆ และได้แบ่งเทคนิคเป็น 5 กลุ่มใหญ่ ๆ ได้แก่ 1) การตรวจจับในระดับข้อความ (String) 2) การตรวจจับในระดับหน่วยคำ (Token) 3) การตรวจจับในระดับต้นไม้ไวยากรณ์ (Parse tree) 4) การตรวจจับในระดับกราฟความขึ้นต่อกัน (Program dependency graph) และ 5) การตรวจจับโดยใช้ค่าจากมาตรวัดต่าง ๆ (Matrices) เช่น จำนวนลูป จำนวนเงื่อนไขในโปรแกรม

นักวิจัยจำนวนหนึ่งได้พัฒนาขั้นตอนวิธีในการตรวจจับความคล้ายกันของซอร์สโค้ดโดยตรวจจับที่ระดับต่าง ๆ เช่น Castro Campos และ Zaragoza Martinez[3] ใช้ขั้นตอนวิธีการหาลำดับร่วมแบบยาวสุด (LCS: Longest common subsequence) ในการตรวจจับโค้ดที่คล้ายกัน โดยพิจารณาที่ระดับของข้อความหรือตัวอักษรต่าง ๆ ที่ปรากฏในโปรแกรม ซึ่งแม้ว่าวิธี LCS นี้ปกติจะใช้เวลานานในการเปรียบเทียบ แต่บทความวิจัยนี้ก็ได้นำเสนอเทคนิคในการปรับปรุงวิธีการค้นหาให้เร็วขึ้นด้วยการใช้การค้นหาแบบแนวกว้างโดยเทคนิค

การทำซ้ำ (Iterative breadth-first search) Son *et al.*[4] นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Parse tree kernel) ซึ่งมีความสามารถในการตรวจโครงสร้างของโปรแกรม ซึ่งให้ประสิทธิภาพถึง 93% Zhao *et al.*[5] ก็นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Abstract syntax tree) ซึ่งเป็นการวิเคราะห์โครงสร้างของโปรแกรม แล้วเปรียบเทียบค่าแฮช (Hash value) ของโปรแกรมว่ามีความคล้ายกันหรือไม่ Kamalim[6] ใช้การตรวจจับในระดับไบต์โค้ด (Byte code) ซึ่งการตรวจจับแบบนี้มีข้อดีคือ ขั้นตอนวิธีจะพิจารณาจากโค้ดที่คอมไพล์แล้ว ซึ่งจะไม่สนใจเครื่องหมายวรรคตอน (Whitespace) และการตั้งชื่อตัวแปรที่ต่างกันก็จะไม่ส่งผล Qinqin และ Chunhai[7] ได้รวบรวมนำเสนอมาตรวัดความคล้ายกันของซอร์สโค้ดต่าง ๆ เช่น การนับจำนวนแอททริบิว (Attribute counting method) การวัดโครงสร้างของโปรแกรม (Structural measurement method) การจัดกลุ่ม (Clustering method) เป็นต้น

ล่าสุดในปี 2018 Schneider *et al.*[8] ได้นำเสนอแนวทางใหม่ในการตรวจจับการคัดลอกซอร์ส โค้ดที่แตกต่างจากงานวิจัยที่มีมาก่อนหน้า กล่าวคืองานวิจัยในอดีตที่ผ่านมาจะตรวจจับจากโปรแกรมที่เขียนเสร็จแล้ว แต่งานวิจัยใหม่นี้จะเก็บข้อมูล (Log) จากการเขียนโปรแกรมของนักเรียนในชั้นเรียน ตลอดกระบวนการเขียนโปรแกรม ซึ่งการใช้ข้อมูลจากพฤติกรรมในการเขียนโปรแกรมนี้สามารถตรวจจับการลอกกันของนักเรียนได้แม่นยำประมาณ %90 อย่างไรก็ตามการใช้วิธีการนี้ยังต้องการการปรับปรุง เนื่องจากประสบกับปัญหาข้อมูลปริมาณมาก และข้อมูลพฤติกรรมการเขียนโปรแกรมมีความซับซ้อนยากต่อการวิเคราะห์

