การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

Test Case Generation Using Genetic Algorithms

นันทนี ช่วยชู

Nuntanee Chuaychoo

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาการจัดการเทคโนโลยีสารสนเทศ
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Management of Information Technology
Prince of Songkla University
2558

การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

Test Case Generation Using Genetic Algorithms

นันทนี ช่วยชู

Nuntanee Chuaychoo

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาการจัดการเทคโนโลยีสารสนเทศ
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Management of Information Technology
Prince of Songkla University
2558

ชื่อวิทยานิพนธ์ การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม
ผู้เขียน นางสาวนันทนี ช่วยชู
สาขาวิชา การจัดการเทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

.....
(ผู้ช่วยศาสตราจารย์ ดร.สุภาภรณ์ กานต์สมเกียรติ)

คณะกรรมการสอบ

.....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วัชรวลี ตั้งคุปตานนท์)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สุภาภรณ์ กานต์สมเกียรติ)

.....กรรมการ
(ดร.ฐิมาพร เพชรแก้ว)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาการจัดการเทคโนโลยีสารสนเทศ

.....
(รองศาสตราจารย์ ดร.ธีระพล ศรีชนะ)
คณบดีบัณฑิตวิทยาลัย

ขอรับรองว่า ผลงานวิจัยนี้มาจากการศึกษาวิจัยของนักศึกษาเอง และได้แสดงความขอบคุณบุคคลที่มีส่วนช่วยเหลือแล้ว

ลงชื่อ.....
(ผู้ช่วยศาสตราจารย์ ดร.สุภาภรณ์ กานต์สมเกียรติ)
อาจารย์ที่ปรึกษาวิทยานิพนธ์

ลงชื่อ.....
(นางสาวนันทนี ช่วยชู)
นักศึกษา

ข้าพเจ้าขอรับรองว่า ผลงานวิจัยนี้ไม่เคยเป็นส่วนหนึ่งในการอนุมัติปริญญาในระดับใดมาก่อน และไม่ได้ถูกใช้ในการยื่นขออนุมัติปริญญาในขณะนี้

ลงชื่อ.....

(นางสาวนันทนี ช่วยชู)

นักศึกษา

ชื่อวิทยานิพนธ์ การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม
ผู้เขียน นางสาวนันทนี ช่วยชู
สาขาวิชา การจัดการเทคโนโลยีสารสนเทศ
ปีการศึกษา 2558

บทคัดย่อ

การทดสอบซอฟต์แวร์ เป็นขั้นตอนหนึ่งที่สำคัญในกระบวนการผลิตซอฟต์แวร์ การทดสอบช่วยให้ซอฟต์แวร์ที่พัฒนามีความน่าเชื่อถือ และมีความถูกต้องเพิ่มมากขึ้น การสร้างกรณีทดสอบเป็นอีกองค์ประกอบหนึ่งที่มีความสำคัญในกระบวนการทดสอบซอฟต์แวร์ คุณภาพของการทดสอบจะขึ้นอยู่กับประสิทธิภาพของกรณีทดสอบ วิทยานิพนธ์นี้นำเสนอวิธีการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม และมีการสร้างเครื่องมือสำหรับสร้างกรณีทดสอบตามวิธีการที่ได้นำเสนอโดยใช้วิชวลสตูดิโอดอทเน็ต โดยขั้นตอนการสร้างกรณีทดสอบที่ได้นำเสนอไปประยุกต์ใช้กับกรณีศึกษา และใช้การประเมินประสิทธิภาพของกรณีทดสอบโดยวิธีการทดสอบแบบกลายพันธุ์ ผลลัพธ์จากการทดสอบพบว่า กรณีทดสอบที่สร้างขึ้นตามวิธีการที่นำเสนอสำหรับกรณีศึกษา ให้ค่าคะแนนการกลายพันธุ์ค่อนข้างสูง

Thesis Title Test case Generation Using Genetic Algorithm
Author Miss Nuntanee Chuaychoo
Major Program Management of Information Technology
Academic Year 2015

Abstract

Software testing is an important step in the software development process. It makes the developed software more reliable and accurate. Generating test cases is a key component in the software testing process. The quality of testing depends on the test case effectiveness. This thesis proposes a method for generating test cases using genetic algorithms. A test case generation tool is developed with Visual Basic .NET for supporting our method. The proposed method was applied on case studies and mutation testing was used to evaluate the test case performance. The results show that the test cases generated by using the proposed method achieve relatively high mutation scores.

.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ ด้วยความกรุณาและอนุเคราะห์ช่วยเหลือเป็นอย่างดีของ ผู้ช่วยศาสตราจารย์ ดร.สุภาภรณ์ กานต์สมเกียรติ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่าง ๆ อันเป็นประโยชน์อย่างยิ่งในการทำวิจัย อีกทั้งยังช่วยแก้ปัญหาต่าง ๆ ที่เกิดขึ้นระหว่างการดำเนินงานวิจัยอีกด้วย

ขอขอบพระคุณคณะกรรมการสอบทุกท่าน ได้แก่ ผู้ช่วยศาสตราจารย์ ดร.วัชรวลี ตั้งคุปตานนท์ ผู้ช่วยศาสตราจารย์ ดร.อำนาจ เปาะทอง และ ดร.ฐิมาพร เพชรแก้ว ที่กรุณาให้คำปรึกษาแนะนำและตรวจทานแก้ไขวิทยานิพนธ์ให้สมบูรณ์

ขอขอบพระคุณอาจารย์ในหลักสูตรการจัดการเทคโนโลยีสารสนเทศทุกท่านสำหรับการให้ความรู้ ข้อแนะนำและความช่วยเหลือในทุก ๆ ด้านในการทำวิจัย

ขอขอบคุณพี่ๆ หน่วยวิจัยนวัตกรรมสารสนเทศ มหาวิทยาลัยวลัยลักษณ์ทุกคนที่คอยเป็นกำลังใจ และให้ความช่วยเหลือในการทำวิจัย

สุดท้ายนี้ ผู้วิจัยขอขอบพระคุณบิดามารดา และครอบครัว ซึ่งเปิดโอกาสให้ได้รับการศึกษาเล่าเรียน ตลอดจนคอยช่วยเหลือและให้กำลังใจผู้วิจัยเสมอมาจนการศึกษาในครั้งนี้สำเร็จลุล่วงไปได้ด้วยดี

นันทนี ช่วยชู

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

รายการตาราง

รายการตาราง (ต่อ)

รายการภาพประกอบ

รายการภาพประกอบ (ต่อ)

รายการภาพประกอบ (ต่อ)

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของงานวิจัย

ในปัจจุบันคอมพิวเตอร์และเทคโนโลยีสารสนเทศเข้ามามีบทบาทในชีวิตประจำวันของคนเราเป็นอย่างมาก ซอฟต์แวร์ประยุกต์ (Application Software) ถูกพัฒนาขึ้นมามากมายเพื่อใช้ในงานด้านต่างๆ ตามความต้องการของผู้ใช้สำหรับงานเฉพาะอย่าง เพื่ออำนวยความสะดวกและเพิ่มความรวดเร็วในการทำงาน ในการพัฒนาซอฟต์แวร์เหล่านี้ มีความจำเป็นที่จะต้องมีกิจกรรมหรือกระบวนการทดสอบซอฟต์แวร์ เพื่อตรวจสอบความถูกต้องของซอฟต์แวร์ก่อนที่จะนำไปใช้งานจริง

การทดสอบซอฟต์แวร์ (Software Testing) [1] เป็นขั้นตอนในกระบวนการพัฒนาซอฟต์แวร์ที่มีบทบาทและมีความสำคัญ การทดสอบเป็นกิจกรรมที่จัดทำขึ้นเพื่อปรับปรุงคุณภาพของซอฟต์แวร์ ค้นหาข้อบกพร่องและลดข้อผิดพลาดจากการทำงานของซอฟต์แวร์ให้เหลือน้อยที่สุด ขั้นตอนหนึ่งที่สำคัญในกระบวนการทดสอบซอฟต์แวร์ คือการคัดเลือกข้อมูลที่นำมาใช้ในการทดสอบ ดังนั้นการนำเสนอขั้นตอนการสร้างกรณีทดสอบที่ดีจะมีส่วนทำให้การทดสอบมีประสิทธิภาพ โดยกรณีทดสอบที่มีประสิทธิภาพจะต้องครอบคลุมคุณลักษณะต่างๆ ของระบบและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไป การสร้างกรณีทดสอบสามารถทำได้แบบ manual และแบบอัตโนมัติ ซึ่งการสร้างกรณีทดสอบแบบอัตโนมัติจะมีข้อดีกว่าในด้านการประหยัดค่าใช้จ่าย ในปัจจุบันมีงานวิจัยหลายชิ้นที่นำเสนอกระบวนการสร้างกรณีทดสอบ ตัวอย่างเช่น Korel [2] และ Michael [3] ได้นำเสนอวิธีการสร้างกรณีทดสอบจากการสุ่ม โดยได้ทำการสุ่มข้อมูลที่เป็นตัวเลขเพื่อเป็นข้อมูลนำเข้า (Input data) สำหรับการทดสอบซอฟต์แวร์ การสร้างกรณีทดสอบจากการสุ่มนี้ เป็นวิธีพื้นฐานสำหรับการสร้างกรณีทดสอบตามข้อกำหนดของซอฟต์แวร์ แต่กรณีทดสอบที่ได้อาจไม่เพียงพอต่อการนำไปใช้เพื่อทดสอบการทำงานทั้งหมดของซอฟต์แวร์ Monalisa [4] ได้นำเสนอวิธีการสร้างกรณีทดสอบโดยอัตโนมัติจากแผนภาพ UML (Unified Modeling Language) ซึ่งเป็นภาษาที่ใช้แผนภาพต่างๆ ในการอธิบายขั้นตอนการทำงานของซอฟต์แวร์ โดยที่มีการใช้ภาษาหรือภาพสัญลักษณ์ที่เป็นภาษาที่นิยมใช้ในการพัฒนาระบบ งานวิจัยชิ้นนี้ได้ใช้ Use Case Diagram และ Sequence Diagram ในการสร้างกรณีทดสอบ Theerapong Lertphumpanya และ Twittie Senivongse [5] ได้นำเสนอวิธีการสร้างกรณีทดสอบโดยอัตโนมัติสำหรับการทดสอบเว็บเซอร์วิสจากการสร้างกราฟควบคุมการไหลและหาเส้นทางที่เหมาะสมสำหรับการสร้างกรณีทดสอบ

ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithms) เป็นวิธีการค้นหาคำตอบที่เหมาะสมสำหรับปัญหาที่พิจารณา โดยใช้วิธีการคัดเลือกแบบธรรมชาติเลียนแบบแนวคิดวิวัฒนาการของสิ่งมีชีวิตที่ว่า สิ่งมีชีวิตที่แข็งแรงกว่าจะมีโอกาสอยู่รอดในธรรมชาติมากว่าสิ่งมีชีวิตที่อ่อนแอกว่า โดยกระบวนการทำงานของขั้นตอนวิธีเชิงพันธุกรรมจะทำการหาคำตอบที่เหมาะสมโดยเริ่มต้นจากการพิจารณาจากกลุ่มของคำตอบของปัญหาที่ถูกสร้างขึ้นโดยการเข้ารหัส ซึ่งเป็นการแปลงค่าตัวแปรหรือพารามิเตอร์ (Parameters) ของปัญหาให้อยู่ในรูปโครงสร้างของโครโมโซม (Chromosomes) ที่กำหนด แล้วจึงนำกลุ่มของคำตอบนั้นไปคัดเลือกโครโมโซมเพื่อหาคำตอบที่เหมาะสมสำหรับสร้างวิวัฒนาการของคำตอบให้ดีขึ้น โดยใช้การปฏิบัติการทางพันธุศาสตร์เพื่อแลกเปลี่ยนค่าพารามิเตอร์ต่างๆ ระหว่างโครโมโซมที่ถูกคัดเลือก ซึ่งจะทำให้คำตอบของปัญหาถูกปรับปรุงให้ดีขึ้นจนกระทั่งพบคำตอบที่เหมาะสมสำหรับปัญหานั้นๆ

การนำขั้นตอนวิธีทางพันธุกรรมมาประยุกต์ใช้ในการสร้างกรณีทดสอบสำหรับกระบวนการทดสอบซอฟต์แวร์จึงเป็นเรื่องที่น่าสนใจ เนื่องจากหลักการทำงานของขั้นตอนวิธีทางพันธุกรรมมีความสอดคล้องกับขั้นตอนการค้นหากรณีทดสอบและมีการปรับเปลี่ยนกรณีทดสอบจนกระทั่งได้กรณีทดสอบที่เหมาะสมที่สุด ดังนั้น งานวิจัยนี้จึงนำเสนอขั้นตอนการสร้างกรณีทดสอบโดยการประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรมเป็นเครื่องมือในการคัดเลือกชุดของกรณีทดสอบที่มีความเหมาะสมสำหรับการทดสอบซอฟต์แวร์

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อนำเสนอกระบวนการสร้างกรณีทดสอบสำหรับกระบวนการทดสอบซอฟต์แวร์โดยประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรม

1.2.2 เพื่อพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบ ตามกระบวนการที่ได้นำเสนอ

1.3 ขอบเขตของงานวิจัย

1.3.1 เสนอขั้นตอนในการสร้างกรณีทดสอบซอฟต์แวร์โดยอาศัยวิธีการเชิงพันธุกรรม

1.3.2 พัฒนาเครื่องมือเพื่อใช้ในการสร้างกรณีทดสอบซอฟต์แวร์โดยอาศัยวิธีการเชิงพันธุกรรม ตามขั้นตอนที่ได้นำเสนอ

1.4 ระยะเวลาการดำเนินงานวิจัย

ระยะเวลาการดำเนินงานวิจัยทั้งหมด 12 เดือน โดยเริ่มโครงการวันที่ 1 พฤษภาคม 2557 และสิ้นสุดโครงการวันที่ 30 เมษายน 2558

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ได้วิธีการที่ใช้ในการสร้างกรณีทดสอบโดยประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรม

1.5.2 ได้เครื่องมือที่ช่วยในการสร้างกรณีทดสอบโดยอัตโนมัติ เป็นการช่วยลดภาระการดำเนินงานของผู้ทดสอบ

บทที่ 2

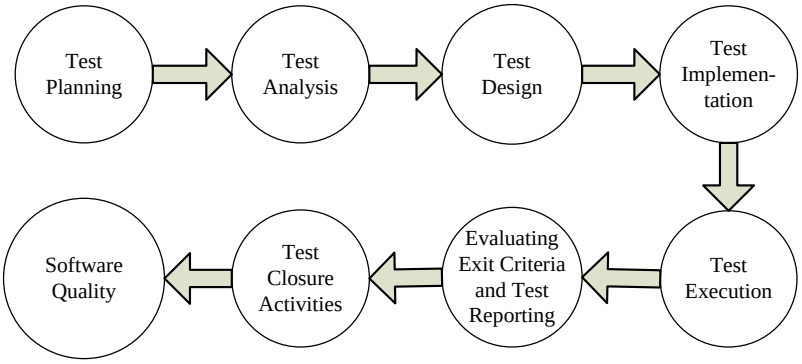
ทฤษฎีและหลักการ

2.1 การทดสอบซอฟต์แวร์ (Software Testing)

การทดสอบซอฟต์แวร์ เป็นการทดสอบความสมบูรณ์ของโปรแกรม รวมทั้งความน่าเชื่อถือและความถูกต้องของผลลัพธ์จากโปรแกรมที่พัฒนาขึ้น การทดสอบเป็นปัจจัยสำคัญปัจจัยหนึ่งของการประกันคุณภาพซอฟต์แวร์ วัตถุประสงค์หลักของการทดสอบซอฟต์แวร์ คือการทดสอบเพื่อค้นหาข้อผิดพลาดที่อาจเกิดขึ้นได้จากซอฟต์แวร์ที่พัฒนาขึ้น การทดสอบที่ดีจะต้องมีความเป็นไปได้สูงที่จะค้นพบข้อผิดพลาดใหม่ๆ และการทดสอบที่ประสบความสำเร็จจะต้องพบข้อผิดพลาด ซึ่งกระบวนการทดสอบที่ดีจะต้องประกอบไปด้วยกรณีทดสอบที่มีความเหมาะสมกับโปรแกรมหรือซอฟต์แวร์นั้นๆ

กรณีทดสอบคือข้อมูลที่ผู้ทดสอบใช้ในการทดสอบฟังก์ชันการทำงานใด ๆ ของซอฟต์แวร์ เพื่อให้ทราบว่าการทำงานของฟังก์ชันการทำงานนั้นมีความถูกต้องหรือไม่ กรณีทดสอบที่ดีต้องมีคุณสมบัติคือ ต้องมีความครอบคลุมคุณลักษณะต่างๆ ของงานที่ทดสอบ และกรณีทดสอบที่มีประสิทธิภาพ คือกรณีทดสอบที่สามารถค้นพบข้อผิดพลาดที่เกิดขึ้นในโปรแกรม

นอกจากนี้การทดสอบซอฟต์แวร์ยังเป็นกระบวนการที่มีความสำคัญต่อการส่งมอบระบบ โดยเฉพาะในขั้นตอนของการทำ UAT (User Acceptance Test) เป็นขั้นตอนการทดสอบซอฟต์แวร์ขั้นตอนสุดท้าย ซึ่งเป็นกระบวนการทดสอบที่ทำโดยผู้ใช้งานระบบ โดยการทดสอบในกระบวนการนี้จะนิยมนำข้อมูลจริงมาใช้ในการทดสอบเพื่อจำลองสถานการณ์จริงในการใช้งานระบบ ซึ่งหากสามารถตรวจพบจุดบกพร่องได้มากและละเอียดเท่าไร ก็จะเป็นการช่วยลดต้นทุนให้แก่ผู้พัฒนาซอฟต์แวร์ และช่วยลดความเสี่ยงให้แก่ผู้ใช้ระบบมากเท่านั้น ดังนั้น การทดสอบซอฟต์แวร์จึงเป็นสิ่งจำเป็นอย่างหลีกเลี่ยงไม่ได้ในอุตสาหกรรมการพัฒนาซอฟต์แวร์ในปัจจุบัน



ภาพที่ 2-1 แสดงกระบวนการทดสอบซอฟต์แวร์ [6]

2.1.1 กระบวนการทดสอบซอฟต์แวร์

กระบวนการในการทดสอบซอฟต์แวร์ แสดงดังภาพที่ 2-1 ซึ่งรายละเอียดในแต่ละขั้นตอนมีดังนี้

(1) Test Planning

เป็นการกำหนดแผนการทดสอบซอฟต์แวร์ ซึ่งรวมถึงการกำหนดตารางกิจกรรมต่างๆ ในการดำเนินการเพื่อทดสอบซอฟต์แวร์ การจัดสรรทรัพยากรที่จำเป็น การกำหนดขอบเขตของงานที่ชัดเจน และการจัดเตรียมสภาพแวดล้อม (Environment) ที่จำเป็นที่ต้องใช้ในการทดสอบทั้งหมด เช่น Hardware, Software (Operating System, Browsers, Database Systems) เครื่องมือและอุปกรณ์ที่เกี่ยวข้อง

(2) Test Analysis

เป็นการกำหนดสถานการณ์ของการทดสอบ ซึ่งสอดคล้องกับความต้องการของระบบและสอดคล้องกับความต้องการในการทำงานของฟังก์ชันต่างๆ ในระบบเพื่อใช้ในการออกแบบการทดสอบ

(3) Test Design

เป็นการจัดทำกรณีทดสอบ (Test Case) ที่ใช้ในการทดสอบ ซึ่งมีลักษณะเป็นรายการตรวจสอบ (Checklist) ที่มีรายละเอียดครอบคลุมทุกคุณสมบัติหรือทุกฟังก์ชันการทำงานของซอฟต์แวร์ที่อยู่ภายใต้การทดสอบ โดยเมื่อทำการทดสอบแล้วเสร็จจะระบุผลลัพธ์ที่ได้จากการทดสอบ (ผ่าน/ไม่ผ่าน) และรายละเอียดอื่นๆ ที่จำเป็นลงไปในรายการตรวจสอบ

(4) Test Implementation

เป็นขั้นตอนการจัดเตรียมกิจกรรมหรือกระบวนการต่างๆ ในการทดสอบ โดยกิจกรรมที่อยู่ภายใต้กระบวนการทดสอบในขั้นตอนนี้ เช่น

- จัดทำตารางปฏิบัติการทดสอบ (Test Execution Schedule)
- จัดเตรียมสภาพแวดล้อมและข้อมูล สำหรับใช้ในการทดสอบ
- จัดเตรียมกระบวนการที่ใช้ในการติดตามผลการทดสอบและข้อบกพร่องที่เกิดจากการทดสอบ
- จัดเตรียมเครื่องมือต่างๆ

(5) Test Execution

เป็นขั้นตอนการดำเนินการทดสอบ

- ดำเนินการทดสอบตามตารางปฏิบัติการทดสอบ
- ดำเนินการบันทึก, แก้ไข, ติดตามข้อผิดพลาดที่เกิดขึ้น
- ติดตามความก้าวหน้าของการทดสอบ
- ควบคุมการทดสอบให้เป็นไปตามแผนที่วางไว้

(6) Evaluating Exit Criteria and Test Reporting

เป็นการตรวจสอบว่าผลการทดสอบเป็นอย่างไร เช่น ได้ทำการทดสอบครบถ้วนตามตารางปฏิบัติการทดสอบแล้วหรือไม่ หรือมีการแก้ไขข้อผิดพลาดที่เกิดจากการทดสอบทั้งหมดแล้วหรือไม่ เป็นต้น และที่สำคัญยังเป็นการรายงานความคืบหน้าการทดสอบให้ทราบ เช่น

