รายงานวิจัยฉบับสมบูรณ์

โครงการ การสร้างกราฟแสดงวิวัฒนาการของซอร์สโค้ด ในระบบตรวจจับการลอกเลียนซอร์สโค้ด

ณัฐนนท์ ลีลาตระกูล
ภารุจ รัตนวรพันธุ์
คนึงนิจ กุโบลา
สุนิสา ริมเจริญ
สุภาวดี ศรีคำดี

โครงการวิจัยประเภทงบประมาณเงินรายได้
(เงินอุดหนุนจากรัฐบาล) ประจำปีงบประมาณ พ.ศ. 2561
มหาวิทยาลัยบูรพา

รหัสโครงการ 2561A10802314
สัญญาเลขที่ 14/2561

รายงานวิจัยฉบับสมบูรณ์

โครงการ การสร้างกราฟแสดงวิวัฒนาการของซอร์สโค้ด ในระบบตรวจจับการลอกเลียนซอร์สโค้ด

ณัฐนนท์ ลีลาตระกูล
ภารุจ รัตนวรพันธุ์
คนึงนิจ กุโบลา
สุนิสา ริมเจริญ
สุภาวดี ศรีคำดี

คณะวิทยาการสารสนเทศ มหาวิทยาลัยบูรพา

กิตติกรรมประกาศ

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

Acknowledgment

This work was financially supported by the Research Grant of Burapha University through National Research Council of Thailand (Grant no. 14/2561).

บทคัดย่อ

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

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 nation 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 issue must be addressed to preclude this misbehavior before it becomes students' habit, and to promote positive attitude in our juveniles. In this paper, we, therefore, 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 mu+lambda 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%.

สารบัญ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัญหาการลอกเลียนวรรณกรรม (Plagiarism) เป็นปัญหาต่อการศึกษาและการวิจัยที่ฝังรากลึกเป็นระยะเวลายาวนาน ทำให้เป็นปัญหาสำคัญระดับชาติ ทรัพยากรมนุษย์จะไม่สามารถพัฒนาแบบก้าวไกลได้เลย หากไม่พยายามอย่างที่สุดที่จะคิดสร้างสรรค์นวัตกรรมของตนเอง โดยเยาวชนรุ่นใหม่มีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด ซึ่งหนึ่งในวิธีเหล่านั้นคือ การคัดลอกผลงานของคนอื่น และนำมาอ้างว่าเป็นของตน ดังนั้น การขโมยความคิดของผู้อื่นมาเป็นของตนเองเป็นปัญหาร้ายแรงที่ส่งผลต่อความเจริญก้าวหน้าของอุตสาหกรรมต่าง ๆ ที่ต้องใช้ซอฟต์แวร์และชื่อเสียงของประเทศในระยะยาว โดยเฉพาะอย่างยิ่งถ้าปัญหานี้เกิดขึ้นกับกำลังสำคัญของชาติในอนาคต นั่นคือ นักเรียนนักศึกษา

การลอกเลียนวรรณกรรมในกลุ่มนักเรียนนักศึกษาจึงเป็นเรื่องเร่งด่วนที่ต้องมีมาตรการมาแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสมซึ่งอาจก่อให้เกิดปัญหาใหญ่ที่ร้ายแรงตามมาได้ การแก้ปัญหาที่จุดเริ่มต้นในวัยเยาว์ จะเป็นการปลูกฝังทัศนคติที่ถูกต้องให้กับเยาวชนของชาติ ซึ่งจะเติบโตไปเป็นกำลังสำคัญของประเทศ และจะนำไปสู่การพัฒนาประเทศอย่างยั่งยืน แนวความคิดนี้สอดคล้องกับการพัฒนาและเสริมสร้างศักยภาพทุนมนุษย์ ตามยุทธศาสตร์ชาติในอีก 20 ปีข้างหน้า ที่ให้ความสำคัญกับการเสริมสร้างและพัฒนาศักยภาพทุนมนุษย์เป็นยุทธศาสตร์ที่ 1 กล่าวคือ “เพื่อให้คนไทยทุกกลุ่มวัยมีทักษะและความรู้ความสามารถที่จะเป็นฐานในการพัฒนาประเทศ มีการเรียนรู้อย่างต่อเนื่องตลอดชีวิตพร้อมรับบริบทการเปลี่ยนแปลงทั้งภายในและภายนอกประเทศ มีพฤติกรรมเสี่ยงทางสุขภาพที่ลดลงและมีคุณภาพชีวิตที่ดีขึ้น มีจิตสำนึกพลเมืองที่มีทัศนคติและพฤติกรรมที่เป็นประโยชน์ต่อส่วนรวม มีคุณธรรม จริยธรรม และ มีค่านิยมตามบรรทัดฐานที่ดีของสังคมไทย”

นอกเหนือจากยุทธศาสตร์ที่ 1 ที่เป็นเรื่องของการเสริมสร้างและพัฒนาศักยภาพทุนมนุษย์แล้ว ยุทธศาสตร์อื่น ๆ ก็มีความสำคัญไม่ยิ่งหย่อนไปกว่ากัน อาทิเช่น ยุทธศาสตร์ที่ 8 ที่เกี่ยวข้องกับการพัฒนาในด้านวิทยาศาสตร์ เทคโนโลยี วิจัย และนวัตกรรม ประเทศชาติจะต้องเร่งสร้างพื้นฐานความรู้ทางด้านวิทยาศาสตร์เทคโนโลยีให้กับเยาวชนของชาติเพื่อที่จะสร้างนวัตกรรมสำหรับขับเคลื่อนเศรษฐกิจและสังคมในอนาคต เทคโนโลยีสารสนเทศมีบทบาทสำคัญและเป็นสิ่งที่อยู่เบื้องหลังความสำเร็จของนวัตกรรมหลากหลายด้าน นอกจากนี้รัฐบาลยังมีนโยบายที่ต้องการผลักดันเศรษฐกิจของประเทศ โดย

วางรากฐานเศรษฐกิจไทยสู่ "ดิจิตอล อีโคโนมี" ยุทธศาสตร์และนโยบายเหล่านี้จะเป็นจริงขึ้นมาได้ต้องอาศัยความรู้ทางเทคโนโลยีสารสนเทศและการผลิตซอฟต์แวร์

การผลิตบัณฑิตทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์เพื่อตอบสนองอุตสาหกรรมการผลิตซอฟต์แวร์ จึงเป็นกลไกสำคัญในการขับเคลื่อนยุทธศาสตร์และนโยบายของชาติในการสร้างนวัตกรรม การจัดการเรียนการสอนวิชาการเขียนโปรแกรมจึงเป็นหัวใจสำคัญของการสร้างบัณฑิตเพื่อให้จบออกไปประกอบอาชีพทางการผลิตซอฟต์แวร์ แต่ดังที่ได้กล่าวไปก่อนหน้า ปัญหาของเยาวชนรุ่นใหม่คือ นักเรียนนักศึกษามีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด ซึ่งหนึ่งในวิธีเหล่านั้นคือการคัดลอกโปรแกรมของคนอื่นมาส่งอาจารย์ ปัญหานี้จะต้องรีบถูกแก้ไข ดังนั้น งานวิจัยนี้จึงนำเสนอวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ด (Source Code) เพื่อให้ทราบถึงแหล่งที่มาของการลอกโปรแกรมในกลุ่มนักเรียนนักศึกษา และเมื่ออาจารย์ผู้สอนทราบว่านักเรียนคนใดเป็นต้นฉบับ คนใดที่เป็นคนลอกเพื่อนมาส่ง จะได้วางแนวทางและมาตรการแก้ไข ตักเตือน และแก้ปัญหาได้ตั้งแต่ต้น

1.2 วัตถุประสงค์ของโครงการวิจัย

1. เพื่อศึกษาและสร้างองค์ความรู้ที่เป็นพื้นฐานเกี่ยวกับพฤติกรรมการส่งงานของนิสิต
2. เพื่อศึกษาและสร้างองค์ความรู้ที่เป็นพื้นฐานสำหรับการหาค่าความคล้ายของโปรแกรม (Source Code Similarity)
3. เพื่อปรับปรุงประสิทธิภาพขั้นตอนวิธีสำหรับการหาค่าความคล้ายของโปรแกรม
4. เพื่อให้การตรวจจับซอร์สโค้ดที่คล้ายกันได้ เพื่อเป็นแนวทางควบคุมและลดการคัดลอกโปรแกรมของนิสิต
5. เพื่อหาวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ด เพื่อเป็นแนวทางสำหรับตักเตือน ป้องกันทั้งผู้ให้คัดลอก และ ผู้ขอคัดลอก
6. เพื่อสร้างระบบสำหรับตรวจความคล้ายคลึงกันของซอร์สโค้ดของนิสิตจำนวนมาก ที่สามารถแสดงผลลัพธ์เป็นกราฟความสัมพันธ์ที่เข้าใจได้ง่าย วิเคราะห์ข้อมูลได้ง่าย
7. เพื่อเป็นการปลูกฝังทัศนคติที่ถูกต้องในเรื่องการสร้างสรรค์ผลงานด้วยตนเอง ไม่ไปลอกเลียนแบบความคิดของคนอื่น ให้กับเยาวชนของชาติ
8. เป็นแนวทางเพื่อลดการคัดลอกซอร์สโค้ดในหมู่นิสิต ส่งเสริมให้นิสิตรู้จักอดทนที่จะสร้างสรรค์งานของตนเอง และ เตรียมตัวเองให้พร้อมสำหรับการทำงานพัฒนาซอฟต์แวร์ในการทำงานจริง
9. ให้ผู้ที่สนใจสามารถนำแนวคิดที่ได้นำเสนอไปประยุกต์ใช้หรือต่อยอดในงานวิจัยของตนเองต่อไป

1.3 ขอบเขตของโครงการวิจัย

1. ศึกษาวิจัยพฤติกรรมการลอกเลียนซอร์สโค้ดของนิสิตนักศึกษาในวิชาการเขียนโปรแกรม
2. ศึกษาวิจัยขั้นตอนวิธีตรวจจับความคล้ายคลึงกันของซอร์สโค้ด
3. ศึกษาวิจัยขั้นตอนวิธีสำหรับหาต้นตอของซอร์สโค้ด ที่นิสิตใช้คัดลอก และ หากลุ่มของนิสิตที่มีซอร์สโค้ดเหมือนกัน
4. ซอร์สโค้ดที่นำมาศึกษาเป็นซอร์สโค้ดที่ถูกเขียนขึ้นด้วยภาษา C++ และ JAVA
5. ระบบที่พัฒนาขึ้นสามารถรองรับผู้ใช้จำนวนมาก และ เป็นโปรแกรมประยุกต์บนเว็บสำหรับการตรวจสอบซอร์สโค้ด (Source code) ที่คล้ายกัน สามารถจับกลุ่มโปรแกรมที่เกิดจากการคัดลอกกัน และ หาต้นตอของการคัดลอก

1.4 ประโยชน์ที่ได้รับ

1. เข้าใจและสามารถสร้างองค์ความรู้ที่เป็นพื้นฐานเกี่ยวกับพฤติกรรมการส่งงานของนิสิต
2. เข้าใจและสามารถสร้างองค์ความรู้ที่เป็นพื้นฐานสำหรับการหาค่าความคล้ายของโปรแกรม
3. ได้ขั้นตอนวิธีสำหรับการหาค่าความคล้ายของโปรแกรมที่มีประสิทธิภาพมากขึ้น
4. ได้ระบบที่สามารถตรวจจับซอร์สโค้ดที่คล้ายกันได้ เพื่อเป็นแนวทางควบคุมและลดการคัดลอกโปรแกรมของนิสิต
5. ได้ระบบที่สามารถระบุต้นตอของการลอกเลียนซอร์สโค้ด เพื่อเป็นแนวทางสำหรับตักเตือนป้องกัน ทั้งผู้ให้คัดลอก และ ผู้ขอคัดลอก
6. ได้ระบบสำหรับตรวจความคล้ายคลึงกันของซอร์สโค้ดที่รองรับซอร์สโค้ดของนิสิตจำนวนมาก และสามารถแสดงผลลัพธ์เป็นกราฟความสัมพันธ์ ทำให้เข้าใจและวิเคราะห์ข้อมูลได้ง่าย
7. ได้ระบบที่เป็นเครื่องมือช่วยปลูกฝังทัศนคติที่ถูกต้องในเรื่องการสร้างสรรค์ผลงานด้วยตนเอง ไม่ไปลอกเลียนแบบความคิดของคนอื่น ให้กับเยาวชนของชาติ
8. เป็นแนวทางต้นแบบสำหรับลดการคัดลอกซอร์สโค้ดในหมู่นิสิต ส่งเสริมให้นิสิตรู้จักอดทนที่จะสร้างสรรค์งานของตนเอง และ เตรียมตัวเองให้พร้อมสำหรับการทำงานพัฒนาซอฟต์แวร์ในการทำงานจริง
9. ให้ผู้ที่สนใจสามารถนำแนวคิดที่ได้นำเสนอไปประยุกต์ใช้หรือต่อยอดในงานวิจัยของตนเองต่อไป

บทที่ 2
เนื้อเรื่อง

2.1 ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องแบ่งเป็น 4 หัวข้อ คือ การลอกเลียนวรรณกรรม เครื่องมือที่ใช้ในการตรวจการลอกเลียนซอร์สโค้ด ตัวชี้วัดความคล้าย และ กราฟ

2.1.1 การลอกเลียนวรรณกรรม

การลอกเลียนวรรณกรรม (Plagiarism) หมายถึง การแอบอ้างงานเขียน หรืองานสร้างสรรค์ของผู้อื่น ไม่ว่าจะเป็นการคัดลอกมาทั้งหมด หรือนำแค่เพียงบางส่วนมาใช้ในงานของตนเอง ทั้งนี้โดยมีเจตนาที่จะปิดบังแหล่งที่มา ทำให้ผู้อื่นเข้าใจว่าผลงานนี้เป็นของตนเอง ไม่ได้มีการอ้างอิงตามหลักวิชาการที่ถูกต้อง

ปัญหาการลอกเลียนวรรณกรรม เป็นปัญหาร้ายแรงในแวดวงวิชาการ ดังจะพบเห็นได้ว่ามีข่าวเป็นระยะ ๆ ที่พบว่ามีการลอกผลงานวิจัย วิทยานิพนธ์ของคนอื่นมาเป็นผลงานของตนเอง รวมไปถึงการลอกเลียนความคิดในการสร้างผลิตภัณฑ์ นวัตกรรมต่าง ๆ ดังนั้น จึงมีการพัฒนาเครื่องมือที่ใช้ตรวจสอบการลอกเลียนวรรณกรรมขึ้นมา เพื่อช่วยตรวจสอบผลงานว่ามีการคัดลอกส่วนหนึ่งส่วนใดมาจากผู้อื่นหรือไม่ เพื่อลดปัญหาการละเมิดทรัพย์สินทางปัญญาของผู้อื่น และสร้างความชอบธรรมในสังคม

วิธีการตรวจสอบการลอกเลียนวรรณกรรมแบ่งได้เป็น 2 ประเภทใหญ่ ๆ คือ

1. External Plagiarism Detection Method

   วิธีการนี้เป็นการตรวจสอบการลอกเลียนเอกสารโดยอาศัยคลังข้อมูล (Corpus) การเปรียบเทียบเพื่อที่จะระบุได้ว่าเอกสารนั้น ๆ ได้มีการลอกเลียนมาจากแหล่งใด อาจจะตรวจสอบจากการเปรียบเทียบสายอักขระ (String Matching), การเปรียบเทียบโดยการดูบริบทและความหมาย (Semantics), การจัดกลุ่มตามความคล้ายคลึง (Clustering), การเปรียบเทียบการลอกเลียนมาจากเอกสารภาษาอื่น (Cross Lingual), การ

เปรียบเทียบเอกสารอ้างอิง (Citation), การตรวจหาเอกลักษณ์ของเอกสาร (Fingerprint) เป็นต้น

2. Intrinsic Plagiarism Detection Method

วิธีการนี้เป็นการตรวจสอบการลอกเลียนวรรณกรรมโดยที่ไม่ต้องอาศัยคลังข้อมูล เช่น การใช้การประมวลผลภาษาธรรมชาติ (Natural Language Processing) ถอดความ (Paraphrasing) และหาความคล้ายคลึง (Similarity) ของคำที่ปรากฏ, การตรวจสอบโครงสร้างของเอกสารเช่น การแบ่งองค์ประกอบของเนื้อหา การแบ่งย่อหน้า การอ้างอิง ฯลฯ, การตรวจสอบชนิดของคำที่ใช้ในแต่ละประโยค (Part of Speech) เป็นต้น

2.1.2 เครื่องมือที่ใช้ในการตรวจการลอกเลียนซอร์สโค้ด

ซอร์สโค้ด (Source Code) คือ ชุดคำสั่งทางซอฟต์แวร์ที่ถูกเขียนขึ้นเพื่อเรียบเรียงขั้นตอนการแก้ปัญหาให้คอมพิวเตอร์ประมวลผลตามคำสั่งที่ระบุ ซอร์สโค้ดที่ถูกเขียนขึ้นมาเป็นลิขสิทธิ์ของโปรแกรมเมอร์หรือผู้ว่าจ้าง การที่มีผู้หนึ่งผู้ใดนำชุดคำสั่งของบุคคลอื่นมาเป็นของตนจึงเป็นเรื่องที่ไม่ถูกต้อง

ในทางการศึกษา วิชาการเขียนโปรแกรมถือเป็นวิชาพื้นฐานที่สอนให้ผู้เรียนเข้าใจวิธีการเขียนชุดคำสั่งเหล่านี้ นิสิตนักศึกษาที่เรียนทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์ จะต้องเรียนวิชาการเขียนโปรแกรมเพื่อจบออกไปประกอบอาชีพในอนาคต ในการเรียนการสอนวิชานี้ ผู้เรียนจำเป็นจะต้องฝึกหัดเขียนโปรแกรมเพื่อแก้ปัญหาที่หลายหลาย เป็นการฝึกทักษะกระบวนการคิดแก้ปัญหา การเรียบเรียงคำสั่งสำหรับเขียนโปรแกรมออกมาเป็นลำดับขั้นตอน ซึ่งกระบวนการฝึกฝนนี้เป็นหัวใจสำคัญที่จะทำให้ผู้เรียนมีความรู้ความสามารถในการประกอบอาชีพโปรแกรมเมอร์เมื่อจบการศึกษา

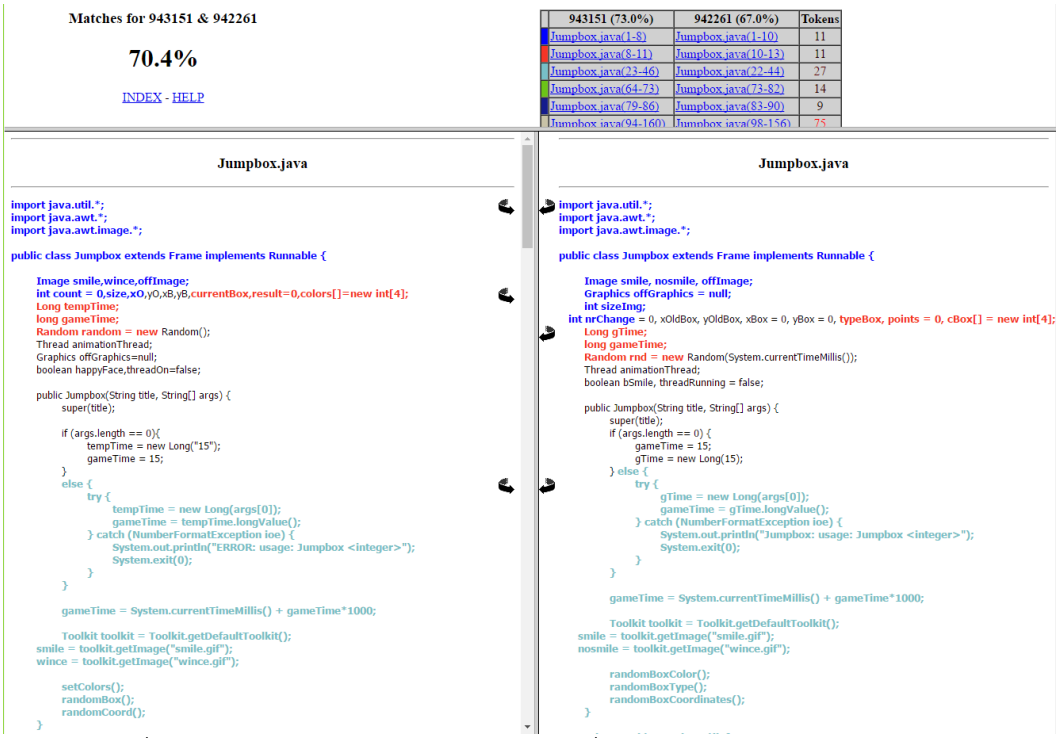
แต่ในปัจจุบัน มีผู้เรียนจำนวนหนึ่งที่ไม่พยายามใช้ความสามารถของตัวเอง แต่จะพยายามหาวิธีที่ง่าย ๆ ที่ไม่ถูกต้องเพื่อที่จะทำให้ตนเองมีงานส่งอาจารย์ นั่นคือการลอกเลียนโปรแกรมของคนอื่นมาส่ง การทำเช่นนี้อาจทำให้ติดเป็นนิสัยและอาจเกิดผลเสียหายร้ายแรงได้ในระยาว ดังนั้น จึงต้องมีมาตรการในการแก้ไขตั้งแต่ในตอนเริ่มแรก เพื่อเป็นการปลูกฝังจิตสำนึกให้กับเยาวชน และเป็นการเสริมสร้างแนวความคิดในการสร้างสรรค์ผลงานด้วยตนเอง