จากขั้นตอนวิธีต่าง ๆ ที่นักวิจัยจำนวนมากได้นำเสนอก็มีผู้นำมาพัฒนาเป็นเครื่องมือสำหรับตรวจจับการคัดลอกซอร์สโค้ด หนึ่งในนั้นคือเครื่องมือที่ชื่อว่า MOSS (Measure Of Software Similarity)[9] ซึ่งเป็นโปรแกรมตรวจจับการคัดลอกที่ถูกพัฒนาขึ้นที่มหาวิทยาลัยสแตนฟอร์ต โปรแกรมที่พัฒนาขึ้นมีการให้บริการผ่านเว็บ และรองรับภาษาโปรแกรมหลายภาษา ได้แก่ C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, HCL2 ซึ่งในงานวิจัยนี้จะใช้โปรแกรม MOSS ในการเปรียบเทียบความคล้ายคลึงกันของซอร์สโค้ด ก่อนที่จะใช้ขั้นตอนวิธีที่นำเสนอในบทความวิจัยนี้มาทำการระบุว่าซอร์สโค้ดใดเป็นต้นฉบับ โดยขั้นตอนวิธีที่ MOSS ใช้ในการเปรียบเทียบความคล้ายกันของซอร์สโค้ดจะเป็นการตรวจจับในระดับข้อความ (Token) กล่าวคือ MOSS จะวิเคราะห์หน่วยคำในโปรแกรมที่นำมาเปรียบเทียบ และระบุคู่ของโปรแกรมที่มีส่วนเหมือนกันมากของซอร์สโค้ด

ขั้นตอนการใช้ MOSS ในการตรวจจับการคัดลอกซอร์สโค้ดทำได้โดย ดาวน์โหลดโปรแกรม MOSS มาที่เครื่องคอมพิวเตอร์ แล้วรันคำสั่งในการตรวจ โดยระบุไฟล์หรือโฟลเดอร์ที่ต้องการเปรียบเทียบ แล้วซอร์สโค้ดที่จะทำการเปรียบเทียบเหล่านี้จะถูกอัพโหลดไปที่เซฟเวอร์ของ MOSS แล้วผู้ใช้จะได้ URL ในการเปิดดูผลลัพธ์ ซึ่งจะแสดงผลการเปรียบเทียบคู่ของไฟล์ที่มีความคล้ายกัน บอกเปอร์เซ็นความคล้าย และบรรทัดที่คล้ายกัน ดังตัวอย่างใน Figure 1 โดยผู้ใช้สามารถคลิ๊กที่ลิงค์ของแต่ละคู่ไฟล์เพื่อเข้าไปดูรายละเอียดของซอร์สโค้ด ซึ่งจะมีสีแสดงส่วนของโค้ดที่คล้ายกันดังตัวอย่างใน Figure 2


Moss Results

Wed Dec 12 02:16:03 PST 2018

Options -l c -m 10

java

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
../TestSetPlagiarism/MiniFactorial.java (88%)	../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (88%)	8
../TestSetPlagiarism/MiniFactorial.java (88%)	../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (88%)	11
../TestSetPlagiarism/MiniFactorial.java (88%)	../TestSetPlagiarism/MiniFactorial-T1-Comments.java (88%)	23
../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (88%)	../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (88%)	11
../TestSetPlagiarism/MiniFactorial-T1-Comments.java (88%)	../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (88%)	23
../TestSetPlagiarism/MiniFactorial-T1-Comments.java (88%)	../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (88%)	23
../TestSetPlagiarism/MiniFactorial-Copy.java (88%)	../TestSetPlagiarism/MiniFactorial.java (88%)	8
../TestSetPlagiarism/MiniFactorial-Copy.java (88%)	../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (88%)	8
../TestSetPlagiarism/MiniFactorial-Copy.java (88%)	../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (88%)	11
../TestSetPlagiarism/MiniFactorial-Copy.java (88%)	../TestSetPlagiarism/MiniFactorial-T1-Comments.java (88%)	23
../TestSetPlagiarism/MiniFactorial.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	8
../TestSetPlagiarism/MiniFactorial.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-DummyMethod.java (67%)	8
../TestSetPlagiarism/MiniFactorial-T3-DummyMethod.java (67%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	8
../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	8
../TestSetPlagiarism/MiniFactorial-T2-RenameAll.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-DummyMethod.java (67%)	8
../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	8
../TestSetPlagiarism/MiniFactorial-T1-Whitespaces.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-DummyMethod.java (67%)	8
../TestSetPlagiarism/MiniFactorial-T1-Comments.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	21
../TestSetPlagiarism/MiniFactorial-T1-Comments.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-DummyMethod.java (67%)	21
../TestSetPlagiarism/MiniFactorial-Copy.java (79%)	../TestSetPlagiarism/MiniFactorial-T3-LoopEndpoint.java (76%)	8