- จำนวนกรณีทดสอบที่ผ่าน/ไม่ผ่านการทดสอบ หรืออยู่ระหว่างการทดสอบ
- จำนวนข้อบกพร่องที่พบจำแนกตามระดับความรุนแรง (Severity) และสถานะ (Status)

(7) Test Closure Activities

คือการสรุปผลหรืออภิปรายผลการทดสอบ เป็นขั้นตอนสุดท้ายในกระบวนการทดสอบซอฟต์แวร์ กิจกรรมที่อยู่ภายใต้กระบวนการนี้ เช่น

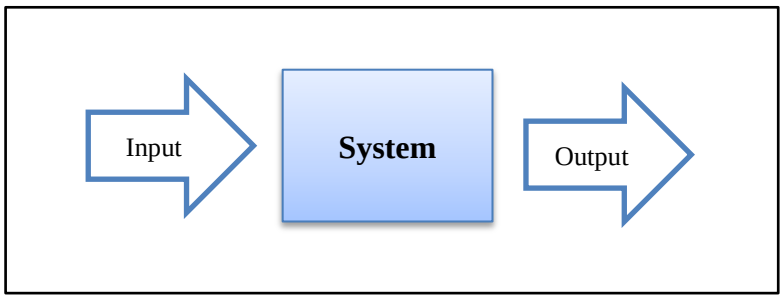
- ตรวจสอบความครบถ้วนของการทดสอบว่าเป็นไปตามตารางปฏิบัติการทดสอบหรือไม่ และทำการแก้ไขข้อบกพร่องที่พบทั้งหมดแล้วหรือไม่

- ทำการส่งมอบงาน เช่น ผลการทดสอบ ข้อมูลที่เกิดจากการทดสอบ และรายงานให้ผู้ที่เกี่ยวข้องรับทราบ
- จัดเก็บข้อมูลและเอกสารสำคัญ เช่น ผลการทดสอบ ข้อมูลที่เกิดจากการทดสอบ และรายงานต่างๆ ลงในแหล่งจัดเก็บข้อมูล (Configuration Management System)

2.1.2 ระดับของการทดสอบซอฟต์แวร์

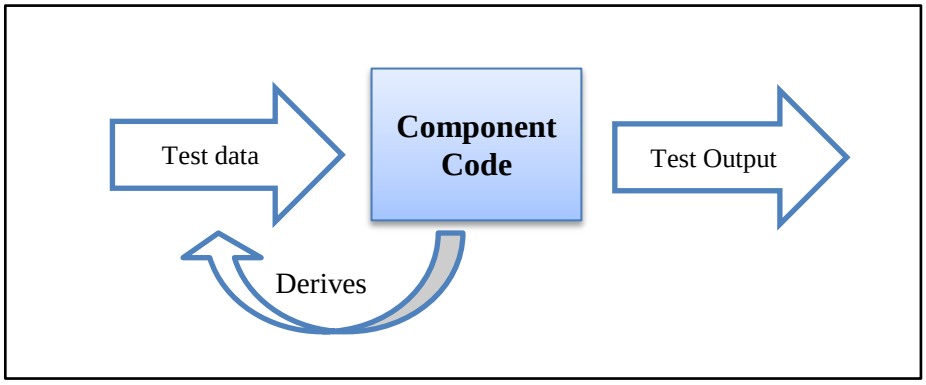
สำหรับเทคนิคในการทดสอบซอฟต์แวร์ (Software Testing Technique) ที่นิยมใช้ได้แก่

1) Black-box Testing คือการทดสอบการทำงานของซอฟต์แวร์ที่ไม่สนใจกลไกภายในของระบบ การทำการทดสอบเน้นเพื่อการตรวจสอบผลการทำงานของระบบในแต่ละหน้าที่ตามข้อกำหนดความต้องการ (Requirement Specification) ว่าถูกต้องตามความต้องการหรือไม่ ผู้ที่ทำการทดสอบ (Tester) นั้น ไม่ทำการเข้าถึงรหัสต้นฉบับ (Source Code) โดยรหัสต้นฉบับจะถูกพิจารณาให้เป็น “Big Black Box” ที่ผู้ทดสอบระบบไม่สามารถเห็นภายในได้ ผู้ทดสอบระบบจะรู้เพียงอย่างเดียวว่าสามารถให้ Input แก่ระบบและระบบจะส่ง Output ออกมาดังภาพที่ 2-2



ภาพที่ 2-2 Black-box Testing

2) White-box Testing คือการทดสอบการทำงานของซอฟต์แวร์ซึ่งพิจารณากลไกภายในของระบบโดยจะมุ่งเน้นพิจารณาโครงสร้างภายใน แนวคิดพื้นฐานของการทดสอบแบบนี้คือ การที่ความผิดพลาดของซอฟต์แวร์จะถูกตรวจพบได้นั้น จะต้องทำการทดลองเรียกใช้คำสั่งต่างๆ ในโปรแกรมให้มากที่สุด การทดสอบโครงสร้างของโปรแกรมเช่นนี้มีความสำคัญมาก เนื่องจากจะช่วยในการวิเคราะห์การทำงานของโปรแกรมเพื่อค้นหาข้อบกพร่องที่อาจจะเกิดขึ้นจากการทำงานของคำสั่งต่างๆในโปรแกรมได้



ภาพที่ 2-3 White-box Testing

การทดสอบซอฟต์แวร์สามารถทำได้หลายระดับซึ่งประกอบด้วย 5 ระดับสามารถแบ่งออกได้ ดังนี้

(1) การทดสอบโมดูล (Unit Testing หรือ Module Testing)

เป็นการทดสอบโปรแกรมทีละโมดูลเพื่อหาข้อผิดพลาด หรือเป็นการทดสอบส่วนที่เล็กที่สุดของโปรแกรมอาจจะเป็น Function หรือ Class โดยกำหนดข้อมูลเข้า (Input) แล้วทดสอบข้อมูลออก (Output) การทดสอบในระดับนี้มักจะทำโดย Programmer และอยู่ในกลุ่มของ White-Box Testing

(2) การทดสอบการรวมกัน (Integration Testing)

เป็นการทดสอบการรวมโมดูล โดยจะแบ่งเป็น 2 ลักษณะ คือ การทดสอบแบบเพิ่มโมดูลจากบนลงล่าง (Top-down Approach) และการทดสอบแบบเพิ่มโมดูลจากล่างขึ้น (Bottom-up Approach)

(3) การทดสอบระบบย่อย (Sub-System Testing)

เป็นการทดสอบระบบย่อยทีละระบบเพื่อหาข้อผิดพลาด ซึ่งการทดสอบในระดับนี้มักจะเป็น Black-box Testing ซึ่งระดับนี้จะทำการทดสอบความเข้ากันได้ของระบบจากการรวมองค์ประกอบย่อย ๆ ของโปรแกรมเข้ามาทำงานด้วยกัน (เนื่องจากการพัฒนาโปรแกรมใดๆ ส่วนใหญ่จะทำการแบ่งตัวโปรแกรมออกเป็นส่วนย่อย ๆ เรียกว่า โมดูล (Module), ฟังก์ชัน (Function) หรือส่วนประกอบ (Component) เพื่อความสะดวกในการพัฒนาและการจัดการ ดังนั้นจึงต้องมีการทดสอบว่าเมื่อนำมารวมกันเพื่อใช้งานเป็นโปรแกรมหลักโปรแกรมเดียวแล้ว จะสามารถทำงานร่วมกันได้อย่างถูกต้องหรือไม่)

(4) การทดสอบทั้งระบบ (System Testing)

จะเป็นการนำระบบย่อยที่ผ่านการทดสอบมาแล้ว มาทดสอบอีกครั้งโดยเพิ่มอีกระบบย่อยหนึ่งเข้าไปเรื่อย ๆ จนกระทั่งครบทุก ๆ ระบบย่อยของตัวโปรแกรมของระบบงานนั้น โดยทุกโปรแกรมเมื่อทำงานร่วมกันแล้วจะได้ผลลัพธ์ที่ถูกต้อง คือทดสอบทั้งระบบเพื่อทดสอบการทำงานโดยรวม

(5) การทดสอบการยอมรับของระบบโดยผู้ใช้ (Acceptance Testing)

เป็นการทดสอบ โดยจะเป็นหน้าที่ของผู้ใช้งาน (End User) ที่จะทำการทดสอบโปรแกรมด้วยตัวเอง เพื่อดูว่าการทำงานของโปรแกรมนั้นเป็นไปตามความต้องการ

2.1.3 การทดสอบแบบความครอบคลุมรหัสโปรแกรม (Code Coverage Testing)

เป็นการทดสอบโดยมีการคำนึงถึงส่วนของรหัสโปรแกรม ที่ประกอบด้วยส่วนของข้อมูลและส่วนของโปรแกรม ข้อมูล ได้แก่ ตัวแปร ค่าคงที่ อาร์เรย์ โครงสร้างข้อมูล อินพุทจากแป้นพิมพ์ และเมาส์ อินพุทและเอาท์พุทจากไฟล์และหน้าจอ อุปกรณ์ไอโอ เป็นต้น การทดสอบโครงสร้างของโปรแกรมสามารถกระทำได้หลายระดับตามระดับของความเข้มข้นหรือที่เรียกว่า ความครอบคลุมรหัสโปรแกรม (Code Coverage) [7] ตัวอย่างระดับความครอบคลุม เช่น

(1) การครอบคลุมคำสั่ง (Statement Coverage - SC)

เป็นรูปแบบพื้นฐานที่สุดของความครอบคลุมรหัสโปรแกรม เป้าหมายของการทดสอบในระดับนี้คือ ต้องการให้ทุกคำสั่งในโปรแกรมได้รับการทดสอบอย่างน้อยหนึ่งครั้ง

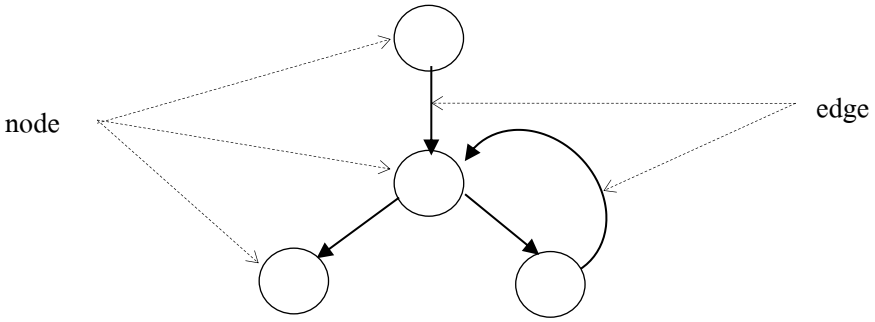
(2) การครอบคลุมการตัดสินใจ (Decision Coverage - DC)

เป็นการทดสอบความครอบคลุมในระดับที่สูงขึ้น โดยต้องมีการพิจารณาการตัดสินใจ (Decision) ของชุดคำสั่งทั้ง 2 กรณี คือ กรณีที่การตัดสินใจเป็นจริง (True) และกรณีที่การตัดสินใจเป็นเท็จ (False) เป้าหมายของการทดสอบในระดับนี้คือ ต้องการให้ทุกการตัดสินใจในโปรแกรมได้รับการทดสอบอย่างน้อยหนึ่งครั้งทั้งกรณีที่การตัดสินใจเป็นจริงและเป็นเท็จ