เครื่องมือที่จะช่วยตรวจสอบการลอกเลียนโปรแกรมจึงเป็นเครื่องมือที่จะเข้ามาช่วยระบุว่าซอร์สโค้ดของใครบ้างที่มีความใกล้เคียงกัน ตัวอย่างของเครื่องมือที่เป็นที่นิยม เช่น

- MOSS (Measure Of Software Similarity) เป็นเครื่องตรวจหาความคล้ายกันของซอร์สโค้ด ซึ่งพัฒนาขึ้นที่มหาวิทยาลัย Stanford 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 รูปแบบผลลัพธ์การเปรียบเทียบความคล้ายกันของซอร์สโค้ดจาก MOSS จะแสดงในรูปแบบไฟล์ HTML โดยมีการแสดงคู่ของโปรแกรมที่คล้ายกัน และ highlight โค้ดในส่วนที่มีความคล้ายกัน ดังตัวอย่างในรูปที่ 2-1

รูปที่ 2-1 ผลลัพธ์การระบุส่วนของโปรแกรมที่มีความคล้ายกันจากระบบ MOSS
(รูปภาพจาก: http://lightonphiri.org/blog/student-programming-plagiarism-dectection-using-moss)

- JPlag ถูกพัฒนาขึ้นที่มหาวิทยาลัย Karlsruhe โดย Guido Malpohl ตั้งแต่ปี ค.ศ. 1996 และได้ถูกปรับปรุง พัฒนาต่อมาเรื่อย ๆ จนกระทั่งในปี ค.ศ. 2005 Jplag ได้ให้บริการผ่านเว็บ เครื่องมือนี้รองรับภาษาในการเขียนโปรแกรมได้แก่ Java, C#, C, C++, Scheme และภาษาธรรมชาติที่เป็นข้อความ ลักษณะการแสดงผลลัพธ์ก็จะมีการเปรียบเทียบและระบุสีเพื่อให้เห็นโค้ดในส่วนที่ตรวจพบว่ามีความคล้ายกัน ตัวอย่างผลลัพธ์จาก Jplag แสดงดังรูปที่ 2-2



รูปที่ 2-2 ผลลัพธ์การระบุส่วนของโปรแกรมที่มีความคล้ายกันจากระบบ Jplag
(รูปภาพจาก: https://jplag.ipd.kit.edu/example/match1.html)

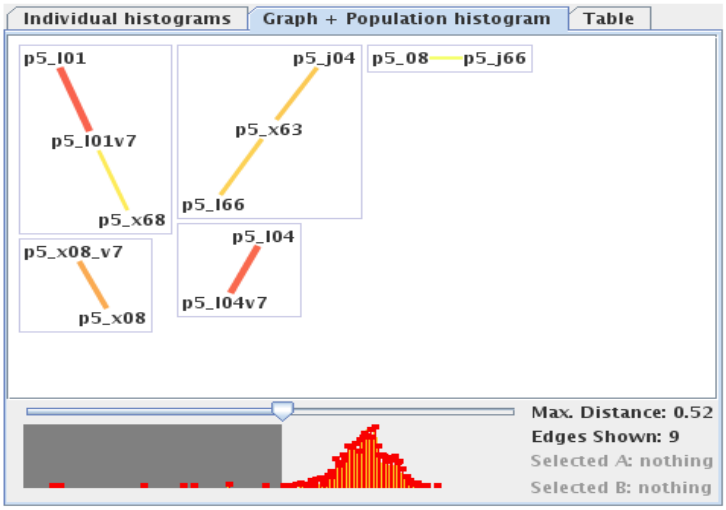
- SIM ถูกพัฒนาขึ้นที่มหาวิทยาลัย Amsterdam ในปี ค.ศ. 1989 โดย Dick Grune โปรแกรมนี้แตกต่างจาก MOSS และ Jplag ตรงที่เป็นโปรแกรมที่รันภายในเครื่อง (Run locally) ไม่ได้เป็นบริการผ่านเว็บ และได้มีการแจกจ่ายต้นฉบับ (Open Source) ให้ผู้ที่สนใจนำไปพัฒนาต่อได้ ภาษาโปรแกรมที่รองรับการตรวจหาความคล้ายกันได้แก่ C, Java, Pascal, Modula-2, Lisp, Miranda และ ภาษาธรรมชาติที่เป็นข้อความ

- Plaggie ถูกพัฒนาขึ้นที่มหาวิทยาลัย Helsinki ในปี ค.ศ. 2002 โดย Ahtiainen และคณะ โปรแกรมนี้รองรับการตรวจหาความคล้ายกันของซอร์สโค้ดภาษาจาวา โดยผู้ใช้สามารถติดตั้ง

โปรแกรมนี้ที่เครื่องของตนเองรวมทั้งผู้พัฒนาได้แจกจ่ายโปรแกรมให้นำไปพัฒนาเองต่อได้ (Open Source) ผลลัพธ์ที่ Plaggie เป็นทั้งแบบโหมดตัวอักษร (Text) และเป็น HTML

- Marble ถูกพัฒนาขึ้นที่มหาวิทยาลัย Utrecht ในปี ค.ศ. 2002 รองรับการตรวจความคล้ายกันของภาษา Java, C#, Perl, PHP, XSLT โดยผู้ใช้จะต้องติดตั้งโปรแกรม Marble ที่เครื่องของตนเอง (Run locally) ผลลัพธ์ของการเปรียบเทียบซอร์สโค้ดเพื่อหาความคล้ายคลึงกันจะแสดงผลอยู่ในรูปไฟล์ข้อความ (Text file)

- AC ถูกพัฒนาขึ้นที่มหาวิทยาลัย Autónoma de Madrid รองรับการตรวจความคล้ายกันของภาษา C, C++, Java เป็นโปรแกรมที่เปิดเผยโค้ดให้นักพัฒนาสามารถดาวน์โหลดไปพัฒนาต่อได้ (Open Scource) ผลลัพธ์การเปรียบเทียบความคล้ายกันของซอร์สโค้ดที่นำมาตรวจสอบ มีการแสดงเป็นกราฟิก ดังตัวอย่างในรูปที่ 2-3



รูปที่ 2-3 การแสดงผลลัพธ์แบบกราฟิกจากระบบ AC
(รูปภาพจาก: http://tangow.ii.uam.es/ac/?q=home)

เครื่องมือดังกล่าวข้างต้น เป็นตัวอย่างของความพยามที่จะสร้างโปรแกรมอัตโนมัติมาตรวจหาความคล้ายกันของซอร์สโค้ดที่ถูกเขียนขึ้นโดยภาษาโปรแกรมต่าง ๆ จะเห็นได้ว่าเครื่องมือแต่ละตัวก็มีจุดเด่นจุดด้อยแตกต่างกันไป ตารางที่ 2-1 ด้านล่าง แสดงการเปรียบเทียบคุณสมบัติของโปรแกรมตรวจหาความคล้ายกันของซอร์สโค้ด

ตารางที่ 2-1 เปรียบเทียบคุณสมบัติของโปรแกรมตรวจหาความคล้ายกันของโค้ด

คุณสมบัติ	MOSS	Jplag	SIM	Plaggie	Marble	AC
1. รองรับภาษา	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	Java, C#, C, C++, Scheme, ภาษาธรรมชาติ	C, Java, Pascal, Modula-2, Lisp, Miranda, ภาษาธรรมชาติ	Java	Java, C#, Perl, PHP, XSLT	C, C++, Java
2. Local หรือ Web-based	web	web	local	local	local	local
3. Open Source	no	no	yes	yes	no	yes
4. url	http://theory.stanford.edu/~aiken/moss/	https://jplag.ipd.kit.edu/	http://dickgrune.com/Programs/similarity_tester/	https://www.cs.hut.fi/Software/Plaggie/	http://foswiki.cs.uu.nl/	http://tangow.ii.uam.es/ac/?q=home

2.1.3 ตัวชี้วัดความคล้าย (Similarity Measure)

หัวใจสำคัญของการระบุความใกล้เคียงกันของซอร์สโค้ดก็คือ การวัดค่าความคล้าย (Similarility) ซึ่งตัววัดความคล้ายที่เป็นที่นิยมใช้ก็มีอยู่หลายตัวชี้วัด เช่น

Jaccard Index

Jaccard Index หรือ Jaccard coefficient เป็นตัวชี้วัดความคล้ายที่ใช้วัดความเหมือนหรือความหลากหลายของเซต โดยคำนวณได้จาก อินเตอร์เซกชันของ เซตA กับ เซตB หารด้วยยูเนียนของ เซต A กับ เซต B โดยค่าผลลัพธ์ที่ได้จะอยู่ระหว่าง 0 ถึง 1 ซึ่งค่าศูนย์หมายถึงไม่มีความคล้ายกันเลย และ หนึ่งคือทั้งสองเซตเหมือนกัน

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Sorensen index

Sorensen index หรือ Sorensen coefficient เป็นตัวชี้วัดความคล้าย ที่ใช้วัดความเหมือนหรือต่างกันของตัวอย่างสายพันธุ์ของสิ่งมีชีวิต โดยคำนวณได้จากสมการ

$$QS = \frac{2C}{A+B}$$

โดยที่ A และ B คือ จำนวนสมาชิกของสายพันธุ์ และ C คือจำนวนที่เหมือนกันของสมาชิกในสายพันธุ์ A และ B โดยค่าผลลัพธ์ที่ได้จะอยู่ระหว่าง 0 ถึง 1 ซึ่งค่าศูนย์หมายถึง ไม่มีความคล้ายกันเลย และ หนึ่งคือทั้งสองชนิดเหมือนกัน

Tanimoto coefficient

Tanimoto coefficient เป็นตัวชี้วัดความคล้ายที่ประยุกต์มาจาก Cosine Similarity และ Jaccard Index โดย Cosine Similarity เป็นการวัดความคล้ายกันเชิงมุมของเวกเตอร์โดย Tanimoto coefficient จะคำนวณจาก สมการ

$$T(A,B) = \frac{A \cdot B}{||A||^2 + ||B||^2 - A \cdot B}$$

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$

$$\|A\| = \sqrt{A_1^2} + \cdots + A_n^2$$

โดย Dot Product ของเวกเตอร์ A และ B หารด้วย Magnitude ของเวกเตอร์ A บวกกับ Magnitude ของเวกเตอร์ B ลบด้วย A • B โดยค่าที่คำนวณได้จากสมการจะมีค่าผลลัพธ์อยู่ระหว่าง 0 ถึง 1 คือ ถ้าค่าที่คำนวณได้มีค่า 0 คือ เวกเตอร์ทั้งสอง ต่างกันโดยสิ้นเชิง หรือถ้าค่าที่คำนวณได้เป็น 1 คือ เวกเตอร์ทั้งสอง เหมือนกัน

TF-IDF Weight

TF-IDF ย่อมาจาก Term Frequency-Inverse Document Frequency เป็นการวัดค่าน้ำหนักที่เอาไว้ประเมินค่าความสำคัญของคำนั้นๆ ในกลุ่มของเอกสารซึ่ง TF-IDF มักจะใช้ในการสืบค้นข้อมูล โดย TF คือ จำนวนของการปรากฏของคำในเอกสารที่สนใจ ส่วน IDF คือ ค่าที่ได้มาจากการคำนวณดังสมการ

$$\mathrm{idf} = \log \frac{\mathrm{N}}{\mathrm{df}}$$

$$TF - IDF = \mathrm{tf} \times \mathrm{idf}$$

โดย N แทนจำนวนเอกสารทั้งหมด และ df คือจำนวนเอกสารที่พบคำที่ปรากฏทั้งหมด การคำนวณ น้ำหนัก ของ TF-IDF จะคำนวณจาก tf คูณด้วย idf (tf คือจำนวนความถี่ของคำที่ปรากฏในเอกสารนั้น ๆ) หลังจากที่คำนวณ น้ำหนักของคำได้แล้ว จึงนำค่าน้ำหนักของคำที่ได้ของแต่ละคำไปหาความคล้ายโดยใช้ตัวชี้วัดความคล้าย Tanimoto coefficient

Jaro-Winkler distance

Jaro-Winkler distance เป็นการวัดความคล้ายคลึงกันของสายอักขระ (String) โดยเป็นการพัฒนามาจาก Jaro distance โดยค่าของความคล้ายที่คำนวณได้ จะมีค่าระหว่าง 0 ถึง 1 โดย 0 คือ

สายอักขระทั้งสองไม่เหมือนกัน หรือเป็น 1 ถ้าสายอักขระทั้งสองเหมือนกัน วิธีคำนวณโดยเริ่มจากคำนวณ Jaro distance ดังสมการ

$$jaro\ distance = \ d_j = \ \frac{1}{3}\left(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{m}\right)$$

$$\text{matching} \ = \ \left[\frac{max(|s_1|, |s_2|)}{2}\right] - \ 1$$

$$jaro - winkler\ distance = \ d_w = \ d_j + \left(\ell p\left(1 - d_j\right)\right)$$

โดยที่

m คือ จำนวนตัวอักษรที่เหมือนกัน

|s1| คือความยาวของสายอักขระที่ 1

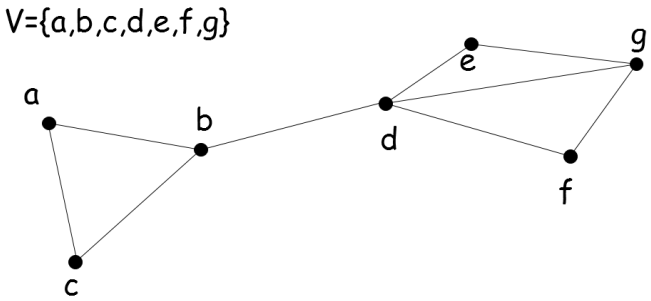
|s2| คือความยาวของสายอักขระที่ 2

t คือ ครึ่งหนึ่งของจำนวนทรานโพสิชั่น

ทรานโพสิชั่น คือ ตัวอักขระที่เหมือนกันแต่ไม่ได้อยู่ในตำแหน่งเดียวกัน โดยระยะห่างของอักขระที่ เหมือนกันแต่ไม่ได้อยู่ในตำแหน่งเดียวกันนั้น จะต้องอยู่ห่างกันไม่เกินกว่าค่าที่คำนวณได้จากสมการ matching หลังจากคำนวณค่าของ Jaro distance ได้แล้ว จึงนำมาคำนวณหาผลลัพธ์ของ Jaro-Winkler distance เพื่อหาค่าผลลัพธ์ โดยผลลัพธ์ที่ได้จะมีค่าระหว่าง 0 ถึง 1 โดย 0 คือสายอักขระทั้งสองไม่เหมือนกัน และ 1 คืออักขระทั้งสองเหมือนกัน

2.1.4 กราฟ (Graph)

กราฟเป็นโครงสร้างข้อมูลชนิดหนึ่ง ประกอบด้วยสองส่วนหลัก คือ โหนด (Node/Vertex) และ เส้นเชื่อม (Edge) โดยเส้นเชื่อมนี้อาจเป็นเส้นเชื่อมที่มีทิศทาง (Directional edge) หรือ มีน้ำหนัก (Weight) ด้วยก็ได้ ตัวอย่างของกราฟแสดงดังรูปที่ 2-4



รูปที่ 2-4 โครงสร้างกราฟที่ประกอบด้วยเซตของโหนด (V) และเซตของเส้นเชื่อม (E)

ต้นไม้ทอดข้ามที่เล็กที่สุด (Minimal spanning tree)

ต้นไม้ (Tree) เป็นโครงสร้างข้อมูลชนิดหนึ่งที่ประกอบไปด้วยโหนด และความสัมพันธ์ระหว่างโหนดเป็นลำดับชั้น การออกแบบเครือข่ายโดยมีความยาวน้อยที่สุด เรียกว่าการแก้ปัญหา “ต้นไม้ทอดข้ามที่เล็กที่สุด” หรือ Minimal spanning tree ตัวอย่างเช่น ในการออกแบบการเดินสายไฟ, ท่อแก๊ส, หรือท่อประปาในหมู่บ้านสร้างใหม่ สายไฟ หรือ ท่อจะต้องเชื่อมบ้านทุกหลังเข้ากับผู้ให้บริการ บ้านแต่ละหลังมีจุดเชื่อมต่อในตำแหน่งเดียวกัน แต่เส้นทางที่เดินเชื่อมต่อระหว่างบ้านจะทำอย่างไรก็ได้ ขอเพียงให้มีการเชื่อมต่อเท่านั้น

Minimal spanning tree ไม่เพียงมีประโยชน์ในการออกแบบการเดินท่อประปาและการเดินสายไฟ แต่ยังมีประโยชน์ในการออกแบบและแก้ปัญหาเครือข่ายคอมพิวเตอร์, เครือข่ายโทรศัพท์, การเดินท่อส่งน้ำมัน, และการกำหนดเส้นทางการบิน เป็นต้น อย่างไรก็ดีการเลือกเส้นทางสำหรับการเดินทางท่องเที่ยวนอกจากระยะทางแล้ว เรายังต้องคำนึงถึงความสะดวกสบาย และค่าใช้จ่ายด้วย ไม่มีใครอยากจะใช้เวลาหลายชั่วโมงที่บินอ้อมผ่านจุดที่ให้บริการใหม่เพียงเพราะตั๋วเครื่องบินราคาถูก ขั้นตอนวิธีที่ใช้กับเมืองในหล่มโคลนใช้กับการเดินทางแบบนี้ไม่ได้ เพราะให้ความสนใจเฉพาะระยะทางเพียงอย่างเดียว

Minimal spanning tree ยังมีประโยชน์ในฐานะเป็นขั้นตอนหนึ่งของขั้นตอนวิธีในการแก้ปัญหาแบบอื่นของกราฟ เช่น ปัญหาการเดินทางของพนักงานขาย (Traveling salesperson problem) ซึ่งต้องการหาเส้นทางที่สั้นที่สุดที่จะสามารถเดินทางไปถึงจุดทุกจุดในเครือข่ายได้

ขั้นตอนวิธีในการหา Minimal spanning tree ที่มีประสิทธิภาพสูงมีหลายแบบ ขั้นตอนวิธีที่ง่ายที่สุดที่ให้คำตอบที่ดีที่สุด เริ่มต้นด้วยการไม่กำหนดเส้นเชื่อมเลย จากนั้นจึงเพิ่มเส้นเชื่อมทีละเส้นตามขนาด โดยเพิ่มให้จากจุดที่เชื่อมต่อกับเครือข่ายอยู่แล้วไปยังจุดใหม่ที่ยังไม่มีการเชื่อมต่อ วิธีการนี้

เรียกว่าขั้นตอนวิธีของครูสกาล (Kruskal's algorithm) เพื่อเป็นเกียรติแก่ Joseph B. Kruskal (1928-2010) ผู้คิดวิธีการนี้ในปี ค.ศ. 1956 ขั้นตอนวิธีของ Kruskal แสดงดังรูปที่ 2-5

```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3   MAKE-SET(v)
4 foreach (u, v) in G.E ordered by weight(u, v), increasing:
5   if FIND-SET(u) ≠ FIND-SET(v):
6     A = A ∪ {(u, v)}
7     UNION(u, v)
8 return A
```

รูปที่ 2-5 ขั้นตอนวิธีของ Kruskal

ต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด (Maximum spanning tree)

ต้นไม้แผ่ทั่วของกราฟ (Spanning tree) คือ กราฟย่อยที่ทุกโหนดมีการเชื่อมต่อกันและไม่มีวงวน ซึ่งโดยทั่วไปนักคอมพิวเตอร์จะคุ้นเคยกับต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมน้อยสุด (Minimum spanning tree) ซึ่งสามารถหาต้นไม้นี้ได้จากขั้นตอนวิธีของพริม (Prim's algorithm) หรือ ขั้นตอนวิธีของครัสคัล (Kruskal's algorithm) รายละเอียดของขั้นตอนวิธีสามารถอ่านได้จากหนังสือเรียนวิชาโครงสร้างข้อมูลและอัลกอริทึมทั่วไป เช่น หนังสืออัลกอริทึม (Sedgewick, 2011) โดยหลักการคร่าว ๆ คือ ใช้ขั้นตอนวิธีเชิงละโมบ (Greedy algorithm) ในการเลือกเส้นเชื่อมที่มีค่าน้ำหนักน้อยที่สุดที่ไม่ทำให้เกิดวงวนในกราฟ โดยเลือกเส้นเชื่อมทีละเส้นจนทุกโหนดในกราฟมีเส้นเชื่อมถึงกันทั้งหมด

แต่ในงานวิจัยนี้ต้องการหาความสัมพันธ์เชื่อมโยงของซอร์สโค้ดที่มีความคล้ายกันมากที่สุด จึงจะใช้การหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดแทน (Maximum spanning tree) ซึ่งวิธีการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดสามารถใช้ขั้นตอนวิธีของครัสคัลได้โดยที่กำหนดให้ค่าน้ำหนักของเส้นเชื่อมแต่ละเส้นเป็นค่าติดลบแทน (Pemmaraju and Skiena, 2003, p. 336)