Figure 1 MOSS results

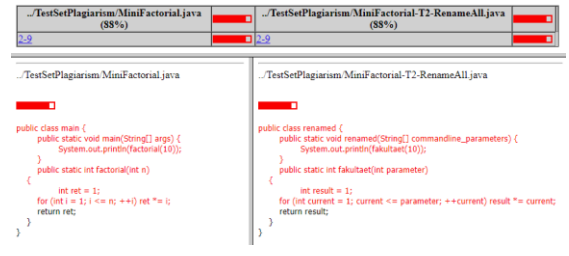

Figure 2 Plagiarized program

ต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด

ต้นไม้แผ่ทั่วของกราฟ (Spanning tree) คือ กราฟย่อยที่ทุกโหนดมีการเชื่อมต่อกันและไม่มีวงวน ซึ่งโดยทั่วไปนักคอมพิวเตอร์จะคุ้นเคยกับต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมน้อยสุด (Minimum spanning tree) ซึ่งสามารถหาต้นไม้นี้ได้จากขั้นตอนวิธีของพริม (Prim's algorithm) หรือ ขั้นตอนวิธีของครัสคัล (Kruskal's algorithm) รายละเอียดของขั้นตอนวิธีสามารถอ่านได้จากหนังสือเรียนวิชาโครงสร้างข้อมูลและอัล

กอริทึมทั่วไป เช่น หนังสืออัลกอริทึม (Sedgewick, 2011)[10] โดยหลักการคร่าว ๆ คือ ใช้ขั้นตอนวิธีเชิงละโมบ (Greedy algorithm) ในการเลือกเส้นเชื่อมที่มีค่าน้ำหนักน้อยที่สุดที่ไม่ทำให้เกิดวงวนในกราฟ โดยเลือกเส้นเชื่อมทีละเส้นจนทุกโหนดในกราฟมีเส้นเชื่อมถึงกันทั้งหมด

แต่ในงานวิจัยนี้ต้องการหาความสัมพันธ์เชื่อมโยงของซอร์สโค้ดที่มีความคล้ายกันมากที่สุด จึงจะใช้การหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดแทน (Maximum spanning tree) ซึ่งวิธีการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดสามารถใช้ขั้นตอนวิธีของครัสคัลได้โดยที่กำหนดให้ค่าน้ำหนักของเส้นเชื่อมแต่ละเส้นเป็นค่าติดลบแทน[11]

ขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l

ขั้นตอนวิธีเชิงวิวัฒนาการ เป็นขั้นตอนวิธีที่ถูกคิดค้นขึ้นโดยได้รับแรงบันดาลใจจากหลักการวิวัฒนาการของสิ่งมีชีวิตในธรรมชาติ ขั้นตอนวิธีรูปแบบนี้ใช้วิธีการค้นหาคำตอบโดยสร้างตัวเลือกที่อาจจะเป็นคำตอบได้ขึ้นมาจำนวนหนึ่ง (เรียกว่าประชากร) ตัวเลือกของคำตอบเหล่านี้จะถูกประเมินว่าคำตอบใดมีค่าความเหมาะสม (Fitness value) ที่ดีกว่ากัน ตัวเลือกที่ดีกว่าจะมีโอกาสอยู่รอด หรือได้เป็นต้นแบบในการสร้างตัวอย่างในรุ่นถัดไป ขั้นตอนวิธีนี้เลียนแบบการคัดสรรทางธรรมชาติ ที่มีกฎว่าสิ่งมีชีวิตที่แข็งแรงกว่าและปรับตัวเข้ากับสิ่งแวดล้อมได้ดีกว่าจะได้สืบพันธุ์และอยู่รอดต่อไป ทำให้ตัวอย่างคำตอบที่สร้างขึ้นมามีการวิวัฒนาการไปสู่คำตอบที่ดีขึ้น กระบวนการสร้างและปรับปรุงตัวอย่างคำตอบนี้จะถูกดำเนินการไปจนครบจำนวนรอบที่ผู้ใช้กำหนดหรือจนกว่าจะพบคำตอบที่ผู้ใช้พึงพอใจ

งานวิจัยได้นี้ ได้ใช้ขั้นตอนวิธีเชิงวิวัฒนาการรูปแบบหนึ่ง ที่เรียกว่าขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l โดยเหตุผลที่ผู้วิจัยเลือกใช้ขั้นตอนวิธีดังกล่าวในการสร้างกฎ (แทนที่จะใช้วิธีอื่น ๆ เช่น ต้นไม้ตัดสินใจ, ข่ายงานประสาทเทียม หรือเทคนิคการเรียนรู้ของเครื่องจักรแบบอื่น ๆ) เนื่องจาก ผู้วิจัยต้องการสร้างกฎที่มนุษย์สามารถเข้าใจและอธิบายความหมายได้ แทนที่จะเป็นโมเดลที่เป็นค่าน้ำหนักในการตัดสินใจ และเหตุผลที่ไม่ใช้ต้นไม้ตัดสินใจเนื่องจากผู้วิจัยต้องการสร้างกฎที่ไม่ผูกติดอยู่กับค่าที่เป็นได้ของแอทริบิวต่าง ๆ แต่เป็นกฎที่เปรียบเทียบความสัมพันธ์ระหว่างแอทริบิวต่าง ๆ ที่นำมาพิจารณา