(3) การครอบคลุมเส้นทาง (Path Coverage - PC)

การทดสอบในระดับนี้จะพิจารณาชุดคำสั่งในลักษณะของกราฟควบคุมการไหล (Control Flow Graph, CFG) ภาพที่ 2-4 แสดงกราฟควบคุมการไหล ซึ่งประกอบด้วย 4 โหนด และ 4 เอจ หากพิจารณาการทดสอบที่ระดับการครอบคลุมคำสั่ง สามารถเทียบได้กับการครอบคลุม

โหนด (Node Coverage) ในกราฟ และในระดับการครอบคลุมการตัดสินใจ สามารถเทียบได้กับการครอบคลุมเอจ (Edge Coverage) ในกราฟ การทดสอบในระดับครอบคลุมเส้นทาง หมายถึง การกำหนดให้มีการทดสอบครอบคลุมทุกๆ เส้นทางที่เป็นไปได้ของกราฟ ซึ่งในทางปฏิบัติไม่สามารถกระทำได้ เนื่องจากกราฟมักมีเส้นทางมากมาย เช่น มีเส้นทางที่มีการวนซ้ำ ดังนั้น Ammann และ Outfutt [8] จึงนำเสนอหลักการครอบคลุมเส้นทางในทางปฏิบัติ กล่าวคือ หากชุดคำสั่งที่ทำการทดสอบมีการวนรอบ (Loop) ที่มีจำนวนรอบได้ไม่จำกัดจำนวน ให้ทำการทดสอบการวนรอบใน 3 กรณี คือ 1) ไม่มีการวนรอบเลย 2) วนรอบหนึ่งครั้ง และ 3) วนรอบหลายครั้ง ในกรณีที่สามนี้เราไม่สามารถกำหนดตายตัวได้ว่าจำนวนการวนรอบเท่าไร โดยเราสามารถกำหนดให้เป็น 2 หรือจำนวนเท่าใดครั้ง



ภาพที่ 2-4 กราฟควบคุมการไหล

(4) การครอบคลุมเงื่อนไขแบบหลายกรณี (Multiple Condition Coverage - MCC)

การทดสอบในระดับนี้จะพิจารณาความสัมพันธ์ระหว่างค่าของเงื่อนไขแต่ละค่าในทุกกรณี ซึ่งจะทำให้มีกรณีที่ต้องพิจารณาเท่ากับ 2^n เมื่อ n คือจำนวนของเงื่อนไขย่อย และค่าความเป็นไปได้ของแต่ละเงื่อนไขย่อยคือกรณีที่เป็นจริงและเท็จ ตัวอย่างเช่น เงื่อนไข *a or b* ซึ่งประกอบด้วยค่าของเงื่อนไข 2 ตัว คือ a และ b ดังนั้นกรณีทดสอบมีจำนวนเท่ากับ 4 ดังตารางที่ 2-1

ตารางที่ 2-1 ตัวอย่างกรณีทดสอบสำหรับการทดสอบในระดับครอบคลุมเงื่อนไขแบบหลายกรณี

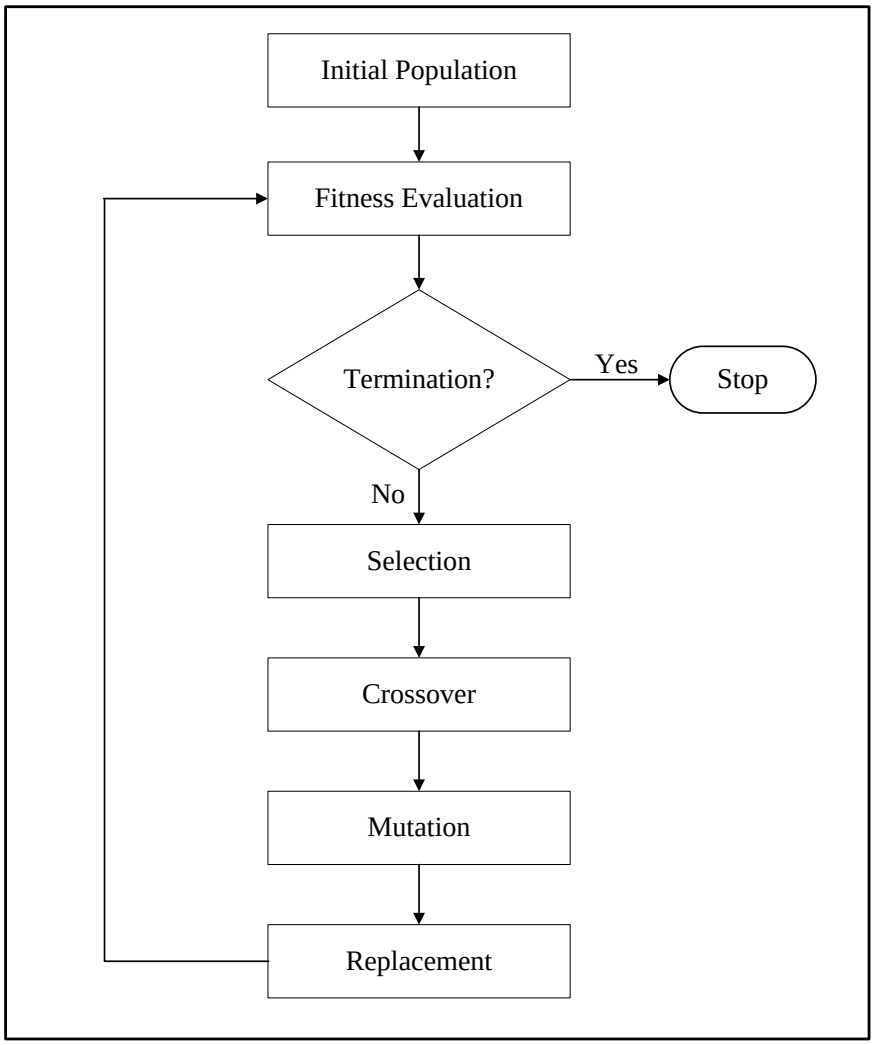
กรณีทดสอบ	a	b	a or b
1	จริง	จริง	จริง
2	จริง	เท็จ	จริง
3	เท็จ	จริง	จริง
4	เท็จ	เท็จ	เท็จ

เกณฑ์การประเมินความครอบคลุมในการทดสอบซอฟต์แวร์แบบโครงสร้างที่ได้กล่าวมาข้างต้น มีข้อดีข้อเสียแตกต่างกัน อย่างไรก็ตามการทดสอบซอฟต์แวร์ไม่ว่าจะทำการทดสอบที่ระดับความครอบคลุมใดๆ ก็ยังคงมีเป้าหมายเดียวกันคือ เพื่อต้องการที่จะครอบคลุมการทำงานของชุดคำสั่งให้ครบถ้วน ต้องการลดความซับซ้อนของการทดสอบที่ไม่จำเป็น และเพื่อหาข้อผิดพลาดที่อาจเกิดขึ้นกับซอฟต์แวร์

2.2 ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm)

ขั้นตอนวิธีเชิงพันธุกรรม เป็นกระบวนการหนึ่งที่อาศัยทฤษฎีการวิวัฒนาการของสิ่งมีชีวิต นำเสนอโดย John Holland [10] และเขาได้นำทฤษฎีดังกล่าวมาประยุกต์ใช้กับเทคนิคการค้นหาในเรื่อง Local Search Algorithm และต่อมาได้นำขั้นตอนวิธีเชิงพันธุกรรมมาใช้เป็นเทคนิคการเรียนรู้ด้วย ขั้นตอนวิธีเชิงพันธุกรรมสามารถนำมาใช้เพื่อค้นหาคำตอบที่ดีขึ้นและเหมาะสมมากขึ้นโดยผ่านการเรียนรู้ด้วยตัวเอง เปรียบเสมือนการวิวัฒนาการของสิ่งมีชีวิตที่สามารถวิวัฒนาการเพื่อการดำรงชีวิตอยู่ โดยนำมาจากแนวคิดที่ว่าสิ่งมีชีวิตทั้งหลายมีทั้งส่วนดีและส่วนที่ไม่ดี ซึ่งลักษณะที่ดีย่อมมีโอกาสอยู่รอดได้มากกว่าและจะได้รับการสนับสนุนให้มีการถ่ายทอดลักษณะทางพันธุกรรมไปยังรุ่นต่อไป

กระบวนการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแสดงดังภาพที่ 2-5 โดยการทำงานจะเริ่มจากการสุ่มโครโมโซมขึ้นมาจำนวนหนึ่งเพื่อสร้างเป็นกลุ่มประชากรต้นกำเนิด (Initial Population) และทำการประเมินค่าความเหมาะสม (Fitness Evaluation) จากฟังก์ชันความเหมาะสมที่กำหนดไว้ จากนั้นจะคัดเลือกประชากร (Selection) มาจำนวนหนึ่งเพื่อใช้เป็นต้นกำเนิดทางสายพันธุ์หรือเป็นกลุ่มประชากรรุ่นพ่อแม่ จากนั้นจะนำประชากรเหล่านี้เข้าสู่กระบวนการปฏิบัติการทางสายพันธุ์ (Genetic Operation) ได้แก่ การไขว้เปลี่ยน (Crossover) และการกลายพันธุ์ (Mutation) ซึ่งจะทำให้เกิดกลุ่มประชากรที่เป็นรุ่นลูก และเมื่อนำประชากรกลุ่มนี้ไปประเมินค่าความเหมาะสมแล้วพบคำตอบที่เหมาะสมกับปัญหา กระบวนการทำงานก็จะจบลง แต่หากคำตอบที่ได้ยังไม่เหมาะสมก็จะดำเนินต่อไปโดยการนำประชากรรุ่นลูกที่ได้ไปแทนที่ (Replacement) ประชากรรุ่นพ่อแม่ เมื่อทำการแทนที่เสร็จแล้วก็จะนำประชากรกลุ่มใหม่ที่ได้กลับเข้าสู่กระบวนการคัดเลือกใหม่อีกครั้ง ซึ่งกระบวนการเหล่านี้จะดำเนินการซ้ำไปเรื่อยๆ จนกระทั่งได้ประชากรกลุ่มที่เหมาะสม



ภาพที่ 2-5 การดำเนินการของขั้นตอนวิธีเชิงพันธุกรรม [10]

2.1.1 การสร้างกลุ่มประชากรต้นกำเนิด (Initial Population)

เป็นขั้นตอนแรกของกระบวนการในขั้นตอนวิธีเชิงพันธุกรรม โครโมโซมมักถูกกำหนดในรูปแบบที่แตกต่างกันออกไปตามปัญหานั้นๆ เช่น

1) โครโมโซมแบบเลขฐานสอง (Binary) เป็นรูปแบบโครโมโซมเริ่มแรกที่นำมาใช้แก้ปัญหาของขั้นตอนวิธีเชิงพันธุกรรม โดยลักษณะของรูปแบบโครโมโซมแบบนี้คือ ทุกตำแหน่งจะมีค่าเป็น บิต 0 หรือ 1