2.2 งานวิจัยที่เกี่ยวข้อง

ที่ผ่านได้มีความพยายามในการคิดค้นวิธีการในการตรวจจับความคล้ายกันของซอร์สโค้ด Agrawal และ Sharma (2016) ได้ศึกษาวิธีการตรวจจับการลอกเลียนซอร์สโค้ดจากงานวิจัยต่าง ๆ และได้แบ่งเทคนิคเป็น 5 กลุ่มใหญ่ ๆ ได้แก่ 1) การตรวจจับในระดับข้อความ (String) 2) การตรวจจับในระดับหน่วยคำ (Token) 3) การตรวจจับในระดับต้นไม้ไวยากรณ์ (Parse tree) 4) การ

ตรวจจับในระดับกราฟความขึ้นต่อกัน (Program dependency graph) และ 5) การตรวจจับโดยใช้ค่าจากมาตรวัดต่าง ๆ (Matrices) เช่น จำนวนลูป จำนวนเงื่อนไขในโปรแกรม

นักวิจัยจำนวนหนึ่งได้พัฒนาขั้นตอนวิธีในการตรวจจับความคล้ายกันของซอร์สโค้ดโดยตรวจจับที่ระดับต่าง ๆ เช่น Castro Campos และ Zaragoza Martinez (2012) ใช้ขั้นตอนวิธีการหาลำดับร่วมแบบยาวสุด (LCS: Longest common subsequence) ในการตรวจจับโค้ดที่คล้ายกันโดยพิจารณาที่ระดับของข้อความหรือตัวอักษรต่าง ๆ ที่ปรากฏในโปรแกรม ซึ่งแม้ว่าวิธี LCS นี้ปกติจะใช้เวลานานในการเปรียบเทียบ แต่บทความวิจัยนี้ก็ได้นำเสนอเทคนิคในการปรับปรุงวิธีการค้นหาให้เร็วขึ้นด้วยการใช้การค้นหาแบบแนวกว้างโดยเทคนิคการทำซ้ำ (Iterative breadth-first search) Son et al. (2013) นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Parse tree kernel) ซึ่งมีความสามารถในการตรวจโครงสร้างของโปรแกรม ซึ่งให้ประสิทธิภาพถึง 93% Zhao et al. (2015) ก็นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Abstract syntax tree) ซึ่งเป็นการวิเคราะห์โครงสร้างของโปรแกรม แล้วเปรียบเทียบค่าแฮช (Hash value) ของโปรแกรมว่ามีความคล้ายกันหรือไม่ Kamalim (2016) ใช้การตรวจจับในระดับไบต์โค้ด (Byte code) ซึ่งการตรวจจับแบบนี้มีข้อดีคือขั้นตอนวิธีจะพิจารณาจากโค้ดที่คอมไพล์แล้ว ซึ่งจะไม่สนใจเครื่องหมายวรรคตอน (Whitespace) และการตั้งชื่อตัวแปรที่ต่างกันก็จะไม่ส่งผล Qinqin และ Chunhai (2017) ได้รวบรวมนำเสนอมาตรวัดความคล้ายกันของซอร์สโค้ดต่าง ๆ เช่น การนับจำนวนแอททริบิว (Attribute counting method) การวัดโครงสร้างของโปรแกรม (Structural measurement method) การจัดกลุ่ม (Clustering method) เป็นต้น

ล่าสุดในปี 2018 Schneider et al. ได้นำเสนอแนวทางใหม่ในการตรวจจับการคัดลอกซอร์สโค้ดที่แตกต่างจากงานวิจัยที่มีมาก่อนหน้า กล่าวคืองานวิจัยในอดีตที่ผ่านมาจะตรวจจับจากโปรแกรมที่เขียนเสร็จแล้ว แต่งานวิจัยใหม่นี้จะเก็บข้อมูล (Log) จากการเขียนโปรแกรมของนักเรียนในชั้นเรียนตลอดกระบวนการเขียนโปรแกรม ซึ่งการใช้ข้อมูลจากพฤติกรรมในการเขียนโปรแกรมนี้สามารถตรวจจับการลอกกันของนักเรียนได้แม่นยำประมาณ 90% อย่างไรก็ตามการใช้วิธีการนี้ยังต้องการการปรับปรุง เนื่องจากประสบกับปัญหาข้อมูลปริมาณมาก และข้อมูลพฤติกรรมการเขียนโปรแกรมมีความซับซ้อนยากต่อการวิเคราะห์

จากขั้นตอนวิธีต่าง ๆ ที่นักวิจัยจำนวนมากได้นำเสนอ ก็มีผู้นำมาพัฒนาเป็นเครื่องมือสำหรับตรวจจับการคัดลอกซอร์สโค้ด หนึ่งในนั้นคือเครื่องมือที่ชื่อว่า MOSS (Measure Of Software Similarity) 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 ก่อนที่จะใช้ขั้นตอนวิธีที่นำเสนอในงานวิจัยนี้มาทำการระบุว่าซอร์สโค้ดใดเป็นต้นฉบับ โดยขั้นตอนวิธีที่ MOSS ใช้ในการเปรียบเทียบความคล้ายกันของซอร์สโค้ดจะเป็นการตรวจจับในระดับข้อความ (Token) กล่าวคือ MOSS จะวิเคราะห์หน่วยคำในโปรแกรมที่นำมาเปรียบเทียบ และระบุคู่ของโปรแกรมที่มีส่วนเหมือนกันมากของซอร์สโค้ด

2.3 วิธีดำเนินการวิจัย

งานวิจัยนี้นำเสนอวิธีการในการระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน พร้อมทั้งสร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน คณะผู้วิจัยได้นำเสนอกฎที่ใช้ในการระบุต้นตอ ซึ่งกฎนี้ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ mu+lambda ซึ่งมีรายละเอียดดังนี้

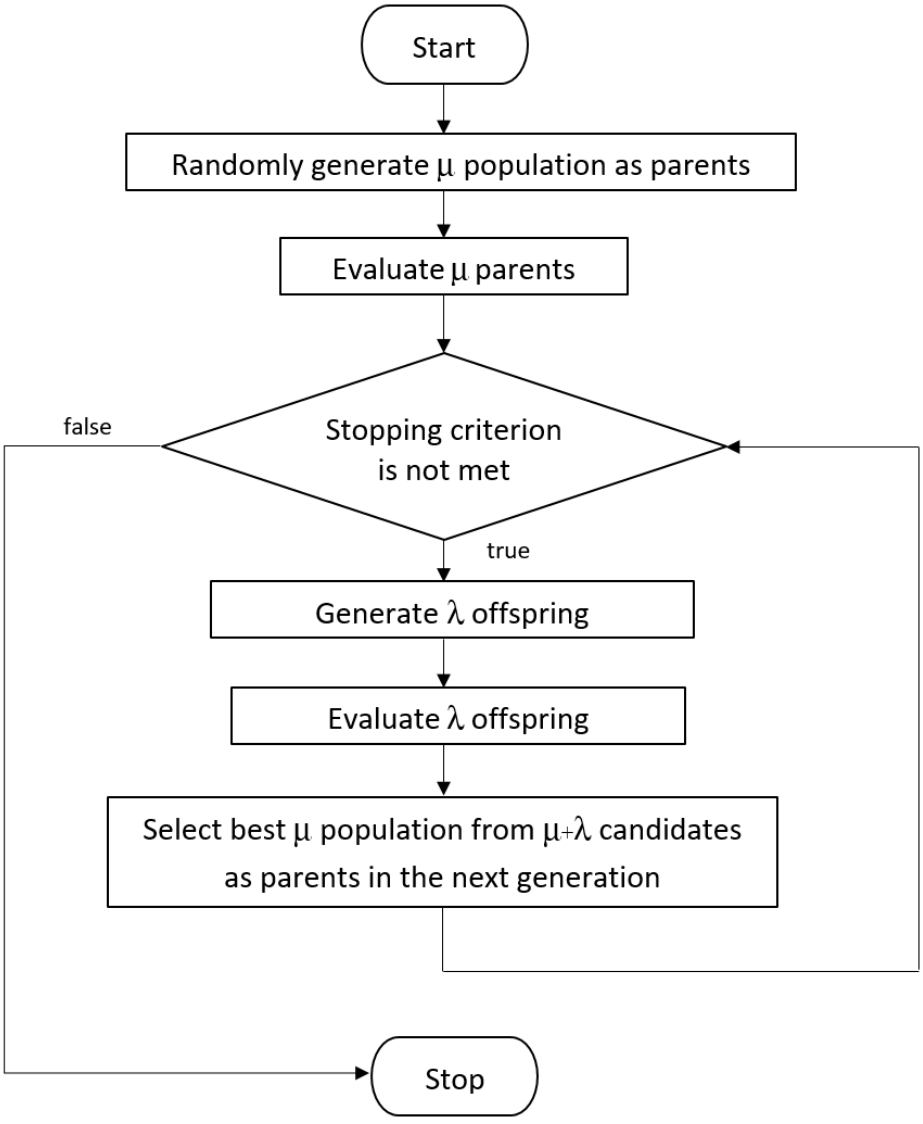
2.3.1 ขั้นตอนวิธีเชิงวิวัฒนาการแบบ mu+lambda

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

งานวิจัยได้นี้ ได้ใช้ขั้นตอนวิธีเชิงวิวัฒนาการรูปแบบหนึ่ง ที่เรียกว่าขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ โดยเหตุผลที่ผู้วิจัยเลือกใช้ขั้นตอนวิธีดังกล่าวในการสร้างกฎ (แทนที่จะใช้วิธีอื่น ๆ เช่น ต้นไม้ตัดสินใจ ข่ายงานประสาทเทียม หรือเทคนิคการเรียนรู้ของเครื่องจักรแบบอื่น ๆ) เนื่องจากผู้วิจัยต้องการสร้างกฎที่มนุษย์สามารถเข้าใจและอธิบายความหมายได้ แทนที่จะเป็นโมเดลที่เป็นค่าน้ำหนักในการตัดสินใจ และเหตุผลที่ไม่ใช้ต้นไม้ตัดสินใจเนื่องจากผู้วิจัยต้องการสร้างกฎที่ไม่ผูกติดอยู่กับค่าที่เป็นได้ของแอทริบิวต่าง ๆ แต่เป็นกฎที่เปรียบเทียบความสัมพันธ์ระหว่างแอทริบิวต่าง ๆ ที่นำมาพิจารณา

หลักการทำงานของขั้นตอนวิธี $\mu+\lambda$ คือ สร้างประชากรรุ่นแรก (ตัวอย่างคำตอบ) ขึ้นมา μ รูปแบบ จากนั้นประเมินค่าความเหมาะสมของประชากรแต่ละตัว แล้วสุ่มเลือกประชากรที่มีค่าความเหมาะสมที่ดีมาเป็นต้นแบบในการสร้างประชากรรุ่นลูกขึ้นมาอีก λ รูปแบบ แล้วจึงคัดเลือกประชากรที่มีค่าความเหมาะสมดีที่สุดโดยพิจารณาจากประชากรทั้งหมด $\mu+\lambda$ ตัว แล้วเลือก mu ตัวที่ดีที่สุดมาเป็นประชากรตั้งต้นในรุ่นถัดไป ขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ แสดงในรูป 2-6

รายละเอียดเพิ่มเติมเกี่ยวกับขั้นตอนวิธีเชิงวิวัฒนาการสามารถอ่านได้จาก Mitchell (1996), Goldberg (1989) และการวิเคราะห์พฤติกรรมการทำงานของขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ สามารถศึกษาได้จาก Ter-Sarkisov และ Marsland (2011)



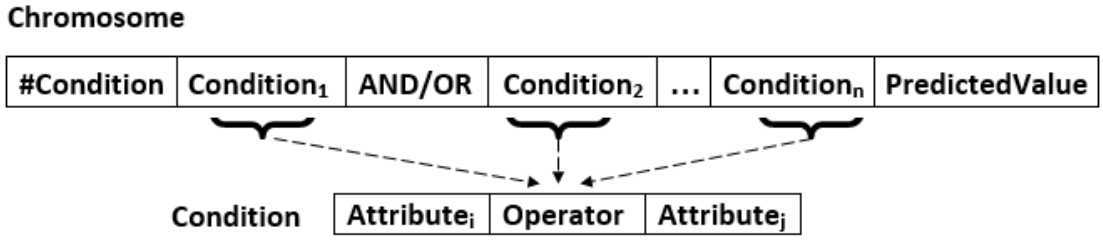
รูปที่ 2-6 mu+lambda evolutionary algorithm

2.3.2 การออกแบบขั้นตอนวิธี

งานวิจัยนี้นำเสนอวิธีการในการระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน พร้อมทั้งสร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน คณะผู้วิจัยได้นำเสนอกฎที่ใช้ในการระบุต้นตอ ซึ่งกฎนี้ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ ซึ่งมีรายละเอียดดังนี้

1) การเข้ารหัสโครโมโซมและการสร้างประชากร

คณะผู้วิจัยได้ออกแบบรูปแบบของกฎ แสดงดังรูปที่ 2-7 โดยแต่ละกฎประกอบด้วยพารามิเตอร์ที่จะถูกวิวัฒนาการ คือ ความยาวของกฎ (จำนวนเงื่อนไขที่ใช้เปรียบเทียบ), ค่าแอทริบิวที่นำพิจารณา, ตัวดำเนินการเปรียบเทียบ, ตัวดำเนินการทางตรรกะ



รูปที่ 2-7 Chromosome encoding

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
4. หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง
5. จำนวนบรรทัดของซอร์สโค้ดไฟล์ที่ 2
6. เปอร์เซ็นต์ความคล้ายที่ไฟล์ที่ 2 ไปเหมือนกับอีกไฟล์หนึ่ง
7. หมายเลขบรรทัดแรกที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง
8. หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

9. จำนวนบรรทัดที่ทั้งสองไฟล์เหมือนกัน

10. ผลเฉลยที่ระบุว่าไฟล์ใดคัดลอกมาจากไฟล์ใด หรือ ไม่ได้มีการคัดลอก

2) การประเมินค่าความเหมาะสม

ประชากรหรือกฎในการระบุต้นตอที่ถูกสุ่มสร้างขึ้นแต่ละอัน จะถูกประเมินค่าความเหมาะสม เพื่อจะได้ทราบว่ากฎใดให้ค่าความถูกต้องในการระบุต้นตอของซอร์สโค้ดได้มากกว่ากัน 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) ที่เผยแพร่ในอินเทอร์เน็ต15 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) ซอร์สโค้ดที่ทำงานเหมือนกัน แต่ใช้คนละวิธี คนละเทคนิคในการเขียนโปรแกรม

ในส่วนของขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ คณะผู้วิจัยได้ใช้พารามิเตอร์ต่าง ๆ แสดงในตาราง 2-2

ตาราง 2-2 Parameter settings

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

บทที่ 3
การเปรียบเทียบประสิทธิภาพของตัวชี้วัดความคล้าย

เนื้อหาในบทนี้นำเสนอการทดสอบประสิทธิภาพตัวชี้วัดความคล้ายในการตรวจสอบการคัดลอกงานของนักศึกษา รายละเอียดเป็นดังนี้

3.1 ขั้นตอนการเตรียมข้อมูล

ข้อมูลเพื่อที่จะนำไปทดสอบหาความคล้ายกันจะมีสองชุดคือ ข้อมูลที่มาจากไฟล์การบ้านของนักศึกษา และ ข้อมูลที่ได้จากการค้นหาจากเสิร์ชเอนจิน (Google) ผู้วิจัยจะแยกคำจากข้อมูลในเอกสารโดยตัดเครื่องหมายวรรคตอน อักขระพิเศษ และ stop word ออก จากนั้นจะจัดเก็บคำที่แยกได้ใน ตารางแฮช หรือเวกเตอร์ โดยการเลือกวิธีการเก็บข้อมูลนั้นก็จะขึ้นอยู่กับจุดประสงค์ในการนำข้อมูลไปใช้ในวิธีที่แตกต่างกัน เราจะจัดเก็บข้อมูลของคำที่แบ่งได้เข้าสู่ตารางแฮช ก็ต่อเมื่อตัวชี้วัดความคล้ายที่จะนำมาคำนวณหาความคล้ายนั้นให้ความสำคัญกับคำ หรือ ความถี่ของคำที่แบ่งได้จากเอกสาร ส่วนการจัดเก็บข้อมูลลงในเวกเตอร์ จะนำมาใช้ ก็ต่อเมื่อตัวชี้วัดความคล้ายของเรานั้นไม่ได้สนใจความถี่ของคำแต่ละคำโดยเฉพาะ แต่จะสนใจที่ลำดับตำแหน่งของคำแทน

การเตรียมข้อมูลจาก Search Engine (Google) นั้นเราจะทำการค้นหาโดยใช้ข้อมูลที่ได้จากไฟล์เอกสาร เช่น สายอักขระที่นำมาจากในเอกสารหรือคำในเอกสารที่มีความถี่ของคำมากที่สุด 10 อันดับ เป็นต้น เพื่อให้ได้ยูอาร์แอลที่คาดว่าน่าจะตรงกับข้อมูลในเอกสาร โดยวิธีการเลือกประโยคหรือคำเพื่อนำไปค้นหาโดยใช้ Google นั้น ในงานวิจัยนี้ได้เลือกวิธีคัดเลือกคำที่จะใช้ อยู่สามวิธี

วิธีแรกคือ ค้นหาโดยใช้คำที่แยกได้มาเรียงต่อกันเป็นประโยค โดยรูปแบบข้อมูลที่เรานำมาใช้คือข้อมูลที่มีรูปแบบการเก็บข้อมูลแบบเวกเตอร์ เพราะการเก็บข้อมูลในเวกเตอร์ จะให้ความสำคัญกับตำแหน่งของคำ

วิธีที่สองคือ การใช้คำมาต่อกันเป็นสายอักขระแบบวิธีแรก แต่เพิ่มวิธีการสุ่มตำแหน่งของสายอักขระ โดยเราจะทำการสุ่มตำแหน่งเริ่มต้นของส่วนที่เราจะนำมาเป็นสายอักขระ จากนั้น ทำการแบ่งเอกสารออกเป็น 3 ส่วน คือ ส่วนต้น ส่วนกลาง และส่วนท้ายของเอกสาร ซึ่งการใช้วิธีนี้ จะทำให้ได้ข้อมูลที่จะนำไปค้นหานั้นมีความหลากหลายมากขึ้น

วิธีที่สามคือ การใช้คำที่มีความถี่มากที่สุดในเอกสารจำนวน 16 คำ มาต่อกันโดยใช้เพิ่มเครื่องหมายบวก “+” ระหว่างคำ (การค้นหาของ Google นั้น การเพิ่มเครื่องหมายบวก “+” ระหว่างคำ คือการค้นหาตามคำที่มีนั้นทุกประการ)

3.2 การเปรียบเทียบความคล้ายคลึงกันของข้อมูล

เมื่อได้ข้อมูลจากวิธีการขั้นต้นแล้ว ต่อไปก็จะเข้าสู่ขั้นตอนการเปรียบเทียบข้อมูลที่ได้จากไฟล์ข้อมูลอินพุตและข้อมูลที่ได้จาก Google โดยในงานวิจัยนี้ได้ทดลองใช้ตัวชี้วัดความคล้าย 5 แบบ ดังนี้

3.2.1 Jaccard Index

การเปรียบเทียบข้อมูลโดยวิธี Jaccard Index นั้นเป็นการเปรียบเทียบโดยสนใจคำที่แยกได้จากข้อมูล กับจำนวนคำทั้งหมดของข้อมูล โดยขั้นตอนแรกเราจะนำข้อมูลจากตารางแฮช มาเก็บไว้ในแถวลำดับ เพื่อที่จะนำไปเปรียบเทียบ เมื่อได้แถวลำดับมาแล้วจึงนำแถวลำดับทั้งสองมาหาว่ามีคำใดบ้างที่เหมือนกันซึ่งก็คือการอินเตอร์เซกชันกันของข้อมูลทั้งสองเซต

3.2.2 Sorensen Index

การเปรียบเทียบข้อมูลโดยวิธี Sorensen Index นั้นจะให้ความสำคัญกับคำที่มีในเอกสาร แต่ไม่สนใจความถี่ของคำ และสนใจกับจำนวนทั้งหมดของคำของทั้งสองเอกสาร โดยเราจะใช้ข้อมูลที่เก็บอยู่ในตารางแฮชมาใช้ เนื่องจากว่า เราสนใจแต่คำในเอกสารนั้น ๆ แต่ไม่ได้สนใจกับลำดับตำแหน่งของคำ จากนั้นเรานำข้อมูลทั้งหมดมาเก็บไว้ในแถวลำดับ เพื่อที่จะนำไปเปรียบเทียบต่อไป

3.2.3 Tanimoto coefficient

การเปรียบเทียบข้อมูลโดยตัวชี้วัดความคล้าย Tanimoto Coefficient จะเปรียบเทียบโดยให้ความสำคัญกับจำนวนความถี่ของคำแต่ละคำในเอกสาร โดยจะทำการเปรียบเทียบโดยนำความถี่ของคำที่เหมือนกันมาคำนวณ โดยจะนำข้อมูลจากตารางแฮชมาใช้งาน และจะเปรียบเทียบโดยนำข้อมูลที่ค้นหาได้จาก Google เป็นตัวกำหนดหลัก ถ้าคำที่อยู่ในข้อมูลอินพุตนั้น ไม่มีอยู่ในข้อมูลที่ได้จาก Google จะกำหนดให้ค่าความถี่นั้นเป็น 0