หลักการทำงานของขั้นตอนวิธี m+l คือ สร้างประชากรรุ่นแรก (ตัวอย่างคำตอบ) ขึ้นมา m รูปแบบ จากนั้นประเมินค่าความเหมาะสมของประชากรแต่ละตัว แล้วสุ่มเลือกประชากรที่มีค่าความเหมาะสมที่ดีมาเป็นต้นแบบในการสร้างประชากรรุ่นลูกขึ้นมาอีก l รูปแบบ แล้วจึงคัดเลือกประชากรที่มีค่าความเหมาะสมดีที่สุดโดยพิจารณาจากประชากรทั้งหมด m+l ตัว แล้วเลือก mu ตัวที่ดีที่สุดมาเป็นประชากรตั้งต้นในรุ่นถัดไป ขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l แสดงใน Figure 3 รายละเอียดเพิ่มเติมเกี่ยวกับขั้นตอนวิธีเชิงวิวัฒนาการสามารถอ่านได้จาก Mitchell[12], Goldberg[13] และการวิเคราะห์พฤติกรรมการทำงานของขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l สามารถศึกษาได้จาก Ter-Sarkisov และ Marsland[14]

วิธีการวิจัย

งานวิจัยนี้นำเสนอวิธีการในการระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน พร้อมทั้งสร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน คณะผู้วิจัยได้นำเสนอกฎที่ใช้ในการระบุต้นตอ ซึ่งกฎนี้ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l ซึ่งมีรายละเอียดดังนี้

1) การเข้ารหัสโครโมโซมและการสร้างประชากร คณะผู้วิจัยได้ออกแบบรูปแบบของกฎ แสดงดัง Figure 4 โดยแต่ละกฎประกอบด้วยพารามิเตอร์ที่จะถูกวิวัฒนาการ คือ ความยาวของกฎ (จำนวนเงื่อนไขที่ใช้เปรียบเทียบ), ค่าแอทริบิวที่นำพิจารณา, ตัวดำเนินการเปรียบเทียบ, ตัวดำเนินการทางตรรกะ

ประชากรแต่ละตัวจะถูกสุ่มสร้างขึ้นตามโครงสร้างโครโมโซมที่กล่าวไว้ข้างต้น โดยแต่ละ condition จะเป็นเงื่อนไขของกฎ ซึ่งประกอบด้วยแอทริบิวต่าง ๆ ที่ถูกสุ่มขึ้นมาเปรียบเทียบกันด้วยตัวดำเนินการเปรียบเทียบต่าง ๆ ได้แก่ >, <, =, >= และ <= โดยแอทริบิวต่าง ๆ ได้มาจากผลการเปรียบเทียบความคล้ายกันของซอร์สโค้ดจาก MOSS แอทริบิวที่ใช้ในงานวิจัยนี้ มีดังนี้

1. จำนวนบรรทัดของซอร์สโค้ดไฟล์ที่ 1
2. เปอร์เซ็นต์ความคล้ายที่ไฟล์ที่ 1 ไปเหมือนกับอีกไฟล์หนึ่ง
3. หมายเลขบรรทัดแรกที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