2) โครโมโซมแบบตัวเลข (Value) รูปแบบโครโมโซมในลักษณะนี้ ทุกตำแหน่งของยีนของโครโมโซมจะมีค่าบางค่าซึ่งสามารถเชื่อมโยงไปยังปัญหาได้ เช่น ตัวอักษร จำนวนจริง คำสั่ง หรืออื่นๆ รูปแบบโครโมโซมแบบนี้สามารถใช้ได้กับปัญหาที่ค่อนข้างซับซ้อน

ในการหาผลลัพธ์ด้วยวิธีการทางพันธุกรรมนั้น ผลลัพธ์เริ่มต้นจะได้มาจากการสุ่มเลือกกลุ่มของผลลัพธ์ที่เป็นไปได้ภายในขอบเขตของข้อมูลให้มีจำนวนโครโมโซมเท่ากับจำนวนประชากรที่ได้กำหนดไว้ ตัวอย่างการกำหนดโครโมโซมอย่างง่ายในขั้นตอนวิธีเชิงพันธุกรรมเป็นเซ็ตของยีนที่เป็นเลขฐานสอง เช่น {100101} โดยตำแหน่งของยีนแต่ละยีนในโครโมโซมจะแทนลักษณะขององค์ประกอบย่อยของชุดคำตอบของปัญหา ซึ่งการกำหนดยีนและโครโมโซม สามารถกำหนดในรูปแบบอื่นได้ ขึ้นอยู่กับโครงสร้างขององค์ประกอบย่อยของคำตอบและลักษณะของปัญหาที่ต้องการแก้ ตัวอย่างเช่น อาจกำหนดโครโมโซมเป็นเซ็ตของยีนที่เป็นเลขฐานสิบ หรืออาจกำหนดโครงสร้างของโครโมโซมเป็น Matrix ของยีนที่เป็นตัวเลขหรือตัวอักษร ตามความต้องการและลักษณะของปัญหา

2.1.2 การประเมินค่าความเหมาะสม (Fitness Evaluation)

โครโมโซมทุกตัวจะต้องมีค่าซึ่งบ่งบอกถึงความเหมาะสมที่จะพิจารณาว่าสมควรนำไปสืบสายพันธุ์ต่อหรือไม่ การประเมินค่าความเหมาะสม จึงเป็นกระบวนการหนึ่งที่ใช้ในการพิจารณาเงื่อนไขของการดำเนินการทางพันธุกรรมว่าต้องดำเนินการต่อหรือจบกระบวนการ หากประเมินค่าความเหมาะสมแล้วพบคำตอบที่เหมาะสมกับปัญหา กระบวนการทำงานก็จะจบลง แต่หากคำตอบที่ได้ยังไม่เหมาะสมก็จะดำเนินต่อไป

2.1.3 การคัดเลือก (Selection)

ประชากรรุ่นใหม่จำเป็นต้องเกิดมาจากประชากรรุ่นพ่อ-แม่ที่มีลักษณะพันธุ์ที่ดี ดังนั้นในการเลือกประชากรรุ่นพ่อ-แม่เพื่อสืบพันธุ์จะต้องเลือกจากโครโมโซมที่ให้ผลลัพธ์ที่ดีเพื่อเป็นคำตอบเริ่มต้นให้กับประชากรรุ่นต่อไปโดยอาศัยทฤษฎีการอยู่รอดของสิ่งมีชีวิต จึงทำให้เกิดรูปแบบมากมายในการเลือกโครโมโซมที่น่าพอใจที่สุดเพื่อนำไปสืบสายพันธุ์ ดังนี้

1) การคัดเลือกแบบวงล้อรูเล็ต (Roulette Wheel) คือ โครโมโซมที่มีค่าความเหมาะสมที่ดีกว่ามีโอกาสในการถูกสุ่มเลือกเป็นพ่อ-แม่พันธุ์ที่สูงกว่า

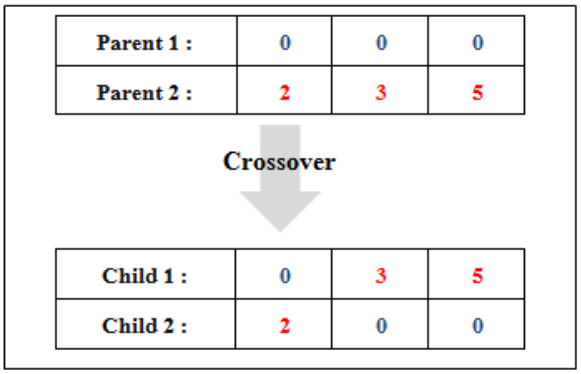
2) การคัดเลือกแบบช่วง (Ranking) คือการเลือกประชากรที่มีค่าความเหมาะสมที่ดีที่สุดโดยไม่สนใจประชากรตัวอื่น

3) การคัดเลือกแบบ Elitist เป็นแนวคิดที่ป้องกันการหาของเส้นทางที่ดีที่สุด คือมีการคัดลอกโครโมโซมที่ดีที่สุดไว้ก่อน ส่วนประชากรที่เหลือที่จะต้องคัดเลือกจะใช้วิธีการเลือกแบบอื่นๆ

2.1.4 การไขว้เปลี่ยน (Crossover)

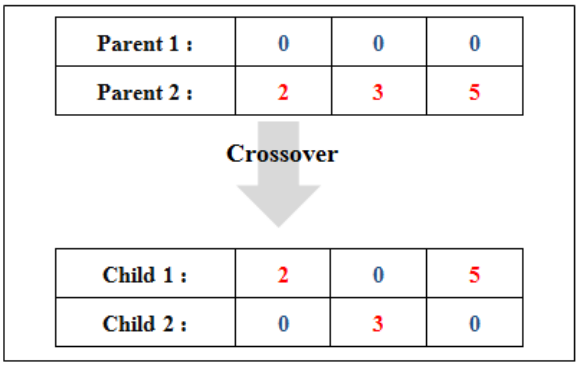
เป็นกระบวนการที่สำคัญของขั้นตอนวิธีเชิงพันธุกรรม การไขว้เปลี่ยนคือการที่โครโมโซมในรุ่นพ่อ-แม่ จะมีการสลับลักษณะทางพันธุกรรมซึ่งกันและกัน โดยอัตราส่วนของจำนวนโครโมโซมรุ่นลูกที่ถูกสร้างขึ้นจากกลุ่มประชากรในแต่ละรุ่นต่อขนาดของประชากรทั้งหมดจะเรียกว่า **อัตราความน่าจะเป็นในการไขว้เปลี่ยน (Probability of Crossover)** ซึ่งเป็นพารามิเตอร์ที่สำคัญในการหาคำตอบของกระบวนการเชิงพันธุกรรม ลักษณะการไขว้เปลี่ยนมี 2 รูปแบบ คือ

1) การไขว้เปลี่ยนแบบจุดเดียว (Single-point Crossover) จะสุ่มจุดตัดขึ้นมาเพียงจุดเดียวบนโครโมโซมพ่อกับโครโมโซมแม่ เพื่อใช้เป็นจุดสำหรับไขว้เปลี่ยนระหว่างโครโมโซม ทำให้เกิดเป็นโครโมโซมใหม่ขึ้นมาตัวอย่างการไขว้เปลี่ยนที่ตำแหน่งที่ 2 แสดงดังภาพที่ 2-6



ภาพที่ 2-6 การไขว้เปลี่ยนแบบจุดเดียว

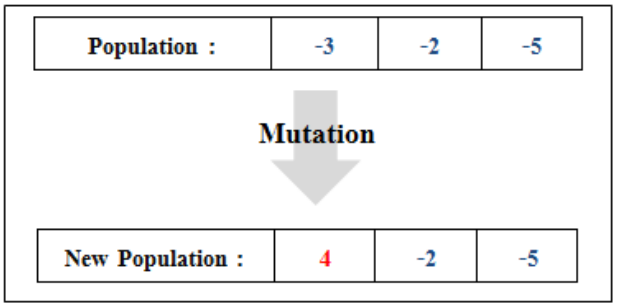
2) การไขว้เปลี่ยนแบบสองจุด (Double-point Crossover) จะสุ่มจุดตัดขึ้นมาสองจุดบนโครโมโซมพ่อกับโครโมโซมแม่ เพื่อใช้เป็นจุดสำหรับไขว้เปลี่ยนระหว่างโครโมโซม ทำให้เกิดเป็นโครโมโซมใหม่ขึ้นมา ตัวอย่างการไขว้เปลี่ยนที่ตำแหน่งที่ 1 และ 3 แสดงดังภาพที่ 2-7



ภาพที่ 2-7 การไขว้เปลี่ยนแบบสองจุด

2.1.5 การกลายพันธุ์ (Mutation)

การกลายพันธุ์ คือ การเปลี่ยนลักษณะทางพันธุกรรมภายในโครโมโซมรุ่นลูกเอง หลังจากที่ได้ทำการสลับสายพันธุ์มาแล้วขั้นตอนวิธีเชิงพันธุกรรมได้นำหลักการนี้มาช่วยสร้างคำตอบใหม่ โดยจะยอมให้ค่าภายในโครโมโซมสามารถเกิดการกลายพันธุ์ได้ เป็นขั้นตอนที่อาจช่วยให้โครโมโซมมีค่าความเหมาะสมดีขึ้นหลังจากการไขว้เปลี่ยน ซึ่งตำแหน่งที่กลายพันธุ์จะเกิดจากการสุ่มโดยเปอร์เซ็นต์ที่จะเกิดการกลายพันธุ์หาได้จาก**อัตราความน่าจะเป็นในการกลายพันธุ์ (Probability of Mutation)** ตัวอย่างการกลายพันธุ์ที่ตำแหน่งที่ 1 แสดงดังภาพที่ 2-8



ภาพที่ 2-8 การกลายพันธุ์

2.1.6 การแทนที่ (Replacement)

กลุ่มประชากรรุ่นลูกที่ได้จากการะบวนการไขว้เปลี่ยน (Crossover) และ การกลายพันธุ์ (Mutation) จะถูกนำไปแทนที่ข้อมูลที่เป็นประชากรรุ่นพ่อแม่ และจะกลายเป็นประชากรรุ่นพ่อแม่ใหม่ ต่อจากนั้นจะนำข้อมูลชุดใหม่ไปเข้าสู่กระบวนการประเมินค่าความเหมาะสม (Fitness Evaluation) อีกครั้งจนกว่าจะได้กลุ่มประชากรที่เหมาะสม

2.1.7 พารามิเตอร์ (Parameters)

พารามิเตอร์ที่สำคัญหรือเป็นพื้นฐานของขั้นตอนวิธีเชิงพันธุกรรม มีดังนี้

1) ขนาดของประชากร (Population Size) เป็นพารามิเตอร์ที่มีความสำคัญและส่งผลต่อการทำงาน เพราะหากกำหนดขนาดประชากรมากเกินไปจะต้องใช้ทรัพยากรสำหรับการคำนวณมากขึ้น เช่น ใช้หน่วยความจำและเวลาในการประมวลผลมากขึ้น เป็นต้น