3.2.4 TF-IDF Weight

การเปรียบเทียบโดยวิธี TF-IDF Weight เป็นการเปรียบเทียบโดยให้ความสำคัญกับความถี่ของคำในเอกสารเช่นเดียวกับ วิธี Tanimoto Coefficient แต่ TF-IDF จะเป็นวิธีปรับค่าน้ำหนักของคำในเอกสารวิธีการคำนวณโดยเราจะนำข้อมูลจากตารางแฮชมาใช้ โดยค่าน้ำหนักนั้นจะประกอบด้วยสองส่วน ส่วนแรก คือ TF (Term Frequency) และ IDF (Inverse Document Frequency)

3.2.5 Jaro-Winkler Distance

การเปรียบเทียบข้อมูลโดยวิธี Jaro-Winkler นั้นเราจะเปรียบเทียบโดยให้ความสำคัญกับลำดับของคำในเอกสาร โดยไม่สนใจความถี่ของคำ ดังนั้นเราจึงใช้เวกเตอร์ในการจัดเก็บข้อมูลเพื่อนำข้อมูลมาเปรียบเทียบ ในที่นี้จะใช้เทคนิค k-gram ในการเลือกความยาวของคำเพื่อที่จะนำมาเปรียบเทียบในตัวอย่างนี้จะกำหนดให้แบ่งเป็นข้อมูลละ 3 คำ (Tri-gram) หลังจากได้ผลลัพธ์จากการคำนวณมาแล้ว เราก็จะทำการตรวจสอบว่า ค่าที่คำนวณได้มีค่ามากกว่าค่าขีดแบ่ง (Threshold) หรือไม่ โดยในงานวิจัยนี้ได้ทดลองใช้ค่าขีดแบ่งหลาย ๆ ระดับ

3.3 ผลการทดลอง

การทดสอบตัวชี้วัดความคล้ายในงานวิจัยนี้ จะใช้ไฟล์เอกสารจากแบบฝึกหัด หรือการบ้านที่นักศึกษาใช้ส่งอาจารย์จริง โดยจะทำการทดสอบสองแบบ แบบแรกคือทดสอบกลุ่มเอกสารเพื่อหาว่าไฟล์ใดที่คัดลอกกันมา โดยการทดสอบนี้จะไม่นำส่วนข้อมูลที่เหมือนกันไปค้นหาข้อมูลใน Google ต่อ แบบที่สองคือเปรียบเทียบไฟล์เอกสารสองไฟล์ โดยเริ่มจากการทดสอบหาความคล้ายกันของไฟล์ทั้งสอง ถ้าไฟล์ทั้งสองนั้นมีส่วนของข้อมูลที่เหมือนกัน เราอาจสรุปว่านักศึกษาคัดลอกเนื้อหาของกันและกัน หรือ นักศึกษาคัดลอกเนื้อหามาจากหน้าเว็บเดียวกัน แล้วค้นหาข้อมูลใน Google ต่อ

3.3.1 ผลการทดสอบความคล้ายกันของกลุ่มเอกสาร

ในการทดสอบนี้เราจะนำตัวชี้วัดความคล้ายแต่ละวิธีมาหาค่าความคล้ายคลึงของเอกสาร ซึ่งจะทำการเปรียบเทียบเอกสารทั้งหมด 3 กลุ่มตัวอย่างด้วยกัน โดยจะทำการเปรียบเทียบทุกไฟล์ ตัวอย่างกลุ่มแรกมีไฟล์ข้อมูลทั้งหมด 41 ไฟล์ ตัวอย่างกลุ่มที่สองมีไฟล์ข้อมูลทั้งหมด 45 ไฟล์ และตัวอย่างกลุ่มที่สามมีไฟล์ข้อมูลทั้งหมด 40ไฟล์ เราจะทำการทดสอบด้วยตัวชี้วัดความคล้ายวิธีต่าง ๆ โดยกำหนดระดับค่าความคล้ายคลึงที่จะยอมรับว่าไฟล์นั้นเหมือนกันหรือค่าขีดแบ่ง (Threshold) คือตั้งแต่ 89.99 เปอร์เซ็นต์ขึ้นไป

จากการตรวจสอบไฟล์เอกสารทั้งสามกลุ่ม โดยในแต่ละกลุ่มจะนำไฟล์ทั้งหมดมาเปรียบเทียบกันทั้งหมดทุกไฟล์ ผลทดสอบแต่ละตัวชี้วัดความคล้าย ดังตารางที่ 3-1 พบว่าวิธีที่ให้จำนวนเอกสารที่คล้ายกันมากที่สุดได้แก่ Sorensen, Tanimoto และ Jaccard วิธีที่ให้จำนวนเอกสารที่มีความคล้ายกันน้อยสุดคือ Jaro-Winkler ซึ่งเมื่อตรวจสอบข้อมูลด้วยตาเปล่าแล้วพบว่าวิธีทดสอบที่ใช้ตัวชี้วัดความคล้าย Jaccard, Sorensen และ Tanimoto ไฟล์ที่มีค่าความคล้ายคลึงกันตั้งแต่ 96 ถึง 100 เปอร์เซ็นต์ นั้นไฟล์ข้อมูลจะมีความคล้ายกันมาก แต่ไฟล์ข้อมูลที่มีค่าความคล้ายคลึงตั้งแต่ 96 เปอร์เซ็นต์ลงไป เมื่อตรวจสอบข้อมูลในไฟล์ดูแล้ว บางไฟล์ไม่ได้มีข้อมูลที่คล้ายกันจนถือว่าคัดลอกกัน

มา ส่วนผลที่ได้จากวิธี Jaro-Winkler ไฟล์ข้อมูลที่มีค่าความคล้ายคลึงกันตั้งแต่ 92 เปอร์เซ็นต์ขึ้นไป เมื่อตรวจสอบข้อมูลในไฟล์แล้วข้อมูลในไฟล์มีความเหมือนกันมาก ส่วนคะแนนที่ต่ำกว่า 92 เปอร์เซ็นต์ลงไปนั้น ข้อมูลในไฟล์ก็จะมีบางส่วนเท่านั้นที่คล้ายกัน

ตาราง 3-1 การเปรียบเทียบตัวชี้วัดความคล้าย

ตัววัดความคล้าย	ชุดข้อมูล	90 - 91.99%	92 - 93.99%	94 - 95.99%	96 - 97.99%	98 - 100%
Jaccard	1	22	7	4	2	9
	2	39	15	7	17	70
	3	12	40	40	41	33
Sorensen	1	22	17	34	11	11
	2	73	108	136	22	87
	3	14	60	34	11	11
Tanimoto	1	19	15	18	20	16
	2	7	15	65	108	336
	3	1	44	18	21	34
TF-IDF	1	0	0	0	0	0
	2	0	0	0	0	0
	3	0	0	0	0	0
Jaro-Winkler	1	0	1	0	0	5
	2	35	31	4	1	29
		10	1	1	1	16

ข้อสังเกต ไฟล์ข้อมูลบางไฟล์มีการสลับลำดับของคำตอบแต่ตัวชี้วัดความคล้าย Jaccard Sorensen และ Tanimoto ก็ยังสามารถให้ค่าคะแนนความคล้ายที่สูง แต่ในตัวชี้วัดความคล้าย Jaro-Winkler กลับให้คะแนนความคล้ายน้อยกว่า

3.3.2 ผลการทดสอบการเปรียบเทียบเอกสารโดยนำข้อมูลที่เหมือนกันไปค้นหาในเสิร์ชเอนจิน

วิธีการทดลองนี้เริ่มจากการทดสอบหาความคล้ายกันของไฟล์เอกสารทั้งสอง ถ้าไฟล์ทั้งสองนั้นมีส่วนของข้อมูลที่เหมือนกัน เราอาจสรุปว่านักศึกษาคัดลอกเนื้อหาของกันและกัน หรือ นักศึกษาคัดลอกเนื้อหามาจากหน้าเว็บเดียวกัน ซึ่งเราจะทำการทดสอบต่อไป โดยจะทำการนำข้อมูลที่เหมือนกันนั้น ไปค้นหาใน Google เพื่อที่จะทำการทดสอบว่าข้อมูลส่วนนี้นั้นคัดลอกมาจากเว็บอะไร จากนั้นเมื่อได้ยูอาร์แอล จากการค้นหาแล้ว จึงนำไฟล์ทั้งสองไปเปรียบเทียบกับยูอาร์แอลที่ค้นหาได้ เพื่อหาว่ามีความเอกสารมีความคล้ายคลึงกับข้อมูลยูอาร์แอล หรือไม่

เมื่อเราได้ส่วนที่เหมือนกันจากไฟล์เอกสารทั้งสองแล้ว จากนั้นจึงนำข้อมูลในส่วนที่เหมือนกันไปค้นหาข้อมูลจาก Google ซึ่งจะได้ยูอาร์แอลผลลัพธ์จากข้อมูลที่เราใช้ค้นหา จากนั้นเราจะทำการเปรียบเทียบไฟล์เอกสารทั้งสองกับข้อมูลที่ค้นหาได้ โดยผลการเปรียบเทียบไฟล์ และ การทดสอบหาความคล้ายกับข้อมูลจากยูอาร์แอลต่าง ๆ เราสามารถสรุปผลการทดสอบได้ดังที่จะวิเคราะห์ในหัวข้อถัดไป

3.3.3 วิเคราะห์คุณลักษณะของตัวชี้วัดความคล้ายวิธีต่าง ๆ

3.3.3.1 Jaccard Index

การทดสอบโดยใช้ตัวชี้วัดความคล้าย Jaccard Index ในการทดสอบหาความคล้ายคลึงกันนั้น Jaccard Index จะไม่สนใจในลำดับตำแหน่งของคำ และ ไม่สนใจความถี่ของคำในเอกสาร ในการทดสอบนั้นถ้าข้อมูลที่นำมาทดสอบมีขนาดของข้อมูลที่ใกล้เคียงกัน ตัวชี้วัดนี้สามารถบอกค่าความคล้ายคลึงกันได้ดี ถึงแม้ว่าข้อมูลที่นำมาเปรียบเทียบกันนั้นจะมีการสลับลำดับตำแหน่งของคำตอบ

แต่การที่ตัวชี้วัดนี้ไม่ได้สนใจกับลำดับตำแหน่งของคำหรือความถี่ของคำ ข้อมูลที่นำมาทดสอบบางอย่างจึงไม่สามารถสรุปได้ว่า ข้อมูลในเอกสารที่นำมาทดสอบนั้นมีความสัมพันธ์กันหรือไม่

3.3.3.2 Sorensen Index

การทดสอบโดยตัวชี้วัดความคล้าย Sorensen Index นี้จะมีความคล้ายกับ Jaccard Index เช่น ถ้าขนาดของข้อมูลใกล้เคียงกันก็สามารถให้ผลลัพธ์ในระดับที่ดี แต่ถ้าข้อมูลที่นำมาทดสอบนั้นมีขนาดแตกต่างกันมาก ค่าความคล้ายที่ได้จะได้เปอร์เซ็นต์ที่ไม่สูงมากนัก เช่นเดียวกับ Jaccard Index ตัวชี้วัดความคล้าย Sorensen นั้นไม่ได้ให้ความสนใจกับลำดับตำแหน่งหรือความถี่ของคำ จึงทำให้ข้อมูลบางอย่างไม่สามารถสรุปได้ว่า ข้อมูลในเอกสารที่นำมาทดสอบนั้นมีความสัมพันธ์กันหรือไม่

3.3.3.3 Tanimoto coefficient

การทดสอบโดยตัวชี้วัดความคล้าย Tanimoto coefficient นี้ตัวชี้วัดความคล้ายจะให้ความสำคัญกับจำนวนความถี่ของคำที่นำมาคำนวณ ซึ่งถ้าเอกสารทั้งสองมีความถี่ที่ใกล้เคียงกัน ตัวชี้วัดความคล้ายคลึงนี้ จะได้ค่าความคล้ายคลึงที่ดี และถ้าจำนวนของคำทั้งหมดของทั้งสองเอกสารไม่ต่างกันมากค่าความคล้ายคลึงจะมีค่ามากขึ้น

3.3.3.4 TF-IDF Weight

การทดสอบโดยใช้วิธี TF-IDF Weight นั้น เมื่อนำมาใช้กับการเปรียบเทียบเอกสารแค่สองเอกสารนั้นไม่สามารถให้ค่าความคล้ายคลึงออกมาได้เนื่องจากตามวิธีคำนวณค่า idf นั้น คำที่ปรากฏในทุก ๆ เอกสารนั้นจะถือว่าคำนั้นมีความสำคัญน้อยหรือไม่มีความสำคัญ

3.3.3.5 Jaro-Winkler distance

การทดสอบโดยตัวชี้วัดความคล้ายด้วยวิธีนี้นั้น ในการตรวจสอบนั้นจะให้ความสำคัญกับลำดับของคำ ถ้าข้อมูลที่นำมาตรวจสอบไม่ได้มีการดัดแปลง หรือ เป็นข้อมูลที่ไม่ใช่ข้อมูลที่ได้มาจากหลาย ๆ ข้อมูลรวมกัน การตรวจสอบด้วยวิธีนี้นั้น จะให้ค่าความคล้ายที่สูงประสิทธิภาพมาก และเนื่องจากการที่มีการให้ความสำคัญกับลำดับของคำทำให้สามารถบอกได้ว่า ข้อมูลนั้น มีความสัมพันธ์กัน

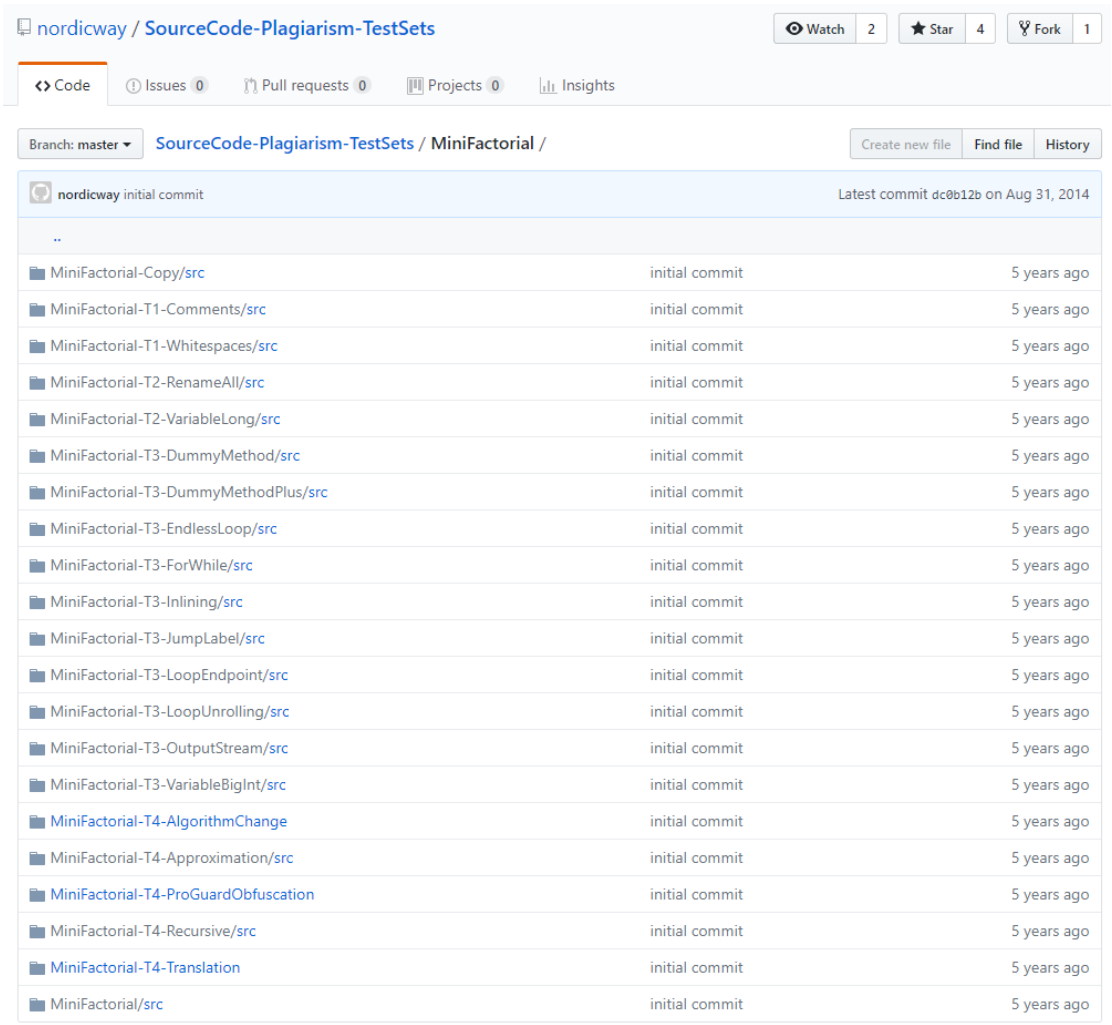
บทที่ 4
การระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน

เนื้อหาในบทนี้นำเสนอการระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน พร้อมทั้งสร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน รายละเอียดเป็นดังนี้

4.1 ข้อมูลที่ใช้ในการทดลอง

งานวิจัยนี้ใช้ข้อมูลการลอกเลียนซอร์สโค้ดจากข้อมูลทดสอบ (Benchmark data) ที่เผยแพร่ในอินเทอร์เน็ต ซึ่งชุดข้อมูลดังกล่าวประกอบด้วยไฟล์ต้นฉบับและซอร์สโค้ดที่มีการลอกเลียนมาจากต้นฉบับจำนวน 21 โปรแกรม ดังรูปที่ 4-1 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) ซอร์สโค้ดที่ทำงานเหมือนกัน แต่ใช้คนละวิธี คนละเทคนิคในการเขียนโปรแกรม



รูปที่ 4-1 ชุดข้อมูลที่ใช้ในการทดลอง

4.2 ผลการตรวจสอบความคล้ายกันของซอร์สโค้ด

ผู้วิจัยได้นำข้อมูลดังที่ได้กล่าวไปในหัวข้อก่อนหน้า มาเปรียบเทียบความคล้ายกันโดยใช้ MOSS พร้อมทั้งรวบรวมคุณลักษณะ (Attribute) ของแต่ละไฟล์ ดังแสดงในตารางที่ 4-2

รายละเอียดของข้อมูลในแต่ละคอลัมน์มีดังนี้

คอลัมน์ที่ 1 คือ ชื่อโปรแกรมที่ 1 ที่นำมาเปรียบเทียบ

คอลัมน์ที่ 2 คือ จำนวนบรรทัดของโค้ดในโปรแกรมที่ 1

คอลัมน์ที่ 3 คือ เปอร์เซ็นต์ความคล้ายที่ไฟล์ที่ 1 ไปเหมือนกับอีกไฟล์หนึ่ง

คอลัมน์ที่ 4 คือ หมายเลขบรรทัดแรกที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

คอลัมน์ที่ 5 คือ หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง

คอลัมน์ที่ 6 คือ ชื่อโปรแกรมที่ 2 ที่นำมาเปรียบเทียบ
คอลัมน์ที่ 7 คือ จำนวนบรรทัดของซอร์สโค้ดไฟล์ที่ 2
คอลัมน์ที่ 8 คือ เปอร์เซ็นต์ความคล้ายที่ไฟล์ที่ 2 ไปเหมือนกับอีกไฟล์หนึ่ง
คอลัมน์ที่ 9 คือ หมายเลขบรรทัดแรกที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง
คอลัมน์ที่ 10 คือ หมายเลขบรรทัดสุดท้ายที่ตรวจจับได้ว่าเหมือนกับอีกไฟล์หนึ่ง
คอลัมน์ที่ 11 คือ จำนวนบรรทัดที่ทั้งสองไฟล์เหมือนกัน

ตาราง 4-2 ผลการเปรียบเทียบความคล้ายกันของซอร์สโค้ดและค่าคุณสมบัติต่าง ๆ

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial.java	12	88	2	9	MiniFactorial-T2-RenameAll.java	12	88	2	9	8
MiniFactorial.java	12	88	2	9	MiniFactorial-T1-Whitespaces.java	13	88	2	12	11
MiniFactorial.java	12	88	2	9	MiniFactorial-T1-Comments.java	35	88	9	31	23

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T1-Whitespaces.java	13	88	2	12	MiniFactorial-T2-RenameAll.java	12	88	2	9	11
MiniFactorial-T1-Comments.java	35	88	9	31	MiniFactorial-T2-RenameAll.java	12	88	2	9	23
MiniFactorial-T1-Comments.java	35	88	9	31	MiniFactorial-T1-Whitespaces.java	13	88	2	12	23
MiniFactorial-Copy.java	12	88	2	9	MiniFactorial.java	12	88	2	9	8
MiniFactorial-Copy.java	12	88	2	9	MiniFactorial-T2-RenameAll.java	12	88	2	9	8

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-Copy.java	12	88	2	9	MiniFactorial-T1-Whitespaces.java	13	88	2	12	11
MiniFactorial-Copy.java	12	88	2	9	MiniFactorial-T1-Comments.java	35	88	9	31	23
MiniFactorial.java	12	79	2	9	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	8
MiniFactorial.java	12	79	2	9	MiniFactorial-T3-DummyMethod.java	18	67	2	9	8

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T3-DummyMethod.java	18	67	2	9	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	8
MiniFactorial-T2-RenameAll.java	12	79	2	9	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	8
MiniFactorial-T2-RenameAll.java	12	79	2	9	MiniFactorial-T3-DummyMethod.java	18	67	2	9	8
MiniFactorial-T1-Whitespaces.java	13	79	2	9	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	8