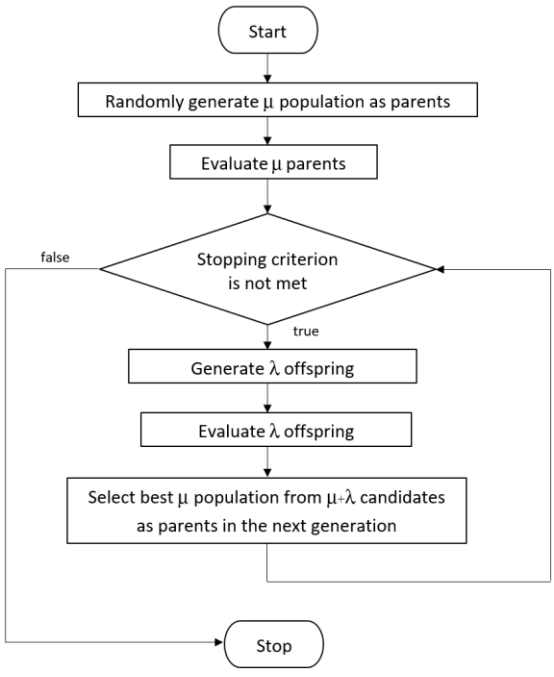

Figure 3 mu+lambda evolutionary algorithm

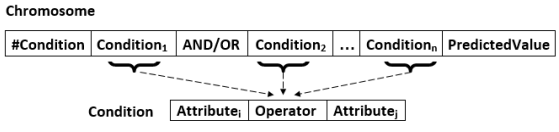

Figure 4 Chromosome encoding

4. หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

5. จำนวนบรรทัดของซอร์สโค้ดไฟล์ที่ 2

6. เปอร์เซ็นต์ความคล้ายที่ไฟล์ที่ 2 ไปเหมือนกับอีกไฟล์หนึ่ง

7. หมายเลขบรรทัดแรกที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

8. หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

9. จำนวนบรรทัดที่ทั้งสองไฟล์เหมือนกัน

10.ผลเฉลยที่ระบุว่าไฟล์ใดคัดลอกมาจากไฟล์ใด หรือ ไม่ได้มีการคัดลอก

2) การประเมินค่าความเหมาะสม

ประชากรหรือกฎในการระบุต้นตอที่ถูกสุ่มสร้างขึ้นแต่ละอัน จะถูกประเมินค่าความเหมาะสมเพื่อจะได้ทราบว่ากฎใดให้ค่าความถูกต้องในการระบุต้นตอของซอร์สโค้ดได้มากกว่ากัน ในงานวิจัยนี้ให้คะแนนค่าความเหมาะสมโดยการนำกฎที่สร้างขึ้นมาได้ไประบุต้นตอกับข้อมูลทดสอบ (Benchmark data) แล้วดูว่ามีกี่กรณีที่ระบุได้ถูกต้อง ในกรณีที่ระบุได้ถูกต้องจะได้ 1 คะแนนต่อหนึ่งกรณีทดสอบ ส่วนถ้ากฎนั้นระบุผลลัพธ์ผิดคะแนนก็จะถูก -1 ดังนั้นคะแนนรวมจากการทดสอบจะถูกใช้เป็นค่าความเหมาะสมในการคัดเลือกประชากรสำหรับวิวัฒนาการในรุ่นถัดไป

3) การสร้างประชากรรุ่นใหม่

ประชากรที่มีค่าความเหมาะสมที่ดีกว่าจะมีโอกาสมากกว่าที่จะถูกสุ่มหยิบมาเป็นต้นแบบในการสร้างประชากรรุ่นลูก ในการสร้างประชากรรุ่นใหม่ขึ้นมา จะนำต้นแบบจากประชากรเดิมมาสุ่มเปลี่ยนค่าต่าง ๆ ในโครโมโซม เช่น เพิ่ม/ลด ความยาวของกฎ, เปลี่ยนตัวดำเนินการทางตรรกะ, เปลี่ยนตัวดำเนินการเปรียบเทียบ, เปลี่ยนค่าของแอทริบิวที่จะทำการเปรียบเทียบ เมื่อสร้างประชากรรุ่นใหม่เสร็จแล้วก็จะนำกฎที่ปรับปรุงใหม่นี้มาประเมินค่าความเหมาะสมอีกครั้งหนึ่ง เพื่อเปรียบเทียบและคัดลอกประชากรที่ดีที่สุดให้อยู่รอดในรุ่นถัดไป

4) ข้อมูลและพารามิเตอร์ที่ใช้ในการทดลอง

งานวิจัยนี้ใช้ข้อมูลการลอกเลียนซอร์สโค้ดจากข้อมูลทดสอบ (Benchmark data) ที่เผยแพร่ในอินเทอร์เน็ต[15] ซึ่งชุดข้อมูลดังกล่าวประกอบด้วยไฟล์ต้นฉบับและซอร์สโค้ดที่มีการลอกเลียนมาจากต้นฉบับจำนวน 21 โปรแกรม โดยผู้สร้างชุดข้อมูลทดสอบนี้ได้แบ่งกลุ่มของชุดข้อมูลการลอกเลียนซอร์สโค้ดเป็น 4 กลุ่ม คือ

1. ชุดที่ 1 (T1) ชุดคำสั่งในโปรแกรมเหมือนกันทุกประการ มีการเปลี่ยนแปลงแค่การเพิ่มช่องว่าง การเว้นวรรค การขึ้นบรรทัดใหม่ หรือการเพิ่มคอมเมนต์เข้าไปในโปรแกรม