2) อัตราความน่าจะเป็นในการไขว้เปลี่ยน (Probability of Crossover) เป็นพารามิเตอร์ที่กำหนดความน่าจะเป็นของการเกิดการไขว้เปลี่ยน (Crossover) ของโครโมโซม ซึ่งจะมีค่าอยู่ระหว่าง 0 ถึง 1 (หรือสามารถแปลงให้อยู่ในรูปแบบร้อยละได้) หากกำหนดอัตราการไขว้เปลี่ยนเป็น 1 แสดงว่าจะเกิดกระบวนการไขว้เปลี่ยนโครโมโซมทุกครั้ง ทำให้คำตอบมีความหลากหลายเนื่องจากเกิดโครโมโซมลูกขึ้นใหม่ตลอดเวลา ส่วนกรณีที่กำหนดอัตราการไว้เปลี่ยนเป็น 0

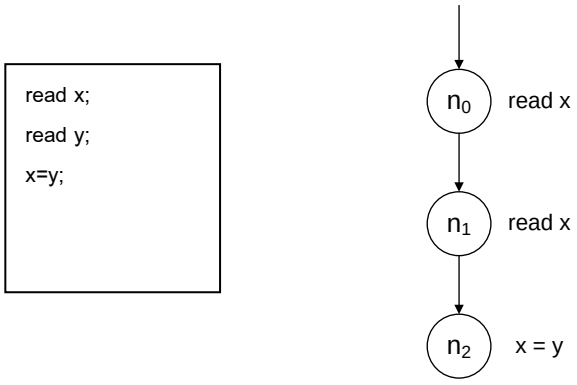
แสดงว่าไม่มีโอกาสเกิดกระบวนการไขว้เปลี่ยนแต่จะคัดลอกโครโมโซมที่พิจารณานั้นมาเป็นโครโมโซมลูกแทน ทำให้พบคำตอบของปัญหาได้ยากเนื่องจากไม่มีคำตอบใหม่ที่เกิดขึ้น

3) อัตราความน่าจะเป็นในการกลายพันธุ์ (Probability of Mutation) เป็นพารามิเตอร์ที่กำหนดความน่าจะเป็นในการกลายพันธุ์ (Mutation) โดยอัตราการเกิดจะอยู่ระหว่าง 0 ถึง 1 หากกำหนดอัตราการกลายพันธุ์เป็น 1 แสดงว่าทุกตำแหน่งบนโครโมโซมสามารถเกิดการกลายพันธุ์หลังจากกระบวนการไขว้เปลี่ยนได้ทุกครั้ง ส่วนกรณีที่กำหนดอัตราการกลายพันธุ์เป็น 0 แสดงว่าไม่มีโอกาสเกิดการกลายพันธุ์ภายในโครโมโซม โดยส่วนใหญ่มักใช้กับกรณีที่ต้องการหาคำตอบที่ได้จากการไขว้เปลี่ยนเท่านั้น

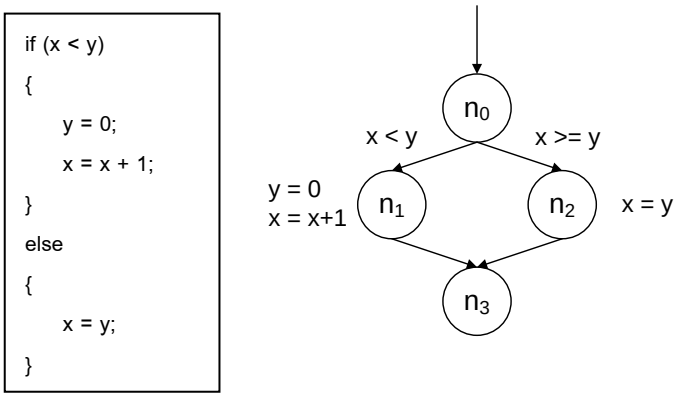
2.3 กราฟควบคุมการไหล (Control Flow Graph)

เป็นกราฟที่ใช้ในการวิเคราะห์การทำงานของโปรแกรมคอมพิวเตอร์ เป็นเครื่องมือที่ช่วยในการทดสอบการทำงานของซอฟต์แวร์ แสดงให้เห็นถึงโครงสร้างการทำงานตามเงื่อนไขต่างๆ ของซอฟต์แวร์ สามารถสร้างได้จากรหัสต้นฉบับ (Source Code) กราฟควบคุมการไหลจะประกอบไปด้วยโหนด (Node) ที่เชื่อมต่อกันด้วยเส้นเชื่อม (Edge) โดยที่แต่ละโหนดคือตัวแทนของคำสั่งในการทำงานของโปรแกรมหรือกลุ่มคำสั่งที่ทำงานเป็นลำดับ

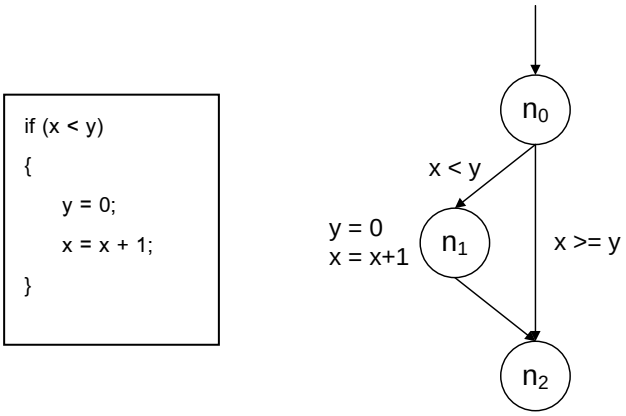
ตัวอย่างการสร้างกราฟควบคุมการไหลจากรหัสต้นฉบับในรูปแบบโครงสร้างแบบต่างๆ ตามหลักการที่ได้นำเสนอโดย Amman และ Offutt [8] แสดงดังภาพที่ 2-9 ถึง 2-14



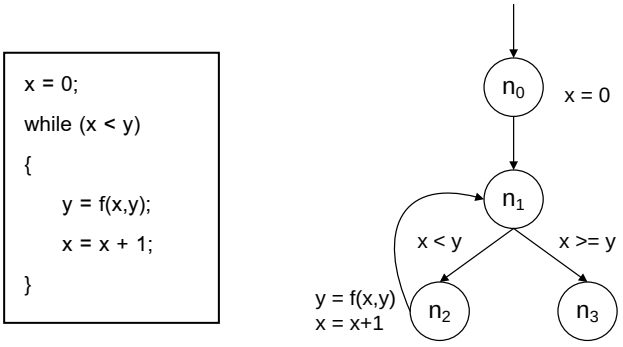
ภาพที่ 2-9 กราฟควบคุมการไหลสำหรับโครงสร้าง Sequence



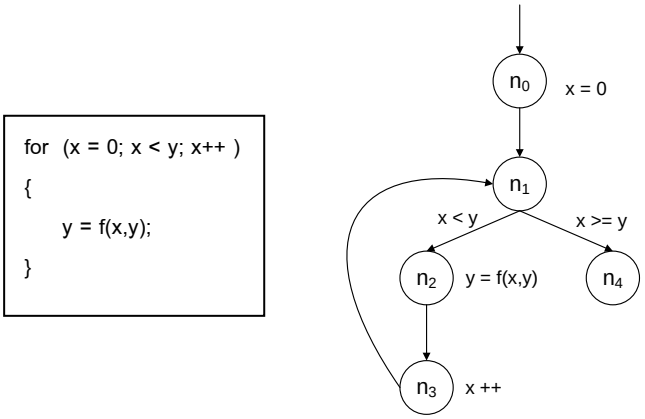
ภาพที่ 2-10 กราฟควบคุมการไหลสำหรับโครงสร้าง If – Else



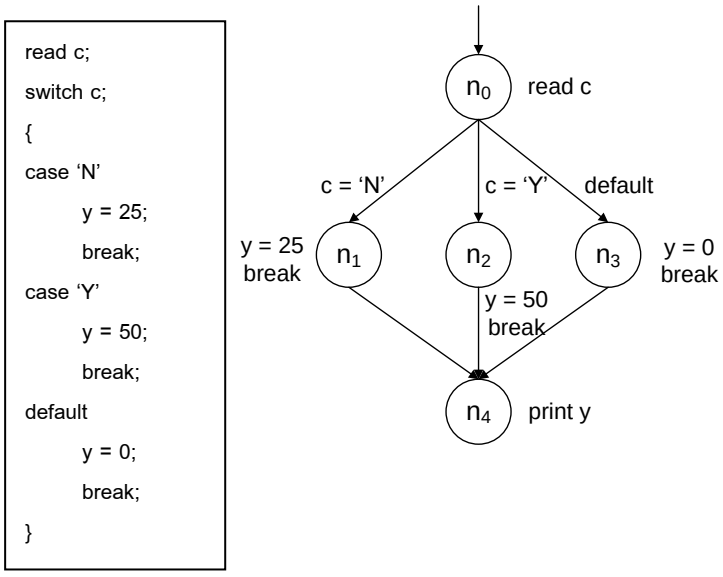
ภาพที่ 2-11 กราฟควบคุมการไหลสำหรับโครงสร้าง If



ภาพที่ 2-12 กราฟควบคุมการไหลสำหรับโครงสร้าง While Loop



ภาพที่ 2-13 กราฟควบคุมการไหลสำหรับโครงสร้าง For Loop



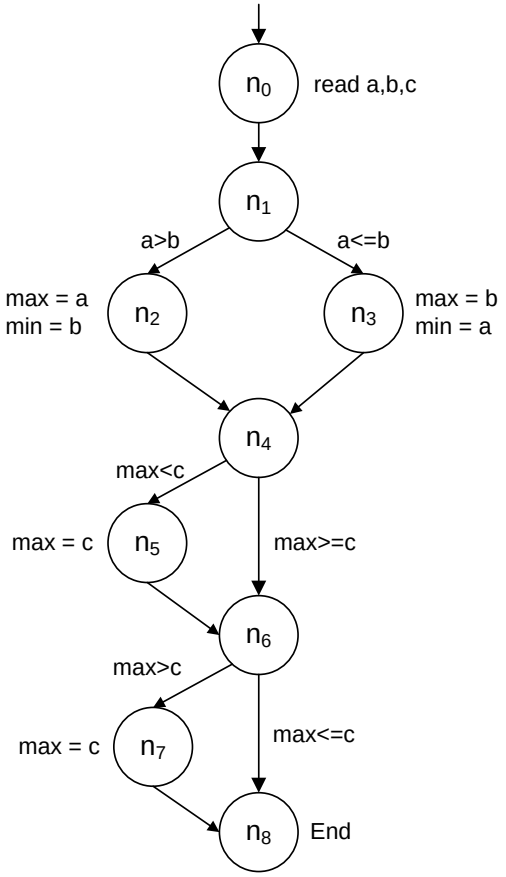
ภาพที่ 2-14 กราฟควบคุมการไหลสำหรับโครงสร้าง Select Case