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T1-Whitespaces.java	13	79	2	9	MiniFactorial-T3-DummyMethod.java	18	67	2	9	8
MiniFactorial-T1-Comments.java	35	79	9	29	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	21
MiniFactorial-T1-Comments.java	35	79	9	29	MiniFactorial-T3-DummyMethod.java	18	67	2	9	21
MiniFactorial-Copy.java	12	79	2	9	MiniFactorial-T3-LoopEndpoint.java	12	76	2	9	8

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-Copy.java	12	79	2	9	MiniFactorial-T3-DummyMethod.java	18	67	2	9	8
MiniFactorial-T3-EndlessLoop.java	18	63	2	9	MiniFactorial-T3-ForWhile.java	16	68	2	9	8
MiniFactorial.java	12	56	4	9	MiniFactorial-T3-OutputStream.java	16	43	8	13	6
MiniFactorial.java	12	56	2	8	MiniFactorial-T3-ForWhile.java	16	56	2	8	7
MiniFactorial.java	12	56	2	8	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	7

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T3-ForWhile.java	16	56	2	8	MiniFactorial-T3-LoopEndpoint.java	12	54	2	8	7
MiniFactorial-T3-EndlessLoop.java	18	53	2	8	MiniFactorial-T3-LoopEndpoint.java	12	54	2	8	7
MiniFactorial-T3-DummyMethod.java	18	48	2	8	MiniFactorial-T3-ForWhile.java	16	56	2	8	7
MiniFactorial-T3-DummyMethod.java	18	48	2	8	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	7
MiniFactorial-T2-RenameAll.java	12	56	4	9	MiniFactorial-T3-OutputStream.java	16	43	8	13	6

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T2-RenameAll.java	12	56	2	8	MiniFactorial-T3-ForWhile.java	16	56	2	8	7
MiniFactorial-T2-RenameAll.java	12	56	2	8	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	7
MiniFactorial-T1-Whitespaces.java	13	56	4	12	MiniFactorial-T3-OutputStream.java	16	43	8	13	9
MiniFactorial-T1-Whitespaces.java	13	56	2	6	MiniFactorial-T3-ForWhile.java	16	56	2	8	7
MiniFactorial-T1-Whitespaces.java	13	56	2	6	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	7

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-T1-Comments.java	35	56	17	31	MiniFactorial-T3-OutputStream.java	16	43	8	13	15
MiniFactorial-T1-Comments.java	35	56	9	27	MiniFactorial-T3-ForWhile.java	16	56	2	8	19
MiniFactorial-T1-Comments.java	35	56	9	27	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	19
MiniFactorial-Copy.java	12	56	4	9	MiniFactorial-T3-OutputStream.java	16	43	8	13	6
MiniFactorial-Copy.java	12	56	2	8	MiniFactorial-T3-ForWhile.java	16	56	2	8	7

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
MiniFactorial-Copy.java	12	56	2	8	MiniFactorial-T3-EndlessLoop.java	18	53	2	8	7

4.3 กฎในการระบุต้นตอของซอร์สโค้ด

ผู้วิจัยได้ทำการทดลองสร้างกฎตามวิธีการที่ได้อธิบายในหัวข้อก่อนหน้า เพื่อระบุต้นตอของซอร์ส โค้ดที่มีความคล้ายกัน โดยกฎที่ได้แสดงดังตารางที่ 4-3

ตาราง 4-3 กฎในการระบุต้นตอของซอร์ส โค้ด

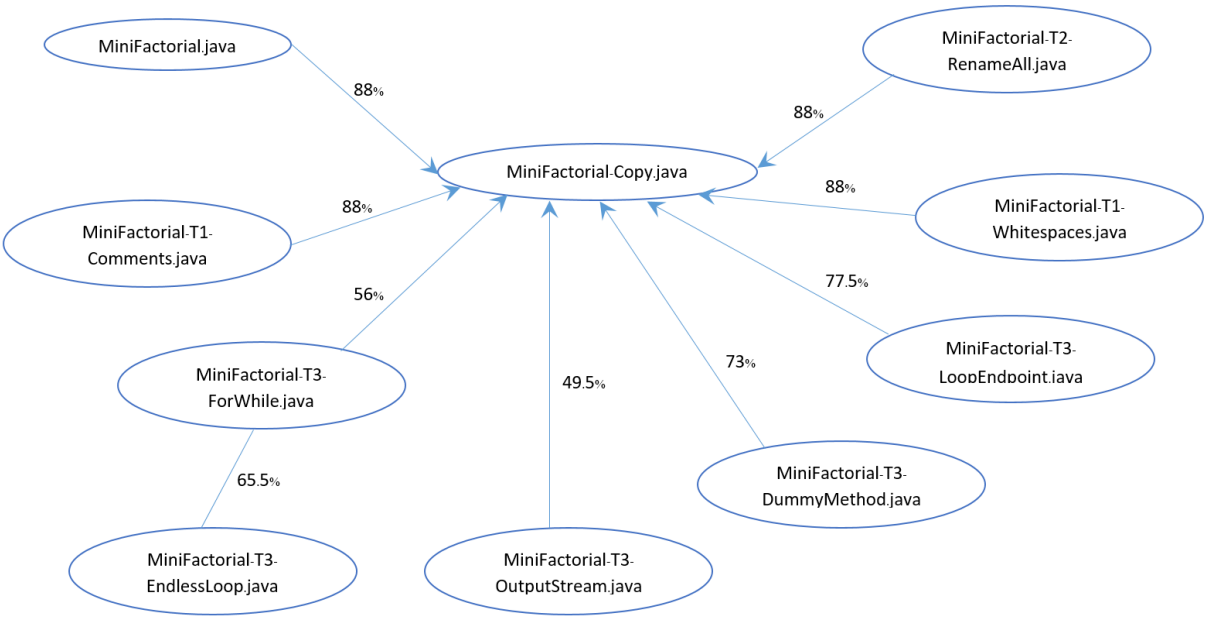
Rule for identifying the original source code	% correct
IF endMatched_File1 < endMatched_File2 OR percentMatched_File1 > percentMatched_File2 **THEN** File1 is original **ELSE IF** endMatched_File1 > endMatched_File2 **THEN** File2 is original **ELSE IF** percentMatched_File1 = percentMatched_File2 **THEN**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

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

จากการใช้กฎดังกล่าวระบุไฟล์ที่มีการคัดลอก พบว่าสามารถระบุไฟล์ต้นฉบับและระบุไฟล์ที่ไม่ได้ลอกเลียนได้ถูกต้อง 90.24%

4.4 ต้นไม้แสดงวิวัฒนาการในการคัดลอกซอร์สโค้ด

เมื่อระบุไฟล์ต้นฉบับได้แล้ว ผู้วิจัยได้สร้างต้นไม้แสดงความสัมพันธ์ และระบุทิศทางของลูกศรเพื่อแสดงลำดับการสืบทอดของซอร์สโค้ดที่มีความคล้ายกัน ซึ่งต้นไม้นี้สร้างได้โดยการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุด แล้วระบุทิศทางโดยใช้กฎที่นำเสนอข้างต้น สำหรับชุดข้อมูลทดสอบนี้สามารถสร้างต้นไม้แสดงความสัมพันธ์ของการลอกเลียนซอร์สโค้ดได้ดังรูปที่ 4-2 ซึ่งจะเห็นได้ว่าไฟล์ต่าง ๆ ส่วนใหญ่แล้วคัดลอกมาจากต้นฉบับ คือ ไฟล์ชื่อ MiniFactorial-Copy.java



รูปที่ 4-2 Phylogenetic tree of source code plagiarism

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-T3-EndlessLoop.java ซึ่งจากต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด ได้เลือกเส้นเชื่อมระหว่างไฟล์คู่นี้ไว้ ซึ่งทั้งสองไฟล์ก็มีความคล้ายกัน แต่ไม่ได้ลอกกันมาโดยตรง เส้นเชื่อมระหว่างไฟล์คู่นี้จึงไม่มีหัวลูกศรระบุต้นฉบับ

บทที่ 5
สรุปผลการดำเนินการและผลผลิต

งานวิจัยนี้นำเสนอขั้นตอนวิธีสำหรับระบุต้นตอของการคัดลอกซอร์สโค้ด โดยได้ศึกษาตัวชี้วัดความคล้าย และได้ทดลองสร้างกฎในการระบุต้นฉบับ

การทดลองเปรียบความคล้ายคลึงของเอกสารและข้อมูลที่สืบค้นได้จาก Google โดยวัดผลจากคะแนนที่ได้ในแต่ละตัวชี้วัดความคล้าย เพื่อบ่งบอกว่าเอกสารใดมีการคัดลอกข้อมูลมาโดยตรงหรือไม่ ผลที่ได้น่าพึงพอใจ แต่เนื่องจากข้อมูลที่นำมาทดสอบนั้นบางข้อมูลมีขนาดแตกต่างกันมากจึงทำให้ค่าความคล้ายที่ได้ไม่สูงมากนัก เช่น ใน ตัวชี้วัดความคล้าย Jaccard Index, Sorensen Index เป็นต้น และ วิธีการหาความคล้ายโดยใช้ตัวชี้วัดความคล้าย TF-IDF นั้น ไม่สามารถหาค่าความคล้ายคลึงของสองเอกสารได้จึงไม่สามารถให้ผลลัพธ์ได้ แต่จากตัวชี้วัดความคล้าย Jaro-Winkler ให้ผลลัพธ์ที่น่าพอใจมาก เนื่องจากสามารถตรวจสอบข้อมูลจากตัวอย่างได้ เป็นเปอร์เซ็นต์ที่สูง โดยสามารถบอกได้ว่าข้อมูลที่นำมาทดสอบนั้นคัดลอกมาจากเว็บไซต์อะไร

ส่วนผลการทดลองสร้างกฎในการระบุต้นตอของซอร์สโค้ดนั้น แสดงให้เห็นว่ากฎที่นำเสนอที่ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ mu+lambda ให้ค่าความถูกต้องในการระบุการลอกเลียนซอร์สโค้ด 90.24% ซึ่งได้ทดสอบกับชุดข้อมูลที่มีทั้งการเปลี่ยนแปลงซอร์ส โค้ดแบบเล็กน้อย และซอร์สโค้ดที่มีการเปลี่ยนวิธีการในการเขียนโปรแกรม ต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดการลอกเลียนซอร์สโค้ดที่สร้างขึ้นนี้ จะเป็นเครื่องมือหนึ่งที่ช่วยให้อาจารย์เห็นภาพรวมของความคล้ายกันของงานเขียนโปรแกรมที่เป็นการบ้านที่นักเรียนส่งมา รวมทั้งได้ข้อสังเกตเกี่ยวกับว่าไฟล์ใดน่าจะเป็นต้นฉบับในการลอกเลียนซอร์สโค้ดมาส่ง เพื่อที่ผู้สอนจะได้ตักเตือนและให้คำแนะนำที่เหมาะสมกับนักเรียนต่อไป อย่างไรก็ตาม งานวิจัยนี้ได้นำเสนอผลการทดลองเบื้องต้น ซึ่งใช้ข้อมูลมาตรฐานที่เป็นข้อมูลทดสอบที่เผยแพร่ให้ใช้ในงานวิจัยทั่วไป (Benchmark) ในอนาคตผู้วิจัยวางแผนจะรวบรวมข้อมูลจริงและปรับปรุงขั้นตอนวิธีให้มีประสิทธิภาพมากขึ้น

ผลงานตีพิมพ์ในที่ประชุมวิชาการระดับชาติ ดังรายละเอียดต่อไปนี้:-

Kulrattanavijitra, T., Leelathakul, N., & Rimcharoen, S. (2018). Effecttiveness of similarity matrics for detecting students' assignment plagiarism, National Conference on Computing and Information Technology, July 5-6. (Thai)

ผลงานตีพิมพ์ในวารสารวิชาการระดับชาติ ดังรายละเอียดต่อไปนี้:-

Leelathakul N., & Rimcharoen, S. (2019). Phylogenetic trees and a rule for identifying original source codes. Journal of Science and Technology Mahasarakham University, 2019. (Thai)

บรรณานุกรม

กิตติยา สุทธิประภา (2560), PLAGIARISM: ความสำคัญของการป้องกันการโจรกรรมทางวิชาการ, อินฟอร์เมชั่น; 24(1):90-97

ศราวุธ รุ่งเจริญกิจ, กิติ์สุชาต พสุภา. (2557). การพัฒนาระบบตรวจแบบฝึกหัดการเขียนโปรแกรมแบบอัตโนมัติ. ใน การประชุมวิชาการและเสนอผลงานวิจัยระดับชาติ “การประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 10” (หน้า 258-263). กรุงเทพฯ: คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ศิริพร อ่วมมีเพียร และ สันติพงษ์ ไทยประยูร, “ระบบติดตามการคัดลอกเนื้อหาเว็บอัตโนมัติโดยใช้วิธีการเลือกข้อความสำคัญ,” *วารสารเทคโนโลยีสารสนเทศ*, vol. 12, no. 2, 2016.

สรวัตร ประภานิติเสถียร และ ไกรศักดิ์ เกษร, “การตรวจการโจรกรรมทางวิชาการด้วยใช้เทคนิค N-gram ร่วมกับเทคนิคการตรวจสอบเชิงความหมายสำหรับเอกสารภาษาไทย,” *Journal of Information Science and Technology*, vol. 5 no. 1, 2015.

อักขราวิสุทธิ์ [Online]. Available http://www.akarawisut.com

Ottenstein, K. J. (1976). An Algorithmic Approach to the Detection and Prevention of Plagiarism. ACM SIGCSE Bulletin, 8(4):30–41.

Halstead, M. H. (1977). Elements of Software Science. Elsevier Science Ltd.

Donaldson, J. L., Lancaster, A.-M. & Sposato, P. H. (1981). A Plagiarism Detection System. In Proc. of the 12th SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’81), pages 21–25

S. Grier. (1981). A Tool That Detects Plagiarism in Pascal Programs. In Proc. of the 12th SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’81), pages 15–20.

H. L. Berghel and D. L. Sallach. (1984). Measurements of Program Similarity in Identical Task Environments. ACM SIGPLAN Notices, 19(8):65–76.

J. A. Faidhi and S. K. Robinson. (1987). An Empirical Approach for Detecting Program Similarity and Plagiarism within a University Programming Environment. Computers & Education, 11(1):11–19.

K. L. Verco and M. J. Wise. (1996) Software for Detecting Suspected Plagiarism: Comparing Structure and Attribute-counting Systems. In Proc. of the 1st

Australasian Conference on Computer Science Education (ACSE '96), pages 81–88.

A. Aiken. (1994). Moss: A System for Detecting Software Plagiarism. http://theory.stanford.edu/~aiken/moss/.

M. J. Wise. (1992). Detection of Similarities in Student Programs: YAP'ing may be Preferable to Plague'ing. In Proc. of the 23rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '92), pages 268–271.

M. J. Wise. (1996). YAP3: Improved Detection of Similarities in Computer Program and Other Texts. In Proc. of the 27th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '96), pages 130–134.

D. Gitchell and N. Tran. (1999). Sim: A Utility for Detecting Similarity in Computer Programs. In Proc. of the 30th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '99), pages 266–270.

L. Prechelt, G. Malphol, and M. Philippsen. (2002). Finding Plagiarisms among a Set of Programs with JPlag. Journal of Universal Computer Science, 8(11):1016–1038.

V. Ciesielski, N. Wu, and S. Tahaghoghi. (2008). Evolving Similarity Functions for Code Plagiarism Detection. In Proc. of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO'08), pages 1453–1460.

Poon, J. Y. H., Sugiyama, K., Tan, Y. F. & Kan, M.-Y. (2012). Instructor-centric source code plagiarism detection and plagiarism corpus. In T. Lapidot, J. Gal-Ezer, M. E. Caspersen & O. Hazzan (eds.), ITiCSE (p./pp. 122-127), : ACM. ISBN: 978-1-4503-1246-2

Ji, J.-H., Park, S.-H., Woo, G. & Cho, H.-G. (2008). Generating Pylogenetic Tree of Homogeneous Source Code in a Plagiarism Detection System. *International Journal of Control, Automation, and Systems,* 6(6), 809-817.

Hage, J., Rademaker, P., & Vugt, N.V. (2011). Plagiarism Detection for Java: a Tool Comparison. CSERC Conference, 33-46.

Naik, R.R., Landge, M.B., & Mahender, C.N. (2015). A Review on Plagiarism Detection Tools. International Jornal of Computer Applications, 125(11), 16-22.

Martin, V.T., Fonte, D., Henriques, P.R., & Cruz, D.D. (2014). Plagiarism Detection: A Tool Survey and Comparison. The 3rd Symposium on Languages, Applications and Technologies, 143-158.

MAC Jiffriya, MAC A. Jahan and R. G. Ragel, " Plagiarism Detection on Electronic Text based Assignments using Vector Space Model," *Proceedings of 7th International Conference on Information and Automation for Sustainability*, 2014.

S. K. Shivaji and Prabhudeva S, " Plagiarism Detection by using Karp-Rabin and String Matching Algorithm Together," *International Journal of Computer Applications*, vol. 116, no. 23, 2015.

S. Wang, H. Qi, L. Kong and C. Du, " Combination of VSM and Jaccard Coefficient for External Plagiarism Detection," *Proceedings of the 2013 International Conference on Machine Learning and Cyberetics*, Tianjin, July 14-17, 2013.

S. Niwattanakul, J. Singthongchai, E. Naenudorn and S. Wanapu, " Using of Jaccard Coefficient for Keywords Similarity," *Proceedings of the International Multi* Conference of Engineers and Computer Scientists, 2013.

Anti-kobpae [Online]. Available https://www.anti-kobpae.in.th

CopyCatch [Online]. Available https://www.anti-kobpae.in.th

text.work [Online]. Available http://text.work

C. D. Manning, P. Raghavan and H. Schutze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

Agrawal M, Sharma, DK. A state of art on source code plagiarism detection. Proceedings of 2nd International Conference on Next Generation Computing Technologies; 2016.

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.

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

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.

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.

Qinqin, L, Chunhai, Z. Research on Algorithm of Program Code Similarity Detection. Proceedings of International Conference on Computer Systems, Electronics and Control; 2017.

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):348-361

MOSS: A System for Detecting Software Similarity [online] . Available from: https://theory.stanford.edu/~aiken/moss/ Accessed Dec 2018.

Sedgewick, R. Algorithms. Massachusetts: Addison-Wesley; 2011. P. 604-636

Pemmaraju, S, Skiena, S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. New York: Cambridge University Press, 2003

Mitchell, M. An Introduction to Genetic Algorithms. Massachusetts: MIT Press, 1996

Goldberg, DE. Genetic Algorithms in Search, Optimization and Machine Learning. Massachusetts: Addison-Wesley, 1989

Ter-Sarkisov, A, Marsland, S. Convergence Properties of ($\mu + \lambda$) Evolutionary Algorithms. Proceedings of the 25th AAAI Conference on Artificial Intelligence; 2011

Source Code Plagiarism Test Sets [online] . Available from: https://github.com/nordicway/SourceCode-Plagiarism-TestSets Dec 2018.

ภาคผนวก ก

การตีพิมพ์ผลงานวิจัย

National Conference on Computing and Information Technology

July 5-6 2018

การเปรียบเทียบประสิทธิภาพของตัวชี้วัดความคล้าย ในการตรวจสอบการคัดลอกงานของนักศึกษา

Effecttiveness of Similarity Matrics for Detecting Students' Assignment Plagiarism

ธีรพงษ์ กุลรัตนาวิจิตรา (Teerapong Kulrattanavijitra), ณัฐนนท์ ลีลาศระกูล (Nutthanon Leelathakul)
และ สุนิสา ริมเจริญ (Sunisa Rimcharoen)
คณะวิทยาการสารสนเทศ มหาวิทยาลัยบูรพา
nutthanon@buu.ac.th, rsunisa@buu.ac.th

บทคัดย่อ

งานวิจัยนี้นำเสนอการเปรียบเทียบประสิทธิภาพของตัวชี้วัดความคล้าย (Similarity metrics) เมื่อถูกนำมาใช้เป็นตัววัดความคล้ายคลึงกันของเอกสารภาษาไทย ซึ่งตัวชี้วัดที่ถูกนำศึกษามีดังนี้ Jaccard Index, Sorensen Index, Tanimoto coefficient, TF-IDF Weight และ Jaro-Winkler distance งานวิจัยนี้จะนำประโยคภายในเอกสารมาตัดแยกเป็นคำ เพื่อเก็บแต่ละคำลงใน hash table (โดยจะเก็บคำ และ ความถี่ของคำทั้งหมดที่แยกได้จากเอกสาร) และ vector (เก็บประโยคซึ่งคงลำดับของคำ โดยไม่เก็บความถี่ของแต่ละคำ) จากนั้นจะเปรียบเทียบเอกสารเพื่อหาส่วนที่เหมือนกันและนำส่วนที่เหมือนกันไปค้นหาข้อมูลจากเว็บไซต์ เพื่อทดสอบว่าเอกสารนั้นคัดลอกมาจากเว็บไซต์เดียวกันหรือไม่ ผลการทดลองสามารถบอกได้ว่าเอกสารมีความคล้ายกัน โดยจากการทดสอบกลุ่มเอกสาร 40 ไฟล์ เมื่อกำหนดระดับความคล้ายคลึงที่มากกว่า 89% ขึ้นไปให้ถือว่าไฟล์นั้นคัดลอกกันมา โปรแกรมสามารถตรวจพบไฟล์ที่ข้อมูลมีความคล้ายคลึงกันถึง 443 ครั้ง จากทั้งหมด 780 ครั้ง และเมื่อทดสอบโดยการเปรียบเทียบหาส่วนที่คล้ายกันและนำส่วนที่คล้ายกันไปค้นหากับหน้าเว็บไซต์ โปรแกรมสามารถบอกได้ว่าเว็บไซต์ไหนมีข้อมูลที่เหมือนกับเอกสารที่ตรวจสอบ

คำสำคัญ: ตัวชี้วัดความคล้าย คัดลอกวรรณกรรม

Abstract

This paper studies efficiency of various similarity metrics: Jaccard Index, Sorensen Index, Tanimoto coefficient, TF-IDF Weight and Jaro-Winkler distance. The metrics are used to measure the similarity of each of pair of documents. The proposed method extracts words from files in an achieve, and stores the words together with their statistics in either a hash table or a vector. The statistics are later used to calculate the similarity among documents. After the comparisons, if any pairs of document are found similar, the algorithm, then, checks whether the similar texts are copied from the same webpage. The performance of document similarity detection is satisfactory. From the experimental result, the program can indicate which documents are similar. The experiments use a group of 40 documents. A pair of documents is defined similar when their similar between them is greater than 89%. The program is able to detect 443 of 780 pairs. In addition, the algorithm accurately checks further whether the similar texts were copied from the same website.