2. ชุดที่ 2 (T2) ชุดคำสั่งเดิม แต่อาจมีการเปลี่ยนชื่อตัวแปร สัญพจน์ (Literal) รวมถึงอาจมีการจัดรูปแบบโค้ดที่แตกต่างกัน

3. ชุดที่ 3 (T3) มีการเปลี่ยนแปลงซอร์สโค้ดเล็กน้อย เช่น มีการ เพิ่ม/ลบ บางคำสั่ง อาจมีการประกาศตัวแปรต่างชนิด ใช้ลูปคนละรูปแบบ รวมถึงอาจมีการจัดรูปแบบโค้ดที่แตกต่างกัน

4. ชุดที่ 4 (T4) ซอร์สโค้ดที่ทำงานเหมือนกันแต่ใช้คนละวิธี คนละเทคนิคในการเขียนโปรแกรม

ในส่วนของขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l คณะผู้วิจัยได้ใช้พารามิเตอร์ต่าง ๆ แสดงใน Table 1

Table 1 Parameter settings

Parameter	Value
The number of parents (mu)	10
The number of childs (lambda)	80
Tournament size	2
The number of generations	50

ผลการวิจัย

ผู้วิจัยได้ทำการทดลองสร้างกฎตามวิธีการที่ได้อธิบายในหัวข้อก่อนหน้า เพื่อระบุต้นตอของซอร์ส โค้ดที่มีความคล้ายกัน โดยกฎที่ได้แสดงใน Table 2

Table 2 Experimental result

Rule for identifying the original source code	% correct
IF endMatched_File1 < endMatched_File2 OR percentMatched_File1 > percentMatched_File2 **THEN** File1 is original **ELSE IF** endMatched_File1 > endMatched_File2 **THEN** File2 is original **ELSE IF** percentMatched_File1 = percentMatched_File2 **THEN** File1 is original **ELSE** not a copy	90.24%