กราฟควบคุมการไหลสำหรับโครงสร้างโปรแกรมในรูปแบบต่างๆ สามารถพิจารณาได้จากรหัสต้นฉบับของโปรแกรมนั้น ซึ่งกราฟควบคุมการไหลที่ได้นั้นสามารถนำมาสร้างเส้นทางสำหรับการทดสอบซอฟต์แวร์ให้ครอบคลุมเส้นทางที่เป็นไปได้

ตัวอย่าง เมื่อพิจารณาโครงสร้างของรหัสโปรแกรมคำนวณหาค่าสูงสุดต่ำสุดดังภาพที่ 2-15 สามารถเขียนกราฟควบคุมการไหลได้ดังภาพที่ 2-16

```
read  a,b,c
if a > b
{
        max = a : min = b
}
else
{
        max = b : min = a
}
if max < c
{
        max = c
}
if max > c
{
        min = c
}
```

ภาพที่ 2-15 รหัสต้นฉบับโปรแกรมคำนวณหาค่าสูงสุดต่ำสุด



ภาพที่ 2-16 กราฟควบคุมการไหลโปรแกรมคำนวณหาค่าสูงสุดต่ำสุด

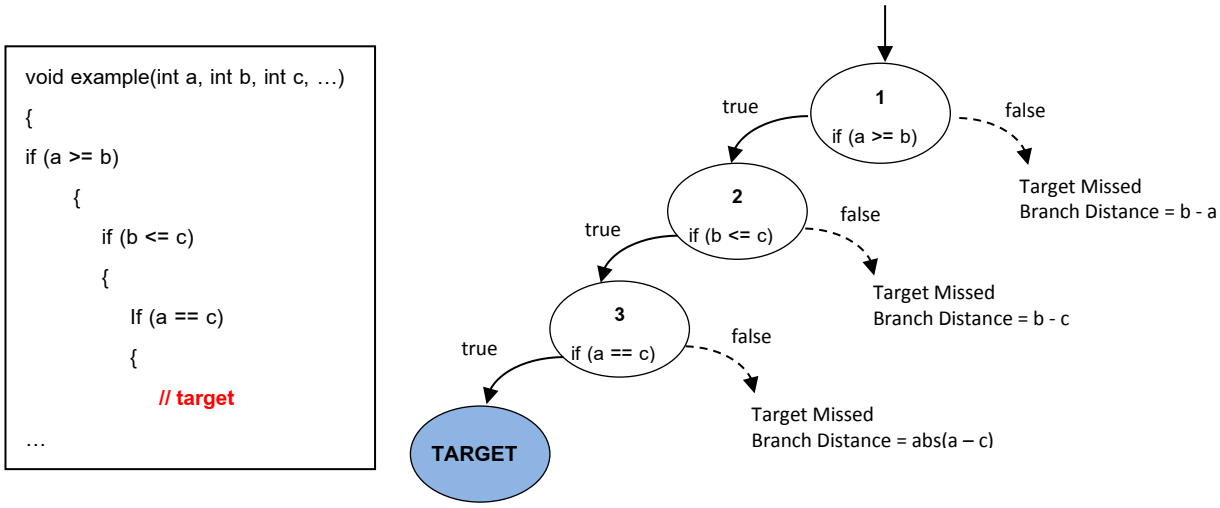
2.4 Branch Distance Calculation

Branch Distance Calculation เป็นวิธีการคำนวณหาระยะห่างระหว่างเส้นทาง 2 เส้นทาง โดยจะพิจารณาตามประเภทการตัดสินใจ (Decision Type) และใช้วิธีการคำนวณตามตารางที่ 2-2

ตารางที่ 2-2 Korel's distance function [2]

Decision Type	Branch Distance
a < b	a - b
a <= b	a - b
a > b	b - a
a >= b	b - a
a == b	abs(a-b)
a <> b	abs(a-b)
a && b	a + b
a || b	min(a , b)

สำหรับในวิทยานิพนธ์นี้ใช้การคำนวณ Branch Distance เพื่อคัดเลือกชุดข้อมูลที่มีความเหมาะสม โดยจะคำนวณค่า Branch Distance สำหรับแต่ละชุดข้อมูลแล้วนำมาพิจารณาเพื่อสร้างกลุ่มประชากรใหม่เพื่อเข้าสู่กระบวนการวิธีเชิงพันธุกรรม



ภาพที่ 2-17 ตัวอย่างการคำนวณหา Branch Distance

ภาพที่ 2-17 แสดงตัวอย่างการคำนวณหา Branch Distance ซึ่งจะคำนวณค่า Branch Distance ระหว่างเส้นทางหนึ่งซึ่งเป็นเส้นทางที่กำลังพิจารณา (Traversal Path) และเส้นทางเป้าหมาย (Target Path) จากตัวอย่าง ในกราฟควบคุมการไหลมี Decision Type จำนวน 3 Decision ได้แก่โหนด 1, 2 และ 3 ดังนั้น ในกรณีที่เส้นทางที่กำลังพิจารณาไม่ตรงกับเส้นทางเป้าหมายจะต้องมีการคำนวณค่า Branch Distance ให้สำหรับโหนดที่เป็น Decision Type ดังตารางที่ 2-3

ตารางที่ 2-3 แสดงการคำนวณค่า Branch Distance สำหรับกราฟควบคุมการไหลในภาพที่ 2-17

Node	Decision	Branch Distance
1	a >= b	b - a
2	a <= b	a - b
3	a == b	abs(a-b)

2.5 การทดสอบแบบกลายพันธุ์ (Mutation Testing)

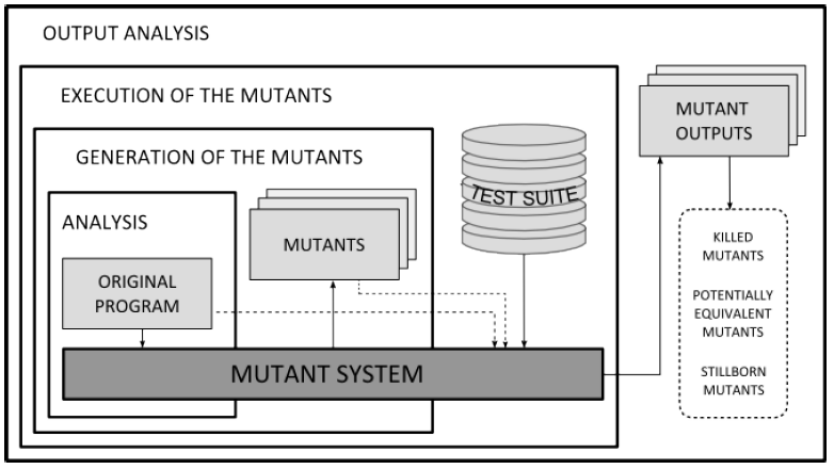
การทดสอบแบบกลายพันธุ์ เป็นการทดสอบโดยการปรับเปลี่ยนโปรแกรมที่ใช้ในการทดสอบให้ต่างไปจากโปรแกรมต้นฉบับ (Original Program) โดยในการปรับเปลี่ยนแต่ละครั้งจะทำเพียง 1 ตำแหน่งเท่านั้น โปรแกรมที่ได้จากการปรับเปลี่ยนเรียกว่าโปรแกรมกลายพันธุ์ (Mutant Program)

ภาพที่ 2-18 แสดงกระบวนการทำงานของการทดสอบแบบกลายพันธุ์ ซึ่งนิยมนำมาใช้เพื่อพิจารณาประสิทธิภาพของกรณีทดสอบ โดยนำชุดกรณีทดสอบ (Test Suit) ชุดเดียวกันมาดำเนินการปฏิบัติกับโปรแกรมต้นฉบับ (Original Program) และโปรแกรมกลายพันธุ์ (Mutant Program) เพื่อเปรียบเทียบผลลัพธ์ หากผลลัพธ์ที่ได้จากโปรแกรมกลายพันธุ์ (Mutant Output) แตกต่างจากผลลัพธ์ที่ได้จากโปรแกรมต้นฉบับจะเรียกโปรแกรมกลายพันธุ์นั้นว่าโปรแกรมกลายพันธุ์ที่ถูกฆ่า (Killed Mutant) แสดงว่ากรณีทดสอบที่ใช้ในการดำเนินการปฏิบัติกับโปรแกรมกลายพันธุ์สามารถตรวจจับข้อผิดพลาดจากโปรแกรมกลายพันธุ์นั้นได้ ส่วนในกรณีที่ผลลัพธ์ที่ได้จากโปรแกรมกลายพันธุ์ไม่มีความแตกต่างกับผลลัพธ์ที่ได้จากโปรแกรมต้นฉบับ จะเรียกโปรแกรมกลายพันธุ์ในลักษณะนี้ว่าโปรแกรมกลายพันธุ์ที่ยังมีชีวิตอยู่ (Live Mutant) สำหรับโปรแกรมกลายพันธุ์ที่ไม่สามารถหากรณีทดสอบใดๆ มาดำเนินการปฏิบัติแล้วให้ผลลัพธ์ที่แตกต่างจากโปรแกรมต้นฉบับได้ จะเรียกโปรแกรมกลายพันธุ์ในลักษณะนี้ว่าโปรแกรมกลายพันธุ์ที่สมมูล (Equivalent Mutant) และอัตราส่วนของโปรแกรมกลายพันธุ์ที่ถูกฆ่า ต่อจำนวนโปรแกรมกลายพันธุ์ทั้งหมด ลบกับโปรแกรม

กลายพันธุ์ที่สมมูล จะแสดงถึงประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นซึ่งจะแสดงเป็นคะแนนการกลายพันธุ์ (Mutation Score)

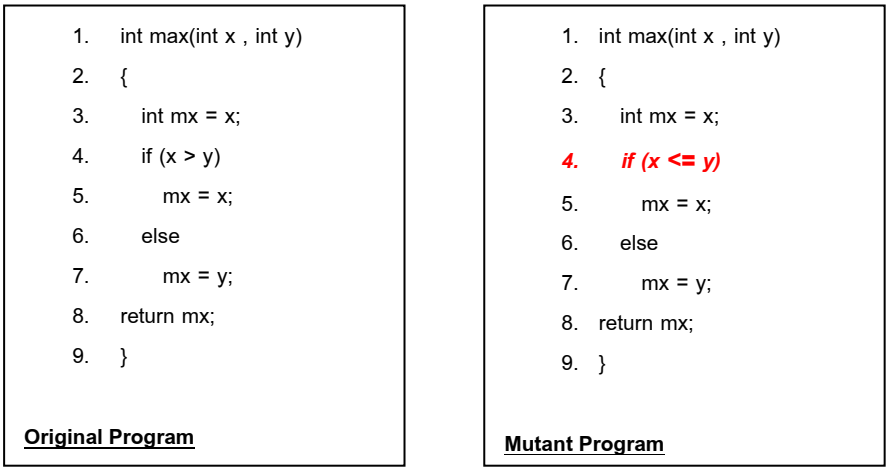
$$\text{Mutation Score} = \frac{\text{Killed Mutants}}{\text{(Total number of Mutants – Equivalent Mutants)}}$$