Keyword: similarity metrics, plagiarism

1. บทนำ

ปัญหาการลอกเลียนวรรณกรรม (Plagiarism) เป็นปัญหาต่อการศึกษาและการวิจัยที่ฝังรากลึกเป็นระยะเวลายาวนาน ทำให้เป็นปัญหาสำคัญระดับชาติ การขโมยความคิดของผู้อื่นมาเป็นของตนเองเป็นปัญหาร้ายแรงที่ส่งผลต่อความเจริญก้าวหน้า

และชื่อเสียงของประเทศในระยะยาว โดยเฉพาะอย่างยิ่งถ้าปัญหานี้เกิดขึ้นกับกำลังสำคัญของชาติในอนาคต นั่นคือนักเรียนนักศึกษา ดังนั้น ปัญหานี้จึงเป็นเรื่องเร่งด่วนที่ต้องมีมาตรการมาแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสมซึ่งอาจก่อให้เกิดปัญหาใหญ่ที่ร้ายแรงตามมาได้ การแก้ปัญหาที่จุดเริ่มต้นในวัยเยาว์ จะเป็นการปลูกฝังทัศนคติที่ถูกต้องให้กับเยาวชนซึ่งจะเติบโตไปเป็นกำลังสำคัญของประเทศและจะนำไปสู่การพัฒนาประเทศอย่างยั่งยืน

ปัญหาการคัดลอกรายงาน การบ้าน หรืองานต่าง ๆ นอกจากจะเป็นการกระทำที่ผิดแล้วยังเป็นการทำให้ผู้ที่คัดลอกนั้นเสียผลประโยชน์ในความรู้ที่ตัวเองควรจะได้ด้วย อย่างเช่น การคัดลอกการบ้านหรือรายงาน ซึ่งเป็นงานที่ควรจะทำด้วยตนเองเพื่อที่จะได้เข้าใจในรายงานที่ทำ หรือ เข้าใจในบทเรียนมากขึ้น ซึ่งการคัดลอกนั้นก็อาจจะมีวิธีการกระทำได้หลาย ๆ วิธี เช่น คัดลอกจากงานของคนอื่น หรือคัดลอกมาจากเว็บไซต์โดยตรง

ด้วยเหตุนี้เพื่อให้การตรวจสอบข้อมูลระหว่างไฟล์เอกสารด้วยกัน หรือ ระหว่างไฟล์เอกสารกับข้อมูลที่มาจากเว็บไซต์นั้นสามารถทำได้โดยอัตโนมัติ งานวิจัยนี้จึงได้ศึกษา ทดลอง และเปรียบเทียบเทคนิคการตรวจสอบความคล้ายกันของเอกสารเพื่อหาตัวชี้วัดความคล้าย ที่ให้ความแม่นยำในการที่จะนำไปใช้ตรวจสอบ ผลลัพธ์ที่ได้จากงานวิจัยนี้จะเป็นประโยชน์อย่างยิ่งที่จะเป็นเหมือนการป้องกันขั้นหนึ่งเพื่อที่จะทำให้ปัญหาการคัดลอกกันนั้นน้อยลงไป

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 การคัดลอกวรรณกรรม (Plagiarism)

ที่ผ่านมามีงานวิจัยจำนวนมากที่พยายามหาขั้นตอนวิธีในการตรวจจับการคัดลอกวรรณกรรม โดยใช้เทคนิคต่าง ๆ เช่น Jiffriya et al. [1] ใช้โมเดลแบบเวกเตอร์สเปซในการตรวจจับการคัดลอกเอกสาร ผลการทดลองพบว่าการใช้ไตรแกรม (Tri-gram) ควบคู่กับตัววัดความคล้ายแบบโคไซน์ (Cosine Similarity) ให้ผลลัพธ์ดีที่สุด Shivaji et al. [2] ใช้เทคนิคการเปรียบเทียบสายอักขระแบบ Karp-Rabin ในการตรวจการคัดลอกงานเขียนของนักศึกษา ด้วยการเปรียบเทียบกับคลังบทความที่มีอยู่ ซึ่งผลการทดลองพบว่ามีความแม่นยำ 85% Wang et al. [3] ใช้โมเดลแบบเวกเตอร์สเปซร่วมกับ Jaccard Coefficient เนื่องจากโมเดลแบบเวกเตอร์สเปซให้ประสิทธิภาพด้านความครอบคลุมสูง (Recall) ในขณะที่ Jaccard Coefficient ช่วยในแง่ของความแม่นยำ (Precision)

ในส่วนของการตรวจจับการคัดลอกวรรณกรรมที่เป็นงานเขียนภาษาไทย สรวัศ และ ไกรศักดิ์ [4] ได้ใช้เทคนิค N-gram ร่วมกับหลักไวยากรณ์ และการตรวจสอบเชิงความหมาย ในการตรวจจับการคัดลอกวรรณกรรมเชิงวิชาการ ด้วยการปรับปรุงโครงสร้างของประโยคเพื่อให้การเปรียบเทียบความคล้ายมีความแม่นยำมากขึ้น ศิริพร และ สันติพงษ์ [5] ได้นำเสนอวิธีการตรวจจับการคัดลอกเนื้อหาจากเว็บเพจหนึ่งไปยังอีกเว็บเพจหนึ่ง โดยการสกัดเนื้อหาและประมวลผลข้อความจากเว็บเพจ เลือกคำสำคัญมาสืบค้นเพื่อเปรียบเทียบกับเว็บเพจอื่น ผลการทดลองพบว่าได้ค่า F-measure เท่ากับ 0.83 Niwattanakul et al. [6] ใช้ตัววัดความคล้าย Jaccard Coefficient ในการหาความคล้ายกันของคำสำคัญ (Keyword) จากพจนานุกรมคำคล้าย (Thesaurus) ด้านการเกษตรในหัวข้อเกี่ยวกับฟาร์ม ผลการวิจัยพบว่า Jaccard Coefficient ให้ผลไม่ค่อยเป็นที่น่าพอใจ นอกจากนี้ยังมีงานวิจัยที่นำมาให้บริการตรวจจับการคัดลอกผ่านทางเว็บไซต์ เช่น อักขราวิสุทธิ์ [7], แอนตี้ ก๊อปแปะ [8], CopyCatch [9] และ text.work [10]

จากงานวิจัยในอดีตที่ผ่านมา พบว่ามีงานวิจัยจำนวนมากที่นำเสนอเสนอเทคนิคต่าง ๆ ในการเปรียบเทียบความคล้ายกันของเอกสาร ซึ่งแต่ละเทคนิคอาจมีความเหมาะสมกับประเภทของงานที่แตกต่างกันไป ในบทความนี้จึงนำเสนอการเปรียบเทียบตัววัดความคล้าย เมื่อนำมาใช้เปรียบเทียบความคล้ายกันของงานเขียนที่เป็นการบ้านภาษาไทยของนักศึกษา อีกทั้งยังเสนอวิธีการเปรียบเทียบงานที่คล้ายกันของนักศึกษากับข้อมูลในเว็บไซต์ เนื่องจากในปัจจุบัน นักศึกษาหลายคนทำการบ้านหรือรายงานโดยใช้วิธีการหาข้อมูลและคัดลอกเนื้อหาจากในเว็บไซต์มาส่งอาจารย์

2.2 การวัดความคล้าย (Similarity Measure)

งานวิจัยนี้ได้ศึกษาเปรียบเทียบตัวชี้วัดความคล้ายคล้าย 5 แบบ ดังนี้

2.2.1 Jaccard Index

Jaccard Index หรือ Jaccard Coefficient เป็นตัวชี้วัดความคล้ายที่ใช้วัดความเหมือนหรือความหลากหลายของสมาชิกในเซต โดยคำนวณได้จาก สมการที่ (1)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

การนำ Jaccard Index มาประยุกต์ใช้ในงานวิจัยนี้ จะกำหนดให้เซต A แทนชุดของคำที่แยกได้จากข้อความที่เป็นอินพุต และ เซต B แทนชุดของคำที่แยกได้จากข้อมูลจากการค้นหาจากเสิร์ชเอนจินโดยที่ไม่สนใจความถี่ของคำที่ซ้ำกันในเอกสาร

2.3.2 Sorensen Index

Sorensen Index หรือ Sorensen Coefficient เป็นตัวชี้วัดความคล้าย ที่ใช้วัดความเหมือนหรือต่างกันของตัวอย่างสายพันธุ์ของสิ่งมีชีวิต โดยคำนวณได้จาก สมการที่ (2) โดยที่ A และ B คือจำนวนสมาชิกของสายพันธุ์ และ C คือจำนวนที่เหมือนกันของสมาชิกในสายพันธุ์ A และ B

$$QS = \frac{2C}{A+B} \quad (2)$$

การนำ Sorensen Index มาประยุกต์ใช้ในงานวิจัยนี้ก็จะคล้ายกับวิธี Jaccard Index โดยสมาชิกเซต A จะแทนด้วยคำที่แยกได้จากอินพุต และ สมาชิกเซต B แทนด้วยคำที่แยกได้จากข้อมูลจากการค้นหาจากเสิร์ชเอนจินโดยที่ไม่สนใจความถี่ของคำที่ซ้ำกันในเอกสาร

2.3.3 Tanimoto Coefficient

Tanimoto Coefficient เป็นตัวชี้วัดความคล้ายที่ประยุกต์มาจาก Cosine Similarity และ Jaccard Index โดย Tanimoto Coefficient จะคำนวณจาก สมการที่ (3) และ (4)

$$T(A,B) = \frac{A \cdot B}{\|A\|^2 + \|B\|^2 - A \cdot B} \quad (3)$$

$$\|A\| = \sqrt{A_1^2 + \cdots + A_n^2} \quad (4)$$

งานวิจัยนี้นำ Tanimoto Coefficient มาประยุกต์ใช้ในการหาความคล้ายกันของเอกสาร โดยแทนค่าในเวกเตอร์ด้วยความถี่ของคำที่ปรากฏในเอกสาร คือจำนวนครั้งของคำที่ปรากฏในเอกสารที่นำมาเปรียบเทียบทั้งเอกสารอินพุตและข้อมูลที่ค้นหาได้จากเสิร์ชเอนจิน

2.3.4 TF-IDF Weight

TF-IDF ย่อมาจาก Term Frequency - Inverse Document Frequency เป็นการวัดค่าน้ำหนักเพื่อประเมินค่าความสำคัญของคำนั้น ๆ ในกลุ่มของเอกสารซึ่ง TF-IDF มักจะใช้ในงานสืบค้นข้อมูล [11] โดย TF คือ จำนวนของการปรากฏของคำในเอกสารที่สนใจ ส่วน IDF คือ ค่าส่วนกลับของจำนวนเอกสารที่มีคำนั้น ๆ ปรากฏ คำนวณได้จากสมการที่ (5) โดย N แทนจำนวนเอกสารทั้งหมด ซึ่งในงานวิจัยนี้กำหนดให้เป็น 2 เนื่องจากเราเปรียบเทียบโดยใช้เอกสารสองข้อมูลเท่านั้น และ df คือจำนวนเอกสารที่พบคำที่ปรากฏทั้งหมด การคำนวณน้ำหนัก ของ TF-IDF จะใช้สมการที่ (6) ในการคำนวณโดยจะคำนวณจากค่า tf คูณด้วย idf

$$\text{idf} = \log \frac{\text{N}}{\text{df}} \quad (5)$$

$$TF - IDF = \text{tf} \times \text{idf} \quad (6)$$

หลังจากที่คำนวณน้ำหนักของคำได้แล้ว จึงนำค่าน้ำหนักของคำที่ได้ของแต่ละคำไปหาความคล้ายโดยใช้ตัวชี้วัดความคล้าย Tanimoto Coefficient มาช่วยในการหาความคล้าย ดังสมการที่ (3)

2.3.5 Jaro-Winkler Distance

Jaro-Winkler Distance เป็นการวัดความคล้ายคลึงกันของสายอักขระ โดยเป็นการพัฒนามาจาก Jaro Distance ซึ่งการคำนวณจะเริ่มจากการหาค่า Jaro Distance โดยใช้สมการที่ (7)

$$jaro\ distance = d_j = \frac{1}{3}\left(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{m}\right) \quad (7)$$

โดยที่ m คือ จำนวนตัวอักษรที่เหมือนกัน

|s1| คือ ความยาวของสายอักขระที่ 1

|s2| คือ ความยาวของสายอักขระที่ 2

t คือ ครึ่งหนึ่งของจำนวนทรานโพสิชั่น

ทรานโพสิชั่น คือ ตัวอักขระที่เหมือนกันแต่ไม่ได้อยู่ในตำแหน่งเดียวกัน โดยระยะห่างของอักขระที่ เหมือนกันแต่ไม่ได้อยู่ในตำแหน่งเดียวกันนั้น จะต้องอยู่ห่างกันไม่เกินกว่าค่าที่คำนวณได้จาก สมการMatching (สมการที่ 8) หลังจากคำนวณค่า Jaro Distance ได้แล้ว จึงคำนวณหาผลลัพธ์ของ Jaro-Winkler Distance ด้วยสมการที่ (9)

$$\text{matching} = \left[\frac{max(|s_1|,|s_2|)}{2}\right] - 1 \quad (8)$$

$$jaro\text{-}winkler\ distance = d_w = d_j + \left(lp(1-d_j)\right) \quad (9)$$

3. วิธีการที่นำเสนอ

บทความนี้นำเสนอการตรวจหาเอกสารที่คล้ายกัน โดยเปรียบเทียบกับข้อความที่สืบค้นได้จากเสิร์ชเอนจิน ผู้วิจัยได้ทดสอบเปรียบเทียบตัวชี้วัดความคล้าย 5 แบบ เพื่อทดสอบหาตัวชี้วัดความคล้ายที่มีประสิทธิภาพดีที่สุดที่จะนำไปใช้ตรวจสอบข้อมูลของเอกสารหรือค้นหาเอกสารที่คล้ายกันได้ ซึ่งขั้นตอนการดำเนินการวิจัยมีดังนี้

3.1 ขั้นตอนการเตรียมข้อมูล

ข้อมูลเพื่อที่จะนำไปทดสอบหาความคล้ายกันจะมีสองชุด คือ ข้อมูลที่มาจากไฟล์การบ้านของนักศึกษา และ ข้อมูลที่ได้จากการค้นหาจากเสิร์ชเอนจิน (Google) ผู้วิจัยจะแยกคำจากข้อมูลในเอกสาร โดยตัดเครื่องหมายวรรคตอน อักขระพิเศษ และ stop word ออก จากนั้นจะจัดเก็บคำที่แยกได้ใน ตารางแฮช หรือเวกเตอร์ โดยการเลือกวิธีการเก็บข้อมูลนั้นก็จะขึ้นอยู่กับจุดประสงค์ในการนำข้อมูลไปใช้ในวิธีที่แตกต่างกัน เราจะจัดเก็บข้อมูลของคำที่แบ่งได้เข้าสู่ตารางแฮช ก็ต่อเมื่อตัวชี้วัดความคล้ายที่จะนำมาคำนวณหาความคล้ายนั้นให้ความสำคัญกับคำ หรือ ความถี่ของคำที่แบ่งได้จากเอกสาร ส่วนการจัดเก็บข้อมูลลงในเวกเตอร์ จะนำมาใช้ ก็ต่อเมื่อตัวชี้วัดความคล้ายของเรานั้น ไม่ได้สนใจความถี่ของคำแต่ละคำโดยเฉพาะ แต่จะสนใจที่ลำดับตำแหน่งของคำแทน

การเตรียมข้อมูลจาก Search Engine (Google) นั้นเราจะทำการค้นหาโดยใช้ข้อมูลที่ได้จากไฟล์เอกสาร เช่น สายอักขระที่นำมาจากในเอกสารหรือคำในเอกสารที่มีความถี่ของคำมากที่สุด 10 อันดับ เป็นต้น เพื่อให้ได้ยูอาร์แอลที่คาดว่าน่าจะตรงกับข้อมูลในเอกสาร โดยวิธีการเลือกประโยคหรือคำเพื่อนำไปค้นหาโดยใช้ Google นั้น ในงานวิจัยนี้ได้เลือกวิธีคัดเลือกคำที่จะใช้ อยู่สามวิธี

วิธีแรกคือ ค้นหาโดยใช้คำที่แยกได้มาเรียงต่อกันเป็นประโยค โดยรูปแบบข้อมูลที่เรานำมาใช้คือข้อมูลที่มีรูปแบบการเก็บข้อมูลแบบเวกเตอร์ เพราะการเก็บข้อมูลในเวกเตอร์ จะให้ความสำคัญกับตำแหน่งของคำ

วิธีที่สองคือ การใช้คำมาต่อกันเป็นสายอักขระแบบวิธีแรก แต่เพิ่มวิธีการสุ่มตำแหน่งของสายอักขระ โดยเราจะทำการสุ่มตำแหน่งเริ่มต้นของส่วนที่เราจะนำมาเป็นสายอักขระ จากนั้นทำการแบ่งเอกสารออกเป็น 3 ส่วน คือ ส่วนต้น ส่วนกลาง และส่วนท้ายของเอกสาร ซึ่งการใช้วิธีนี้ จะทำให้ได้ข้อมูลที่จะนำไปค้นหานั้นมีความหลากหลายมากขึ้น

วิธีที่สามคือ การใช้คำที่มีความถี่มากที่สุดในเอกสารจำนวน 16 คำ มาต่อกันโดยใช้เพิ่มเครื่องหมายบวก "+" ระหว่างคำ (การค้นหาของ Google นั้น การเพิ่มเครื่องหมายบวก "+" ระหว่างคำ คือการค้นหาตามคำที่มีนั้นทุกประการ)

3.2 การเปรียบเทียบความคล้ายคลึงกันของข้อมูล

เมื่อได้ข้อมูลจากวิธีการขั้นต้นแล้ว ต่อไปก็จะเข้าสู่ขั้นตอนการเปรียบเทียบข้อมูลที่ได้จากไฟล์ข้อมูลอินพุตและข้อมูลที่ได้จาก Google โดยในงานวิจัยนี้ได้ทดลองใช้ตัวชี้วัดความคล้าย 5 แบบ ดังนี้

3.2.1 Jaccard Index

การเปรียบเทียบข้อมูลโดยวิธี Jaccard Index นั้นเป็นการเปรียบเทียบโดยสนใจคำที่แยกได้จากข้อมูล กับจำนวนคำทั้งหมดของข้อมูล โดยขั้นตอนแรกเราจะนำข้อมูลจากตารางแฮช มาเก็บไว้ในแถวลำดับ เพื่อที่จะนำไปเปรียบเทียบ เมื่อได้แถวลำดับมาแล้วจึงนำแถวลำดับทั้งสองมาหาว่ามีคำใดบ้างที่เหมือนกันซึ่งก็คือการอินเตอร์เซกชันกันของข้อมูลทั้งสองเซต

3.2.2 Sorensen Index

การเปรียบเทียบข้อมูลโดยวิธี Sorensen Index นั้นจะให้ความสำคัญกับคำที่มีในเอกสาร แต่ไม่สนใจความถี่ของคำ และสนใจกับจำนวนทั้งหมดของคำของทั้งสองเอกสาร โดยเราจะใช้ข้อมูลที่เก็บอยู่ในตารางแฮชมาใช้ เนื่องจากว่า เราสนใจแต่คำในเอกสารนั้น ๆ แต่ไม่ได้สนใจกับลำดับตำแหน่งของคำ จากนั้นเรานำข้อมูลทั้งหมดมาเก็บไว้ในแถวลำดับ เพื่อที่จะนำไปเปรียบเทียบต่อไป