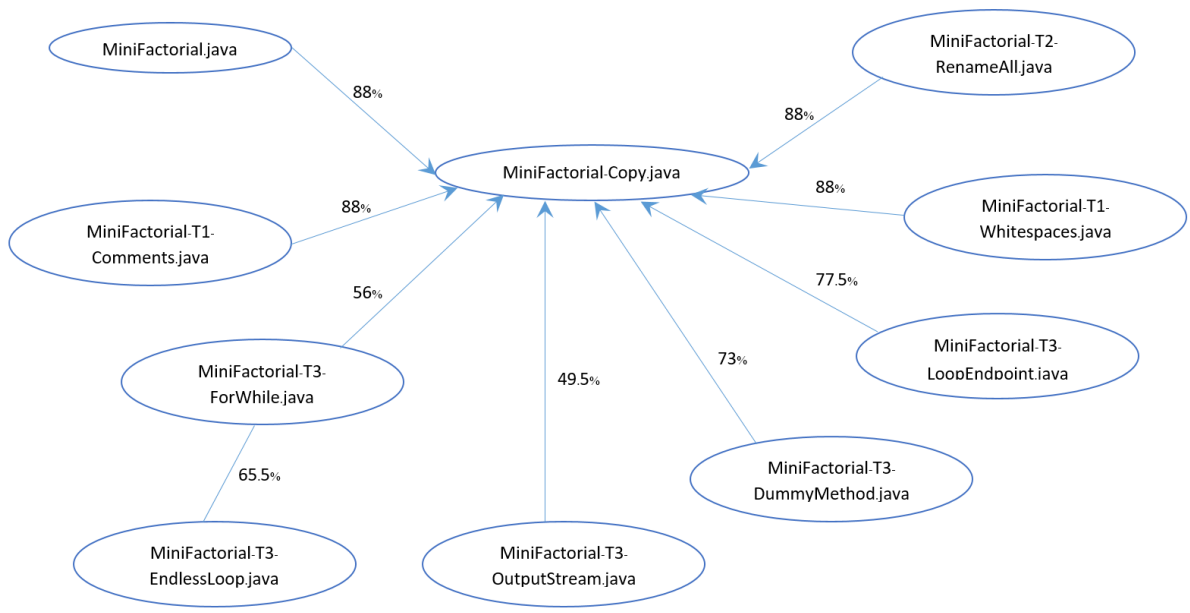

Figure 5 Phylogenetic tree of source code plagiarism

จากแอทริบิวทั้งหมดที่ได้มาจากผลลัพธ์การตรวจความคล้ายกันของซอร์สโค้ดโดยใช้ MOSS นั้น ขั้นตอนวิธีเชิงวิวัฒนาการได้คัดเลือกเหลือเพียงการใช้ หมายเลขบรรทัดสุดท้ายที่โค้ดเหมือนกัน และ เปอร์เซ็นต์ความคล้ายกันของซอร์สโค้ด เป็นเงื่อนไขในการระบุว่าไฟล์ใดเป็นต้นฉบับ โดยเริ่มต้นจะพิจารณาว่า ถ้าหมายเลขบรรทัดสุดท้ายที่ไฟล์ที่ 1เหมือนกับไฟล์ที่ 2 น้อยกว่า หมายเลขบรรทัดที่ไฟล์ที่ 2 เหมือนกับไฟล์ที่ 1 หรือ เปอร์เซ็นต์ความคล้ายของไฟล์ที่หนึ่งมากกว่า เปอร์เซ็นต์ความคล้ายของไฟล์ที่ 2 จะระบุว่า ไฟล์ที่ 1 เป็นต้นฉบับ แต่ถ้าไม่เข้าเงื่อนไขดังกล่าว จะพิจารณาต่อว่า ถ้า หมายเลขบรรทัดสุดท้ายที่ไฟล์ที่ 1 เหมือนกับไฟล์ที่ 2 มากกว่า หมายเลขบรรทัดที่ไฟล์ที่ 2 เหมือนกับไฟล์ที่ 1 จะระบุว่าไฟล์ที่ 2 เป็นต้นฉบับ แต่ถ้าไม่ใช่ก็จะเปรียบเทียบอีกว่า เปอร์เซ็นต์ความคล้ายของทั้ง 2 ไฟล์เท่ากันหรือไม่ ถ้าเท่ากัน จะระบุว่าไฟล์ที่ 1 เป็นต้นฉบับ แต่ถ้าไม่เข้าเงื่อนไขทั้งหมดที่กล่าวมาข้างต้น ก็จะระบุว่าทั้ง 2 ไฟล์ไม่ได้ลอกเลียนกัน

จากการใช้กฎดังกล่าวระบุไฟล์ที่มีการคัดลอก พบว่าสามารถระบุไฟล์ต้นฉบับและระบุไฟล์ที่ไม่ได้ลอกเลียนได้ถูก

ต้อง %90.24

เมื่อระบุไฟล์ต้นฉบับได้แล้ว ผู้วิจัยได้สร้างต้นไม้แสดงความสัมพันธ์ และระบุทิศทางของลูกศรเพื่อแสดงลำดับการสืบทอดของซอร์สโค้ดที่มีความคล้ายกัน ซึ่งต้นไม้นี้สร้างได้โดยการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุด แล้วระบุทิศทางโดยใช้กฎที่นำเสนอข้างต้น สำหรับชุดข้อมูลทดสอบนี้สามารถสร้างต้นไม้แสดงความสัมพันธ์ของการลอกเลียนซอร์สโค้ดได้ดัง Figure 5 ซึ่งจะเห็นได้ว่าไฟล์ต่าง ๆ ส่วนใหญ่แล้วคัดลอกมาจากต้นฉบับ คือ ไฟล์ชื่อ MiniFactorial-Copy.java อย่างไรก็ตาม ไฟล์ MiniFactorial-Copy.java กับไฟล์ชื่อ MiniFactorial.java ที่จริงแล้วคือไฟล์เดียวกัน ซึ่งไฟล์ที่เหมือนกันทุกประการนี้ กฎที่นำเสนอสามารถบอกได้ว่ามีการลอกกัน แต่ไม่สามารถระบุได้ชัดเจนว่าไฟล์ใดเป็นต้นฉบับ ซึ่งในที่นี้กฎที่ได้ระบุว่าไฟล์ MiniFactorial-Copy.java เป็นต้นฉบับ ทิศทางของลูกศรจึงเป็นดังภาพ นอกจากนี้ยังมีความสัมพันธ์ของอีก 1 คู่ไฟล์ คือไฟล์ชื่อ MiniFactorial-T-ForWhile.java กับไฟล์ชื่อ MiniFactorial-T-3EndlessLoop.java ซึ่งจากต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด ได้เลือกเส้นเชื่อมระหว่างไฟล์คู่นี้ไว้ ซึ่งทั้งสองไฟล์ก็มีความคล้ายกัน แต่ไม่ได้ลอกกันมาโดยตรง เส้นเชื่อมระหว่างไฟล์คู่นี้จึงไม่มีหัวลูกศรระบุต้นฉบับ