3.2.3 Tanimoto coefficient

การเปรียบเทียบข้อมูลโดยตัวชี้วัดความคล้าย Tanimoto Coefficient จะเปรียบเทียบโดยให้ความสำคัญกับจำนวนความถี่ของคำแต่ละคำในเอกสาร โดยจะทำการเปรียบเทียบโดยนำความถี่ของคำที่เหมือนกันมาคำนวณ โดยจะนำข้อมูลจากตารางแฮชมาใช้งาน และจะเปรียบเทียบโดยนำข้อมูลที่ค้นหาได้จาก Google เป็นตัวกำหนดหลัก ถ้าคำที่อยู่ในข้อมูลอินพุตนั้น ไม่มีอยู่ในข้อมูลที่ได้จาก Google จะกำหนดให้ค่าความถี่นั้นเป็น 0

3.2.4 TF-IDF Weight

การเปรียบเทียบโดยวิธี TF-IDF Weight เป็นการเปรียบเทียบโดยให้ความสำคัญกับความถี่ของคำในเอกสาร เช่นเดียวกับ วิธี Tanimoto Coefficient แต่ TF-IDF จะเป็นวิธีปรับค่าน้ำหนักของคำในเอกสารวิธีการคำนวณโดยเราจะนำข้อมูลจากตารางแฮชมาใช้ โดยค่าน้ำหนักนั้นจะประกอบด้วยสองส่วน ส่วนแรก คือ TF (Term Frequency) และ IDF (Inverse Document Frequency)

3.2.5 Jaro-Winkler Distance

การเปรียบเทียบข้อมูลโดยวิธี Jaro-Winkler นั้นเราจะเปรียบเทียบโดยให้ความสำคัญกับลำดับของคำในเอกสาร โดยไม่สนใจความถี่ของคำ ดังนั้นเราจึงใช้เวกเตอร์ในการจัดเก็บข้อมูลเพื่อนำข้อมูลมาเปรียบเทียบ ในที่นี้จะใช้เทคนิค k-gram ในการเลือกความยาวของคำเพื่อที่จะนำมาเปรียบเทียบในตัวอย่างนี้จะกำหนดให้แบ่งเป็นข้อมูลละ 3 คำ (Tri-gram) หลังจากได้ผลลัพธ์จากการคำนวณมาแล้ว เราก็จะทำการตรวจสอบว่า ค่าที่คำนวณได้มีค่ามากกว่าค่าขีดแบ่ง (Threshold) หรือไม่ โดยในงานวิจัยนี้ได้ทดลองใช้ค่าขีดแบ่งหลาย ๆ ระดับ

4. ผลการทดลอง

การทดสอบตัวชี้วัดความคล้ายในงานวิจัยนี้ จะใช้ไฟล์เอกสารจากแบบฝึกหัด หรือการบ้านที่นักศึกษาใช้ส่งอาจารย์จริง โดยจะทำการทดสอบสองแบบ แบบแรกคือทดสอบกลุ่มเอกสารเพื่อหาว่าไฟล์ใดที่คัดลอกกันมา โดยการทดสอบนี้จะไม่นำส่วนข้อมูลที่เหมือนกันไปค้นหาข้อมูลใน Google ต่อ แบบที่สองคือเปรียบเทียบไฟล์เอกสารสองไฟล์ โดยเริ่มจากการทดสอบหาความคล้ายกันของไฟล์ทั้งสอง ถ้าไฟล์ทั้งสองนั้นมีส่วนของข้อมูลที่เหมือนกัน เราอาจสรุปว่านักศึกษาคัดลอกเนื้อหาของกันและกัน หรือ นักศึกษาคัดลอกเนื้อหามาจากหน้าเว็บเดียวกัน แล้วค้นหาข้อมูลใน Google ต่อ

4.1 ผลการทดสอบความคล้ายกันของกลุ่มเอกสาร

ในการทดสอบนี้เราจะนำตัวชี้วัดความคล้ายแต่ละวิธีมาหาค่าความคล้ายคลึงของเอกสาร ซึ่งจะทำการเปรียบเทียบเอกสารทั้งหมด 3 กลุ่มตัวอย่างด้วยกัน โดยจะทำการเปรียบเทียบทุกไฟล์ ตัวอย่างกลุ่มแรกมีไฟล์ข้อมูลทั้งหมด 41 ไฟล์ ตัวอย่างกลุ่มที่สองมีไฟล์ข้อมูลทั้งหมด 45 ไฟล์ และตัวอย่างกลุ่มที่สามมีไฟล์ข้อมูลทั้งหมด 40 ไฟล์ เราจะทำการทดสอบด้วยตัวชี้วัดความคล้ายวิธีต่าง ๆ โดยกำหนดระดับค่าความคล้ายคลึงที่จะยอมรับว่าไฟล์นั้นเหมือนกันหรือค่าขีดแบ่ง (Threshold) คือ ตั้งแต่ 89.99 เปอร์เซ็นต์ขึ้นไป

จากการตรวจสอบไฟล์เอกสารทั้งสามกลุ่ม โดยในแต่ละกลุ่มจะนำไฟล์ทั้งหมดมาเปรียบเทียบกันทั้งหมดทุกไฟล์ ผลทดสอบแต่ละตัวชี้วัดความคล้าย ดังตารางที่ 1 พบว่าวิธีที่ให้จำนวนเอกสารที่คล้ายกันมากที่สุดได้แก่ Sorensen, Tanimoto และ Jaccard วิธีที่ให้จำนวนเอกสารที่มีความคล้ายกันน้อยสุด คือ Jaro-Winkler ซึ่งเมื่อตรวจสอบข้อมูลด้วยตาเปล่าแล้วพบว่าวิธีทดสอบที่ใช้ตัวชี้วัดความคล้าย Jaccard, Sorensen และ Tanimoto ไฟล์ที่มีค่าความคล้ายคลึงกันตั้งแต่ 96 ถึง 100 เปอร์เซ็นต์ นั้นไฟล์ข้อมูลจะมีความคล้ายกันมาก แต่ไฟล์ข้อมูลที่มีค่าความคล้ายคลึงตั้งแต่ 96 เปอร์เซ็นต์ลงไป เมื่อตรวจสอบข้อมูลในไฟล์ดูแล้ว บางไฟล์ไม่ได้มีข้อมูลที่คล้ายกันจนถือว่าคัดลอกกันมา ส่วนผลที่ได้จากวิธี Jaro-Winkler ไฟล์ข้อมูลที่มีค่าความคล้ายคลึงกันตั้งแต่ 92 เปอร์เซ็นต์ขึ้นไป เมื่อตรวจสอบข้อมูลในไฟล์แล้วข้อมูลในไฟล์มีความเหมือนกันมาก ส่วนคะแนนที่ต่ำกว่า 92 เปอร์เซ็นต์ลงไปนั้น ข้อมูลในไฟล์ก็จะมีบางส่วนเท่านั้นที่คล้ายกัน

ตารางที่ 1: การเปรียบเทียบตัวชี้วัดความคล้าย

ตัววัดความคล้าย	ชุดข้อมูล	90 - 91.99%	92 - 93.99%	94 - 95.99%	96 - 97.99%	98 - 100%
Jaccard	1	22	7	4	2	9
	2	39	15	7	17	70
	3	12	40	40	41	33
Sorensen	1	22	17	34	11	11
	2	73	108	136	22	87
	3	14	60	34	11	11
Tanimoto	1	19	15	18	20	16
	2	7	15	65	108	336
	3	1	44	18	21	34
TF-IDF	1	0	0	0	0	0
	2	0	0	0	0	0
	3	0	0	0	0	0
Jaro-Winkler	1	0	1	0	0	5
	2	35	31	4	1	29
	3	10	1	1	1	16

ข้อสังเกต ไฟล์ข้อมูลบางไฟล์มีการสลับลำดับของคำตอบ แต่ตัวชี้วัดความคล้าย Jaccard Sorensen และ Tanimoto ก็ยังสามารถให้ค่าคะแนนความคล้ายที่สูง แต่ในตัวชี้วัดความคล้าย Jaro-Winkler กลับให้คะแนนความคล้ายน้อยกว่า

4.2 ผลการทดสอบการเปรียบเทียบเอกสารโดยนำข้อมูลที่เหมือนกันไปค้นหาในสิริซแอนจิน

วิธีการทดลองนี้เริ่มจากการทดสอบหาความคล้ายกันของไฟล์เอกสารทั้งสอง ถ้าไฟล์ทั้งสองนั้นมีส่วนของข้อมูลที่เหมือนกัน เราอาจสรุปว่านักศึกษาคัดลอกเนื้อหาของกันและกัน หรือ นักศึกษาคัดลอกเนื้อหามาจากหน้าเว็บเดียวกัน ซึ่งเราจะทำการทดสอบต่อไป โดยจะทำการนำข้อมูลที่เหมือนกันนั้นไปค้นหาใน Google เพื่อที่จะทำการทดสอบว่าข้อมูลส่วนนี้นั้นคัดลอกมาจากเว็บอะไร จากนั้นเมื่อได้ยูอาร์แอล จากการค้นหาแล้ว จึงนำไฟล์ทั้งสองไปเปรียบเทียบกับยูอาร์แอลที่ค้นหาได้ เพื่อหาว่ามีความเอกสารมีความคล้ายคลึงกับข้อมูลยูอาร์แอลหรือไม่

เมื่อเราได้ส่วนที่เหมือนกันจากไฟล์เอกสารทั้งสองแล้ว จากนั้นจึงนำข้อมูลในส่วนที่เหมือนกันไปค้นหาข้อมูลจาก Google ซึ่งจะได้ยูอาร์แอลผลลัพธ์จากข้อมูลที่เราใช้ค้นหา จากนั้นเราจะทำการเปรียบเทียบไฟล์เอกสารทั้งสองกับข้อมูลที่ค้นหาได้ โดยผลการเปรียบเทียบไฟล์ และ การทดสอบหาความคล้ายกับข้อมูลจากยูอาร์แอลต่าง ๆ เราสามารถสรุปผลการทดสอบได้ดังที่จะวิเคราะห์ในหัวข้อถัดไป

4.3 วิเคราะห์คุณลักษณะของตัวชี้วัดความคล้ายวิธีต่าง ๆ

4.3.1 Jaccard Index

การทดสอบโดยใช้ตัวชี้วัดความคล้าย Jaccard Index ในการทดสอบหาความคล้ายคลึงกันนั้น Jaccard Index จะไม่สนใจในลำดับตำแหน่งของคำ และ ไม่สนใจความถี่ของคำในเอกสาร ในการทดสอบนั้นถ้าข้อมูลที่นำมาทดสอบมีขนาดของข้อมูลที่ใกล้เคียงกัน ตัวชี้วัดนี้สามารถบอกค่าความคล้ายคลึงกันได้ดี ถึงแม้ว่าข้อมูลที่นำมาเปรียบเทียบกันนั้นจะมีการสลับลำดับตำแหน่งของคำตอบ

แต่การที่ตัวชี้วัดนี้ไม่ได้สนใจกับลำดับตำแหน่งของคำหรือความถี่ของคำ ข้อมูลที่นำมาทดสอบบางอย่างจึงไม่สามารถสรุปได้ว่า ข้อมูลในเอกสารที่นำมาทดสอบนั้นมีความสัมพันธ์กันหรือไม่

4.3.2 Sorensen Index

การทดสอบโดยตัวชี้วัดความคล้าย Sorensen Index นี้จะมีความคล้ายกับ Jaccard Index เช่น ถ้าขนาดของข้อมูลใกล้เคียงกันก็สามารถให้ผลลัพธ์ในระดับที่ดี แต่ถ้าข้อมูลที่นำมาทดสอบนั้นมีขนาดแตกต่างกันมาก ค่าความคล้ายที่ได้จะได้เปอร์เซ็นต์ที่ไม่สูงมากนัก เช่นเดียวกับ Jaccard Index ตัวชี้วัดความคล้าย Sorensen นั้นไม่ได้ให้ความสนใจกับลำดับตำแหน่งหรือความถี่ของคำ จึงทำให้ข้อมูลบางอย่างไม่สามารถสรุปได้ว่า ข้อมูลในเอกสารที่นำมาทดสอบนั้นมีความสัมพันธ์กันหรือไม่

4.3.3 Tanimoto coefficient

การทดสอบโดยตัวชี้วัดความคล้าย Tanimoto coefficient นี้ตัวชี้วัดความคล้ายจะให้ความสำคัญกับจำนวนความถี่ของคำที่นำมาคำนวณ ซึ่งถ้าเอกสารทั้งสองมีความถี่ที่ใกล้เคียงกัน ตัวชี้วัดความคล้ายคลึงนี้ จะได้ค่าความคล้ายคลึงที่ดี และถ้าจำนวนของคำทั้งหมดของทั้งสองเอกสารไม่ต่างกันมากค่าความคล้ายคลึงจะมีค่ามากขึ้น

4.4.4 TF-IDF Weight

การทดสอบโดยใช้วิธี TF-IDF Weight นั้น เมื่อนำมาใช้กับการเปรียบเทียบเอกสารแค่สองเอกสารนั้นไม่สามารถให้ค่าความคล้ายคลึงออกมาได้เนื่องจากตามวิธีคำนวณค่า idf นั้น คำที่ปรากฏในทุก ๆ เอกสารนั้นจะถือว่าคำนั้นมีความสำคัญน้อยหรือไม่มีความสำคัญ

4.5.5 Jaro-Winkler distance

การทดสอบโดยตัวชี้วัดความคล้ายด้วยวิธีนี้นั้น ในการตรวจสอบนั้นจะให้ความสำคัญกับลำดับของคำ ถ้าข้อมูลที่นำมาตรวจสอบไม่ได้มีการดัดแปลง หรือ เป็นข้อมูลที่ไม่ใช่ข้อมูลที่ได้มาจากหลาย ๆ ข้อมูลรวมกัน การตรวจสอบด้วยวิธีนี้นั้น จะให้ค่าความคล้ายที่สูงประสิทธิภาพมาก และเนื่องจากการที่มีการให้ความสำคัญกับลำดับของคำทำให้สามารถบอกได้ว่า ข้อมูลนั้น มีความสัมพันธ์กัน

5. สรุป

การทดลองในงานวิจัยนี้เป็นการเปรียบความคล้ายคลึงของเอกสารและข้อมูลที่สืบค้นได้จาก Google โดยวัดผลจากคะแนนที่ได้ในแต่ละตัวชี้วัดความคล้าย เพื่อบ่งบอกว่าเอกสาร

ใดมีการคัดลอกข้อมูลมาโดยตรงหรือไม่ ซึ่งผลที่ได้น่าพึงพอใจ แต่เนื่องจากข้อมูลที่นำมาทดสอบนั้นบางข้อมูลมีขนาดแตกต่างกันมากจึงทำให้ค่าความคล้ายที่ได้ไม่สูงมากนัก เช่น ใน ตัวชี้วัดความคล้าย Jaccard Index, Sorensen Index เป็นต้น และ วิธีการหาความคล้ายโดยใช้ตัวชี้วัดความคล้าย TF-IDF นั้น ไม่สามารถหาค่าความคล้ายคลึงของสองเอกสารได้จึงไม่สามารถให้ผลลัพธ์ได้ แต่จากตัวชี้วัดความคล้าย Jaro-Winkler ให้ผลลัพธ์ที่น่าพอใจมาก เนื่องจากสามารถตรวจสอบข้อมูลจากตัวอย่างได้เป็นเปอร์เซ็นต์ที่สูง โดยสามารถบอกได้ว่าข้อมูลที่นำมาทดสอบนั้นคัดลอกมาจากเว็บไซต์อะไร

6. กิตติกรรมประกาศ

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

เอกสารอ้างอิง

[1] MAC Jiffriya, MAC A. Jahan and R. G. Ragel, "Plagiarism Detection on Electronic Text based Assignments using Vector Space Model," *Proceedings of 7th International Conference on Information and Automation for Sustainability*, 2014.

[2] S. K. Shivaji and Prabhudeva S, "Plagiarism Detection by using Karp-Rabin and String Matching Algorithm Together," *International Journal of Computer Applications*, vol. 116, no. 23, 2015.

[3] S. Wang, H. Qi, L. Kong and C. Du, "Combination of VSM and Jaccard Coefficient for External Plagiarism Detection," *Proceedings of the 2013 International Conference on Machine Learning and Cyberetics*, Tianjin, July 14-17, 2013.

[4] สรวัศ ประภานิศสถียร และ ไกรศักดิ์ เกษร, "การตรวจการโจรกรรมทางวิชาการด้วยใช้เทคนิค N-gram ร่วมกับเทคนิคการตรวจสอบเชิงความหมายสำหรับเอกสารภาษาไทย," *Journal of Information Science and Technology*, vol. 5 no. 1, 2015.

[5] สิริพร อ่วมมีเพียร และ สันติพงษ์ ไทยประยูร, "ระบบคิดตามการคัดลอกเนื้อหาเว็บอัตโนมัติโดยใช้วิธีการเลือกข้อความสำคัญ," *วารสารเทคโนโลยีสารสนเทศ*, vol. 12, no. 2, 2016.

[6] S. Niwattanakul, J. Singthongchai, E. Naenudorn and S. Wanapu, "Using of Jaccard Coefficient for Keywords Similarity," *Proceedings of the International Multi* Conference of Engineers and Computer Scientists, 2013.

[7] อักขราวิสุทธิ์ [Online]. Available https://www.akarawisut.com

[8] Anti-kobpae [Online]. Available https://www.anti-kobpae.in.th

[9] CopyCatch [Online]. Available https://www.anti-kobpae.in.th

[10] text.work [Online]. Available https://text.work

[11] C. D. Manning, P. Raghavan and H. Schutze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

ภาคผนวก ข

การตีพิมพ์ผลงานวิจัย

Journal of Science and Technology Mahasarakham University

2019

นิพนธ์ต้นฉบับ

ต้นไม้วิวัฒนาการและกฎสำหรับการระบุต้นตอของการลอกเลียนซอร์สโค้ด

PHYLOGENETIC TREES AND A RULE FOR IDENTIFYING ORIGINAL SOURCE CODES

ณัฐนนท์ ลีลาตระกูล[1], สุนิสา ริมเจริญ[2]

Nutthanon Leelathakul, Sunisa Rimcharoen

Received:

บทคัดย่อ

การลอกเลียนวรรณกรรม (Plagiarism) ถือเป็นการทำผิดทางจริยธรรมที่ร้ายแรง หากไม่มีการป้องกันหรือตรวจจับที่ดีแล้ว งานสร้างสรรค์ใหม่ ๆ จะเกิดขึ้นน้อย ทำให้เป็นปัญหาต่อการศึกษาและการวิจัยของประเทศ การจัดการเรียนการสอนวิชาการเขียนโปรแกรมก็ประสบปัญหานี้อย่างมาก กล่าวคือ นักเรียนมีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุดคือ การคัดลอกโปรแกรมของเพื่อนแล้วมาส่งอาจารย์ ปัญหานี้จะต้องรีบถูกแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสม ดังนั้น งานวิจัยนี้จึงนำเสนอวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ดเพื่อให้ทราบถึงแหล่งที่มาของการลอกโปรแกรมในกลุ่มนักเรียนนักศึกษา และเมื่ออาจารย์ผู้สอนทราบว่านักเรียนคนใดเป็นต้นฉบับ คนใดที่เป็นคนลอกเพื่อนมาส่ง อาจารย์จะได้วางแนวทางและมาตรการแก้ไข ตักเตือน และแก้ปัญหาได้ตั้งแต่ต้น งานวิจัยนี้ได้นำเสนอขั้นตอนวิธีในการระบุต้นฉบับของซอร์สโค้ดโดยใช้กฎที่สร้างขึ้นจากขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ 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 nation 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 issue must be addressed to preclude this misbehavior before it becomes students' habit, and to promote positive attitude in our juveniles. In this paper, we, therefore, 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 $\mu+\lambda$ evolutionary algorithm, for identifying which source codes are original. First, we run the maximum spanning tree algorithm to generate a phylogenetic tree, representing the

ผู้ช่วยศาสตราจารย์ คณะวิทยาการสารสนเทศ มหาวิทยาลัยบูรพา, อำเภอเมือง จังหวัดชลบุรี 20131

Assistant Professor, Faculty of Informatics, Burapha university, Muang District, Chonburi 20131, Thailand.

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

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

บทนำ

ปัญหาการลอกเลียนวรรณกรรม (Plagiarism) เป็นปัญหาสำคัญระดับชาติ ทรัพยากรมนุษย์จะไม่สามารถพัฒนาแบบก้าวไกลได้เลย หากไม่พยายามอย่างที่สุดที่จะคิดสร้างสรรค์นวัตกรรมของตนเอง อีกทั้งยังเป็นปัญหาต่อการศึกษาและการวิจัยที่ฝังรากลึกเป็นระยะเวลายาวนาน โดยเยาวชนรุ่นใหม่มีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด ซึ่งหนึ่งในวิธีเหล่านั้นคือ การคัดลอกผลงานของคนอื่น และนำมาอ้างว่าเป็นของตน ดังตัวอย่างในงานวิจัยของกิตติยา (2560)[1] ที่ได้ศึกษาความสำคัญของการป้องกันการโจรกรรมทางวิชาการ ดังนั้น การขโมยความคิดของผู้อื่นมาเป็นของตนเองเป็นปัญหาร้ายแรงที่ส่งผลต่อความเจริญก้าวหน้าและชื่อเสียงของประเทศในระยะยาว โดยเฉพาะอย่างยิ่งถ้าปัญหานี้เกิดขึ้นกับกำลังสำคัญของชาติในอนาคต นั่นคือ นักเรียนนักศึกษา

การลอกเลียนวรรณกรรมในกลุ่มนักเรียนนักศึกษาจึงเป็นเรื่องเร่งด่วนที่ต้องมีมาตรการมาแก้ไข เพื่อไม่ให้เยาวชนของชาติประพฤติตนไปในทางที่ไม่เหมาะสมซึ่งอาจก่อให้เกิดปัญหาใหญ่ที่ร้ายแรงตามมาได้ การแก้ปัญหาที่จุดเริ่มต้นในวัยเยาว์ จะเป็นการปลูกฝังทัศนคติที่ถูกต้องให้กับเยาวชนของชาติซึ่งจะเติบโตไปเป็นกำลังสำคัญของประเทศ และจะนำไปสู่การพัฒนาประเทศอย่างยั่งยืน

การผลิตบัณฑิตทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์เพื่อตอบสนองอุตสาหกรรมการผลิตซอฟต์แวร์ เป็นหนึ่งในกลไกสำคัญในการขับเคลื่อนยุทธศาสตร์และนโยบายของชาติในการสร้างนวัตกรรม การจัดการเรียนการสอนวิชาการเขียนโปรแกรมจึงเป็นหัวใจสำคัญของการสร้างบัณฑิตเพื่อให้จบออกไปประกอบอาชีพทางการผลิตซอฟต์แวร์ แต่ดังที่ได้กล่าวไปก่อนหน้า ปัญหาของเยาวชนรุ่นใหม่ คือ นักเรียนนักศึกษามีแนวโน้มที่จะแก้ปัญหาเฉพาะหน้าด้วยวิธีการที่ง่ายที่สุด โดยการคัดลอกโปรแกรมของคนอื่นมาส่งอาจารย์ ปัญหานี้จะต้องรีบถูกแก้ไข ดังนั้น งานวิจัยนี้จึงนำเสนอวิธีการระบุต้นตอของการลอกเลียนซอร์สโค้ด (Source code) เพื่อให้ทราบถึงแหล่งที่มาของการลอกโปรแกรมในกลุ่มนักเรียนนักศึกษา และเมื่ออาจารย์ผู้สอนทราบว่านักเรียนคนใดเป็นต้นฉบับ คนใดที่เป็นคนลอกเพื่อนมาส่ง จะได้วางแนวทางและมาตรการแก้ไข ตักเตือนและแก้ปัญหาได้ตั้งแต่ต้น

ในบทความวิจัยนี้ ผู้วิจัยได้นำเสนอวิธีการในการระบุว่าซอร์สโค้ดใดเป็นต้นฉบับ และได้สร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน โดยต้นไม้นี้ถูกสร้างขึ้นจากการหาต้นไม้แผ่ทั่วแบบต่ำน้ำหนักรวมมากที่สุด และผู้วิจัยได้ระบุทิศทางของลูกศรเพื่อแสดงว่าซอร์สโค้ดใดเป็นต้นฉบับโดยอาศัยกฎที่ได้จากขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ ซึ่งจะกล่าวรายละเอียดต่อไป

การคัดลอกซอร์สโค้ด

ซอร์สโค้ด (Source code) คือ ชุดคำสั่งทางซอฟต์แวร์ที่ถูกเขียนขึ้นเพื่อเรียบเรียงขั้นตอนการแก้ปัญหาให้คอมพิวเตอร์ประมวลผลตามคำสั่งที่ระบุ วิชาการเขียนโปรแกรมถือเป็นวิชาพื้นฐานที่สอนให้ผู้เรียนเข้าใจวิธีการเขียนชุดคำสั่งเหล่านี้ นิสิตนักศึกษาที่เรียนทางด้านเทคโนโลยีสารสนเทศและคอมพิวเตอร์ จะต้องเรียนวิชาการเขียนโปรแกรมเพื่อจบออกไปประกอบอาชีพในอนาคต ในการเรียนการสอนวิชานี้ ผู้เรียนจำเป็นจะต้องฝึกหัดเขียนโปรแกรมเพื่อแก้ปัญหาที่หลายหลาย เป็นการฝึกทักษะกระบวนการคิดแก้ปัญหา การเรียบเรียงคำสั่งสำหรับเขียนโปรแกรมออกมาเป็นลำดับขั้นตอน ซึ่งกระบวนการฝึกฝนนี้เป็นหัวใจสำคัญที่จะทำให้ผู้เรียนมี

ความรู้ความสามารถในการประกอบอาชีพโปรแกรมเมอร์เมื่อจบการศึกษา

แต่ในปัจจุบัน มีผู้เรียนจำนวนหนึ่งที่ไม่พยายามใช้ความสามารถของตัวเอง แต่จะพยายามหาวิธีที่ง่าย ๆ ที่ไม่ถูกต้องเพื่อที่จะทำให้ตนเองมีงานส่งอาจารย์ นั่นคือการลอกเลียนโปรแกรมของคนอื่นมาส่ง การทำเช่นนี้อาจทำให้ติดเป็นนิสัยและอาจเกิดผลเสียหายร้ายแรงได้ในระยาว ดังนั้น จึงต้องมีมาตรการในการแก้ไขตั้งแต่ในตอนเริ่มแรก เพื่อเป็นการปลูกฝังจิตสำนึกให้กับเยาวชน และเป็นการเสริมสร้างแนวความคิดในการสร้างสรรค์ผลงานด้วยตนเอง

ที่ผ่านได้มีความพยายามในการคิดค้นวิธีการในการตรวจจับความคล้ายกันของซอร์สโค้ด Agrawal และ Sharma (2016)[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 (2012)[3] ใช้ขั้นตอนวิธีการหาลำดับร่วมแบบยาวสุด (LCS: Longest common subsequence) ในการตรวจจับโค้ดที่คล้ายกัน โดยพิจารณาที่ระดับของข้อความหรือตัวอักษรต่าง ๆ ที่ปรากฏในโปรแกรม ซึ่งแม้ว่าวิธี LCS นี้ปกติจะใช้เวลานานในการเปรียบเทียบ แต่บทความวิจัยนี้ก็ได้นำเสนอเทคนิคในการปรับปรุงวิธีการค้นหาให้เร็วขึ้นด้วยการใช้การค้นหาแบบแนวกว้างโดยเทคนิคการทำซ้ำ (Iterative breadth-first search) Son et al. (2013)[4] นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Parse tree kernel) ซึ่งมีความสามารถในการตรวจโครงสร้างของโปรแกรม ซึ่งให้ประสิทธิภาพถึง 93% Zhao et al. (2015)[5] ก็นำเสนอการตรวจจับโดยใช้ต้นไม้ไวยากรณ์ (Abstract syntax tree) ซึ่งเป็นการวิเคราะห์โครงสร้างของโปรแกรม แล้วเปรียบเทียบค่าแฮช (Hash value) ของโปรแกรมว่ามีความคล้ายกันหรือไม่ Kamalim (2016)[6] ใช้การตรวจจับในระดับไบต์โค้ด (Byte code) ซึ่งการตรวจจับแบบนี้มีข้อดีคือ ขั้นตอนวิธีจะพิจารณาจากโค้ดที่คอมไพล์แล้ว ซึ่งจะไม่สนใจเครื่องหมายวรรคตอน (Whitespace) และการตั้งชื่อตัวแปรที่ต่างกันก็จะไม่ส่งผล Qinqin และ Chunhai (2017)[7] ได้รวบรวมนำเสนอมาตรวัดความคล้ายกันของซอร์สโค้ดต่าง ๆ เช่น การนับจำนวนแอททริบิว (Attribute counting method) การวัดโครงสร้างของโปรแกรม (Structural measurement method) การจัดกลุ่ม (Clustering method) เป็นต้น

ล่าสุดในปี 2018 Schneider et al.[8] ได้นำเสนอแนวทางใหม่ในการตรวจจับการคัดลอกซอร์สโค้ดที่แตกต่างจากงานวิจัยที่มีมาก่อนหน้า กล่าวคือ งานวิจัยในอดีตที่ผ่านมาจะตรวจจับจากโปรแกรมที่เขียนเสร็จแล้ว แต่งานวิจัยใหม่นี้จะเก็บข้อมูล (Log) จากการเขียนโปรแกรมของนักเรียนในชั้นเรียน ตลอดกระบวนการเขียนโปรแกรม ซึ่งการใช้ข้อมูลจากพฤติกรรมในการเขียนโปรแกรมนี้สามารถตรวจจับการลอกกันของนักเรียนได้แม่นยำประมาณ 90% อย่างไรก็ตามการใช้วิธีการนี้ยังต้องการการปรับปรุง เนื่องจากประสบกับปัญหาข้อมูลปริมาณมาก และข้อมูลพฤติกรรมการเขียนโปรแกรมมีความซับซ้อนยากต่อการวิเคราะห์

จากขั้นตอนวิธีต่าง ๆ ที่นักวิจัยจำนวนมากได้นำเสนอ ก็มีผู้นำมาพัฒนาเป็นเครื่องมือสำหรับตรวจจับการคัดลอกซอร์สโค้ด หนึ่งในนั้นคือเครื่องมือที่ชื่อว่า MOSS (Measure Of Software Similarity)[9] ซึ่งเป็นโปรแกรมตรวจจับการคัดลอกที่ถูกพัฒนาขึ้นที่มหาวิทยาลัยสแตนฟอร์ด 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 ก่อนที่จะใช้ขั้นตอนวิธีที่นำเสนอในบทความ

วิจัยนี้มาทำการระบุว่าซอร์สโค้ดใดเป็นต้นฉบับ 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

Figure 1 MOSS results

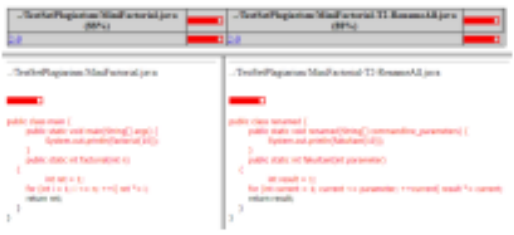


Figure 2 Plagiarized program

ต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด

ต้นไม้แผ่ทั่วของกราฟ (Spanning tree) คือ กราฟย่อยที่ทุกโหนดมีการเชื่อมต่อกันและไม่มีวงวน ซึ่งโดยทั่วไปนักคอมพิวเตอร์จะคุ้นเคยกับต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมน้อยสุด (Minimum spanning tree) ซึ่งสามารถหาต้นไม้นี้ได้จากขั้นตอนวิธีของพริม (Prim's algorithm) หรือ ขั้นตอนวิธีของครัสคัล (Kruskal's algorithm) รายละเอียดของขั้นตอนวิธีสามารถอ่านได้จากหนังสือเรียนวิชาโครงสร้างข้อมูลและอัลกอริทึมทั่วไป เช่น หนังสืออัลกอริทึม (Sedgewick, 2011)[10] โดยหลักการคร่าว ๆ คือ ใช้ขั้นตอนวิธีเชิงละโมบ (Greedy algorithm) ในการเลือกเส้นเชื่อมที่มีค่าน้ำหนักน้อยที่สุดที่ไม่ทำให้เกิดวงวนในกราฟ โดยเลือกเส้นเชื่อมทีละเส้นจนทุกโหนดในกราฟมีเส้นเชื่อมถึงกันทั้งหมด

แต่ในงานวิจัยนี้ต้องการหาความสัมพันธ์เชื่อมโยงของซอร์สโค้ดที่มีความคล้ายกันมากที่สุด จึงจะใช้การหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดแทน (Maximum spanning tree) ซึ่งวิธีการหาต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากสุดสามารถใช้ขั้นตอนวิธีของครัสคัลได้โดยที่กำหนดให้ค่าน้ำหนักของเส้นเชื่อมแต่ละเส้นเป็นค่าติดลบแทน (Pemmaraju and Skiena, 2003, p. 336)[11]

ขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$

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

กระบวนการสร้างและปรับปรุงตัวอย่างคำตอบนี้จะถูกดำเนินการไปจนครบจำนวนรอบที่ผู้ใช้กำหนดหรือจนกว่าจะพบคำตอบที่ผู้ใช้พึงพอใจ

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

หลักการทำงานของขั้นตอนวิธี μ+λ คือ สร้างประชากรรุ่นแรก (ตัวอย่างคำตอบ) ขึ้นมา μ รูปแบบ จากนั้นประเมินค่าความเหมาะสมของประชากรแต่ละตัว แล้วสุ่มเลือกประชากรที่มีค่าความเหมาะสมที่ดีมาเป็นต้นแบบในการสร้างประชากรรุ่นลูกขึ้นมาอีก λ รูปแบบ แล้วจึงคัดเลือกประชากรที่มีค่าความเหมาะสมดีที่สุดโดยพิจารณาจากประชากรทั้งหมด μ+λ ตัว แล้วเลือก mu ตัวที่ดีที่สุดมาเป็นประชากรตั้งต้นในรุ่นถัดไป ขั้นตอนวิธีเชิงวิวัฒนาการแบบ μ+λ แสดงใน Figure 3

รายละเอียดเพิ่มเติมเกี่ยวกับขั้นตอนวิธีเชิงวิวัฒนาการสามารถอ่านได้จาก Mitchell (1996)[12], Goldberg (1989)[13] และการวิเคราะห์พฤติกรรมการทำงานของขั้นตอนวิธีเชิงวิวัฒนาการแบบ μ+λ สามารถศึกษาได้จาก Ter-Sarkisov และ Marsland (2011)[14]

วิธีการวิจัย

งานวิจัยนี้นำเสนอวิธีการในการระบุต้นตอของซอร์สโค้ดจากซอร์สโค้ดที่มีความคล้ายคลึงกัน พร้อมทั้งสร้างต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดซอร์สโค้ดที่มีความคล้ายกัน คณะผู้วิจัยได้นำเสนอกฎที่ใช้ในการระบุต้นตอ ซึ่งกฎนี้ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ μ+λ 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

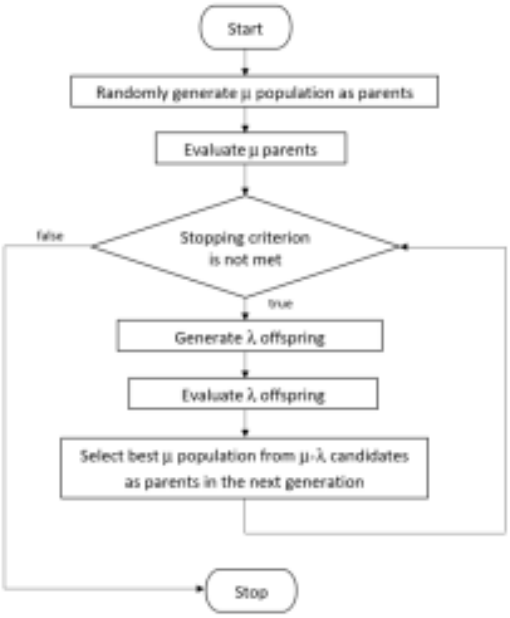


Figure 3 mu+lambda evolutionary algorithm

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 $\mu+\lambda$ 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 เพื่อระบุต้นตอของซอร์สโค้ดที่มีความคล้ายกัน 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%

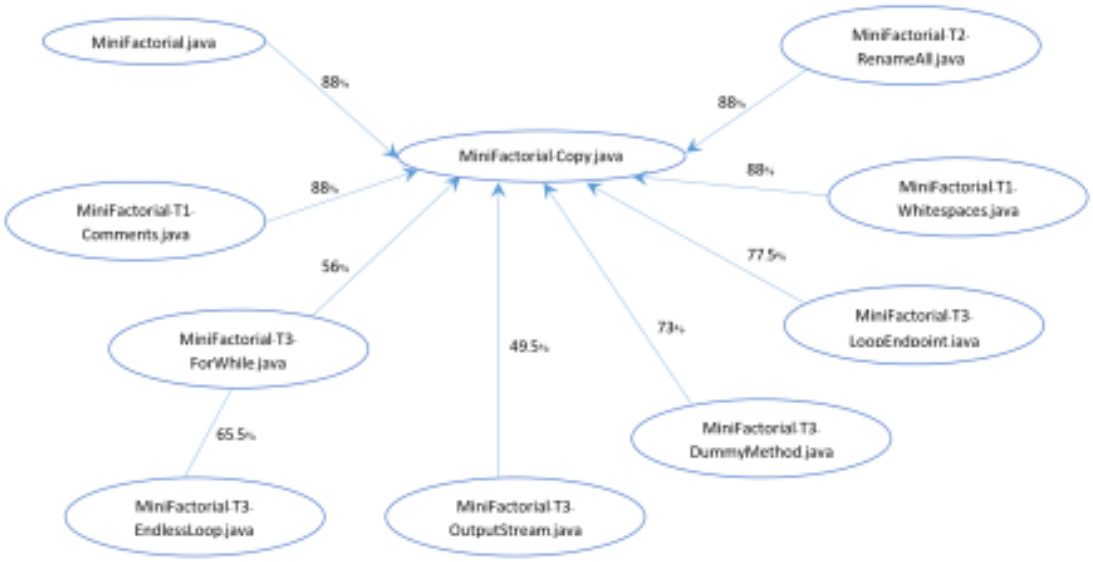


Figure 5 Phylogenetic tree of source code plagiarism

จากแอทริบิวทั้งหมดที่ได้มาจากผลลัพธ์การตรวจความคล้ายกันของซอร์สโค้ดโดยใช้ MOSS นั้น ขั้นตอนวิธีเชิงวิวัฒนาการได้คัดเลือกเหลือเพียงการใช้หมายเลขบรรทัดสุดท้ายที่โค้ดเหมือนกัน และเปอร์เซ็นต์ความคล้ายกันของซอร์สโค้ด เป็นเงื่อนไขในการระบุว่าไฟล์ใดเป็นต้นฉบับ โดยเริ่มต้นจะพิจารณาว่า ถ้าหมายเลขบรรทัดสุดท้ายที่ไฟล์ที่ 1 เหมือนกับไฟล์ที่ 2 น้อยกว่า หมายเลขบรรทัดที่ไฟล์ที่ 2 เหมือนกับไฟล์ที่ 1 หรือ เปอร์เซ็นต์ความคล้ายของไฟล์ที่หนึ่ง มากกว่า เปอร์เซ็นต์ความคล้ายของไฟล์ที่ 2 จะระบุว่า ไฟล์ที่ 1 เป็นต้นฉบับ แต่ถ้าไม่เข้าเงื่อนไขดังกล่าว จะพิจารณาต่อว่า ถ้า หมายเลขบรรทัดสุดท้ายที่ไฟล์ที่ 1 เหมือนกับไฟล์ที่ 2 มากกว่าหมายเลขบรรทัดที่ไฟล์ที่ 2 เหมือนกับไฟล์ที่ 1 จะระบุว่าไฟล์ที่ 2 เป็นต้นฉบับ แต่ถ้าไม่ใช่ก็จะเปรียบเทียบอีกว่า เปอร์เซ็นต์ความคล้ายของทั้ง 2 ไฟล์เท่ากันหรือไม่ ถ้าเท่ากัน จะระบุว่าไฟล์ที่ 1 เป็นต้นฉบับ แต่ถ้าไม่เข้าเงื่อนไขทั้งหมดที่กล่าวมาข้างต้น ก็จะระบุว่าทั้ง 2 ไฟล์ไม่ได้ลอกเลียนกัน

จากการใช้กฎดังกล่าวระบุไฟล์ที่มีการคัดลอก พบว่าสามารถระบุไฟล์ต้นฉบับและระบุไฟล์ที่ไม่ได้ลอกเลียนได้ถูกต้อง 90.24%

เมื่อระบุไฟล์ต้นฉบับได้แล้ว ผู้วิจัยได้สร้างต้นไม้แสดงความสัมพันธ์ และระบุทิศทางของลูกศรเพื่อแสดงลำดับการสืบทอดของซอร์สโค้ดที่มีความคล้ายกัน ซึ่งต้นไม้นี้สร้างได้โดยการหาต้นไม้แผ่ทั่ว

แบบค่าน้ำหนักรวมมากสุด 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-T3-EndlessLoop.java ซึ่งจากต้นไม้แผ่ทั่วแบบค่าน้ำหนักรวมมากที่สุด ได้เลือกเส้นเชื่อมระหว่างไฟล์คู่นี้ไว้ ซึ่งทั้งสองไฟล์ก็มีความคล้ายกัน แต่ไม่ได้ลอกกันมาโดยตรง เส้นเชื่อมระหว่างไฟล์คู่นี้จึงไม่มีหัวลูกศรระบุต้นฉบับ

วิจารณ์และสรุปผล

จากผลการทดลองแสดงให้เห็นว่ากฎที่นำเสนอที่ได้มาจากการวิวัฒนาการคำตอบโดยใช้ขั้นตอนวิธีเชิงวิวัฒนาการแบบ $\mu+\lambda$ ให้ค่าความถูกต้องในการระบุการลอกเลียนซอร์สโค้ด 90.24% ซึ่งได้ทดสอบกับชุดข้อมูลที่มีทั้งการเปลี่ยนแปลงซอร์สโค้ดแบบเล็กน้อย และซอร์สโค้ดที่มีการเปลี่ยนวิธีการในการเขียนโปรแกรม ต้นไม้แสดงความสัมพันธ์และลำดับการสืบทอดการลอกเลียนซอร์สโค้ดที่สร้างขึ้นนี้จะเป็นเครื่องมือหนึ่งที่ช่วยให้อาจารย์เห็นภาพรวมของความคล้ายกันของงานเขียนโปรแกรมที่เป็นการบ้านที่นักเรียนส่งมา 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):348-361
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. 604-636
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 (μ + λ) 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.