วิจารณ์และสรุปผล

จากผลการทดลองแสดงให้เห็นว่ากฎที่นำเสนอที่ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ m+l ให้ค่าความถูกต้องในการระบุการลอกเลียนซอร์สโค้ด 90.24% ซึ่งได้ทดสอบกับชุดข้อมูลที่มีทั้งการเปลี่ยนแปลงซอร์ส โค้ดแบบเล็กน้อย และซอร์สโค้ดที่มีการเปลี่ยนวิธีการในการเขียนโปรแกรม ต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดการลอกเลียนซอร์สโค้ดที่สร้างขึ้นนี้ จะเป็นเครื่องมือหนึ่งที่ช่วยให้อาจารย์เห็นภาพรวมของความคล้ายกันของงานเขียนโปรแกรมที่เป็นการบ้านที่นักเรียนส่งมา รวมทั้งได้ข้อสังเกตเกี่ยวกับว่าไฟล์ใดน่าจะเป็นต้นฉบับในการลอกเลียนซอร์สโค้ดมาส่ง เพื่อที่ผู้สอนจะได้ตักเตือนและให้คำแนะนำที่เหมาะสมกับนักเรียนต่อไป อย่างไรก็ตาม งานวิจัยนี้ได้นำเสนอผลการทดลองเบื้องต้น ซึ่งใช้ข้อมูลมาตรฐานที่เป็นข้อมูลทดสอบที่เผยแพร่ให้ใช้ในงานวิจัยทั่วไป (Benchmark) ในอนาคตผู้วิจัยวางแผนจะรวบรวมข้อมูลจริงและปรับปรุงขั้นตอนวิธีให้มีประสิทธิภาพมากขึ้น

กิตติกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนการวิจัยจากงบประมาณเงินรายได้ (เงินอุดหนุนจากรัฐบาล) ประจำปีงบประมาณ พ.ศ. 2561 มหาวิทยาลัยบูรพา ผ่านสำนักงานคณะกรรมการวิจัยแห่งชาติ เลขที่สัญญา 14/2561

เอกสารอ้างอิง

1. กิตติยา สุทธิประภา (2560), PLAGIARISM: ความสำคัญของการป้องกันการโจรกรรมทางวิชาการ, อินฟอร์เมชั่น; 24(1):90-97
2. Agrawal M, Sharma, DK. A state of art on source code plagiarism detection. Proceedings of 2nd International Conference on Next Generation Computing Technologies; 2016.
3. Castro Campos RA, Zaragoza Martinez FJ. Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem. Proceedings of 9th International Conference on Electrical Engineering, Computing Science and Automatic Control; 2012.
4. Son, JW, Noh, TG, Song HJ, Park, SB. An application for plagiarized source code detection based on a parse tree kernel, Engineering Applications of Artificial Intelligence 2013;26:1911-1918
5. Zhao, J, Xia K, Fu, Y, Cui B. An AST-based Code Plagiarism Detection Algorithm. Proceedings of 10th International Conference on Broadband and Wireless Computing, Communication and Applications; 2015.
6. Kamalim, O. Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. Proceedings of International Conference on Information & Communication Technology and Systems; 2016.
7. Qinqin, L, Chunhai, Z. Research on Algorithm of Program Code Similarity Detection. Proceedings of International Conference on Computer Systems, Electronics and Control; 2017.
8. Schneider, J, Bernstein, A, Brocke, JV, Damevski, K, Shepherd, DC. Detecting Plagiarism Based on the Creation Process, IEEE Transactions on Learning Technologies 2018;11(3):348361-
9. MOSS: A System for Detecting Software Similarity [online]. Available from: https://theory.stanford.edu/~aiken/moss/ Accessed Dec 2018.
10. Sedgewick, R. Algorithms. Massachusetts: Addison-

Wesley; 2011. P. 604636-

11. Pemmaraju, S, Skiena, S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. New York: Cambridge University Press, 2003
12. Mitchell, M. An Introduction to Genetic Algorithms. Massachusetts: MIT Press, 1996
13. Goldberg, DE. Genetic Algorithms in Search, Optimization and Machine Learning. Massachusetts: Addison-Wesley, 1989
14. Ter-Sarkisov, A, Marsland, S. Convergence Properties of ($\mu + \lambda$) Evolutionary Algorithms. Proceedings of the 25th AAAI Conference on Artificial Intelligence; 2011
15. Source Code Plagiarism Test Sets [online]. Available from: https://github.com/nordicway/SourceCode-Plagiarism-TestSets Dec 2018.