หากคะแนนการกลายพันธุ์มีค่าเข้าใกล้ 1 หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพสูง และหากคะแนนการกลายพันธุ์มีค่าเข้าใกล้ศูนย์ หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพค่อนข้างต่ำ



ภาพที่ 2-18 Mutation Testing Process [13]

จากภาพที่ 2-19 โปรแกรมกลายพันธุ์ (Mutant Program) ถูกปรับเปลี่ยนให้ต่างจากโปรแกรมต้นฉบับ (Original Program) โดยการเปลี่ยนคำสั่งในบรรทัดที่ 4 จากเครื่องหมายมากกว่า (>) เป็นเครื่องหมายน้อยกว่าหรือเท่ากับ (<=) ซึ่งการปรับเปลี่ยนจะกระทำโดยอาศัยตัวดำเนินการที่เรียกว่าตัวดำเนินการการกลายพันธุ์ (Mutation Operator) คือรูปแบบการดำเนินการเพื่อจะทำให้โปรแกรมที่เกิดค่าที่ผิดพลาดขึ้น ดำเนินการการกลายพันธุ์จะประกอบไปด้วยตัวดำเนินการ (Operator) หลายรูปแบบที่แตกต่างกันสำหรับแต่ละภาษา ตัวอย่างรูปแบบต่างๆ ของตัวดำเนินการการกลายพันธุ์สำหรับภาษา C# และภาษา Visual Basic แสดงดังตารางที่ 2-4 เช่น AORS (Arithmetic Operations Replacement) เป็นการแทนตัวดำเนินการทางคณิตศาสตร์ด้วยค่าที่แตกต่างไป เช่นแทนเครื่องหมายบวก (+) ด้วยเครื่องหมายลบ (-) สำหรับ ROR (Relational Operations Replacement) เป็นการแทนตัวดำเนินการเชิงสัมพันธ์ ด้วยค่าที่แตกต่างกันออกไป เช่นแทนความสัมพันธ์ AND ด้วยความสัมพันธ์ OR เป็นต้น



ภาพที่ 2-19 ตัวอย่างโปรแกรมต้นฉบับและโปรแกรมกลายพันธุ์

ตารางที่ 2-4 Mutation Operators [16]

Operator Type	Name	Description
Arithmetic	AOR	arithmetic operations replacement
	AOI	arithmetic operations insertion
	AOD	arithmetic operations deletion
Relational	ROR	relational operations replacement
Conditional	COR	conditional operations replacement
	COI	conditional operations insertion
	COD	conditional operations deletion
Logical	LOR	logical operations replacement
	LOI	logical operations insertion
	LOD	logical operations deletion
Shift	SOR	shift operations replacement
	SOIA	shift operations insertion
	SODA	shift operations deletion
Replacement	PR	parameter replacement
	LVR	local variable replacement

จากตารางที่ 2-4 งานวิจัยนี้เลือกใช้ตัวดำเนินการการกลายพันธุ์ (Mutation Operator) ในรูปแบบของการแทนที่ (Replacement) โดยใช้ประเภทของตัวดำเนินการในรูปแบบต่างๆ ซึ่งพิจารณาจากลักษณะโครงสร้างของโปรแกรมแต่ละโปรแกรมไปแทนที่โปรแกรมต้นฉบับเพื่อสร้างโปรแกรมกลายพันธุ์

2.6 งานวิจัยที่เกี่ยวข้อง

2.6.1 งานวิจัยเรื่อง Application of Genetic Algorithm in Software Testing โดย Praveen Ranjan Srivastava และ Tai-hoon Kim [14]

งานวิจัยนี้ ได้นำเสนอวิธีการประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรมในกระบวนการทดสอบซอฟต์แวร์ เพื่อเพิ่มประสิทธิภาพในการทดสอบซอฟต์แวร์ และระบุเส้นทางที่ทำให้เกิดความผิดพลาด โดยใช้วิธีการสร้างกราฟควบคุมการไหล (Control Flow Graph, CFG) และกำหนดค่าน้ำหนัก (Weight) ให้กับเส้นทางต่างๆในกราฟควบคุมการไหล เส้นทางที่มีผลรวมค่าน้ำหนักมากจะเป็นเส้นทางที่ทำให้เกิดความผิดพลาด ในส่วนของการดำเนินการทางพันธุกรรม งานวิจัยนี้ใช้การไขว้เปลี่ยนแบบจุดเดียว (Single-point Crossover) ที่มีอัตราส่วนของการไขว้เปลี่ยน เท่ากับ 0.8 และมีค่าคงที่การกลายพันธุ์เท่ากับ 0.3 มาดำเนินการกับโครโมโซมเพื่อหาชุดของข้อมูลที่ใช้ในการทดสอบซอฟต์แวร์ที่มีประสิทธิภาพดีที่สุด ในส่วนของการประเมินค่าความเหมาะสม งานวิจัยนี้ใช้วิธีการคำนวณหาค่าน้ำหนักของแต่ละโหนดของกราฟควบคุมการไหลสำหรับทุกๆ เส้นทางที่ใช้ในการทดสอบ งานวิจัยชิ้นนี้พบว่า การนำขั้นตอนวิธีเชิงพันธุกรรมมาประยุกต์ใช้สำหรับกระบวนการทดสอบซอฟต์แวร์ สามารถช่วยในการหาข้อผิดพลาดที่อาจเกิดขึ้นจาก การทำงานของซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

2.6.2 งานวิจัยเรื่อง Automatic Software Structural Testing by Using Evolutionary Algorithms for Test Data Generations โดย Maha Alzabidi , Ajay Kumar และ A.D. Shaligram [15]

งานวิจัยนี้ ได้นำเสนอวิธีการสร้างกรณีทดสอบแบบอัตโนมัติโดยใช้ขั้นตอนวิธีวิวัฒนาการ (Evolutionary Algorithms) ร่วมกับการทดสอบการครอบคลุมเส้นทางการทำงานของโปรแกรม (Path Coverage) โดยใช้กราฟควบคุมการไหล (Control Flow Graph, CFG) สำหรับนำเสนอเส้นทางสำหรับใช้ในการทดสอบโปรแกรม ในส่วนของการดำเนินการทางวิวัฒนาการงานวิจัยชิ้นนี้ใช้วิธีการประเมินค่าความเหมาะสมของกรณีทดสอบที่ได้จาก Shifted-Modified-Similarity (SMS) ซึ่งเป็นการคำนวณค่าระยะทางระหว่างเส้นทางที่ใช้ในการทดสอบ เพื่อนำมาประเมินค่าความเหมาะสมให้กับประชากรรุ่นต่อไป และในส่วนของการดำเนินการทางวิวัฒนาการใช้การไขว้เปลี่ยน

(Crossover) และการกลายพันธุ์ (Mutation) จากผลการวิจัยนี้พบว่าการไขว้เปลี่ยนแบบ 2 จุด (Double-point Crossover) ทำให้เกิดประชากรใหม่ที่มีประสิทธิภาพมากกว่าการไขว้เปลี่ยนแบบจุดเดียว (Single-point Crossover) ส่วนการกลายพันธุ์เมื่อใช้อัตราความน่าจะเป็นในการกลายพันธุ์ที่มีค่าน้อย (งานวิจัยชิ้นนี้ใช้อัตราความน่าจะเป็นในการกลายพันธุ์เท่ากับ 0.005 :ซึ่งทำให้เกิดโอกาสในการกลายพันธุ์ที่น้อยมาก) จะทำให้เกิดประชากรที่มีประสิทธิภาพดีกว่าการใช้อัตราความน่าจะเป็นในการกลายพันธุ์ที่มีค่าสูง และงานวิจัยนี้ยังได้ทำการเปรียบเทียบผลที่ได้จากการสร้างกรณีทดสอบโดยใช้ทฤษฎีวิวัฒนาการกับการสร้างกรณีทดสอบแบบสุ่มกับโปรแกรมตัวอย่าง ได้แก่ Tri-Class และ Max-Min ผลที่ได้พบว่าการสร้างกรณีทดสอบโดยใช้ทฤษฎีวิวัฒนาการสามารถสร้างกรณีทดสอบได้ดีกว่าการสร้างกรณีทดสอบแบบสุ่ม

2.6.3 งานวิจัยเรื่อง Testing Object-Oriented Code Through a Specifications-Based Mutation Engine โดย Pantelis Stylianos Yiasemis และ Andreas S. Andreou [16]

งานวิจัยนี้ นำเสนอการใช้การทดสอบแบบกลายพันธุ์ (Mutation Testing) ในการตรวจจับข้อผิดพลาดที่เกิดจากโปรแกรมที่เขียนขึ้นจากการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ในภาษา C# และภาษา Visual Basic โดยส่วนใหญ่การทดสอบแบบกลายพันธุ์ จะถูกนำมาใช้ในการประเมินประสิทธิภาพของกรณีทดสอบ งานวิจัยนี้ได้พัฒนาเครื่องมือสำหรับใช้ในการตรวจจับข้อผิดพลาดที่อาจเกิดจากการเขียนโปรแกรมโดยใช้การทดสอบแบบกลายพันธุ์ ซึ่งเครื่องมือที่พัฒนาขึ้นจะมีในส่วนของการกำหนดตัวดำเนินการการกลายพันธุ์ (Mutation Operator) โดยให้ผู้ใช้เป็นผู้กำหนดเพื่อนำมาใช้ประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้น ลักษณะการดำเนินการของการใช้การทดสอบแบบกลายพันธุ์ คือจะปรับเปลี่ยนโปรแกรมที่ใช้ในการทดสอบให้ต่างไปจากโปรแกรมต้นฉบับไปตามประเภทของตัวดำเนินการการกลายพันธุ์ และในงานวิจัยชิ้นนี้ยังได้ทำการเปรียบเทียบจำนวนของการทดสอบแบบกลายพันธุ์ ในการทดลองกับโปรแกรมที่เขียนด้วยภาษา C# และภาษา Visual Basic โดยงานวิจัยชิ้นนี้พบว่า จำนวนของการทดสอบแบบกลายพันธุ์ ที่ได้จากการทดสอบของโปรแกรมที่เขียนด้วยภาษาทั้งสองภาษาไม่มีความแตกต่างกัน และเครื่องมือที่นำเสนอสามารถที่จะค้นหาข้อผิดพลาดที่เกิดจากการเขียนโปรแกรมได้อย่างมีประสิทธิภาพ

จากการศึกษางานวิจัยที่เกี่ยวข้องทั้ง 3 ชิ้นนี้ทำให้ผู้วิจัยได้แนวทางในการทำวิทยานิพนธ์ฉบับนี้จากหลักการการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมจากงานวิจัยชิ้นที่ 1 ได้นำแนวคิดการทดสอบซอฟต์แวร์ในระดับการทดสอบการครอบคลุมเส้นทางการทำงานจากงานวิจัยชิ้นที่ 2 และได้แนวคิดสำหรับการประเมินประสิทธิภาพของกรณีทดสอบโดยใช้การทดสอบแบบกลายพันธุ์จากงานวิจัยชิ้นที่ 3

บทที่ 3

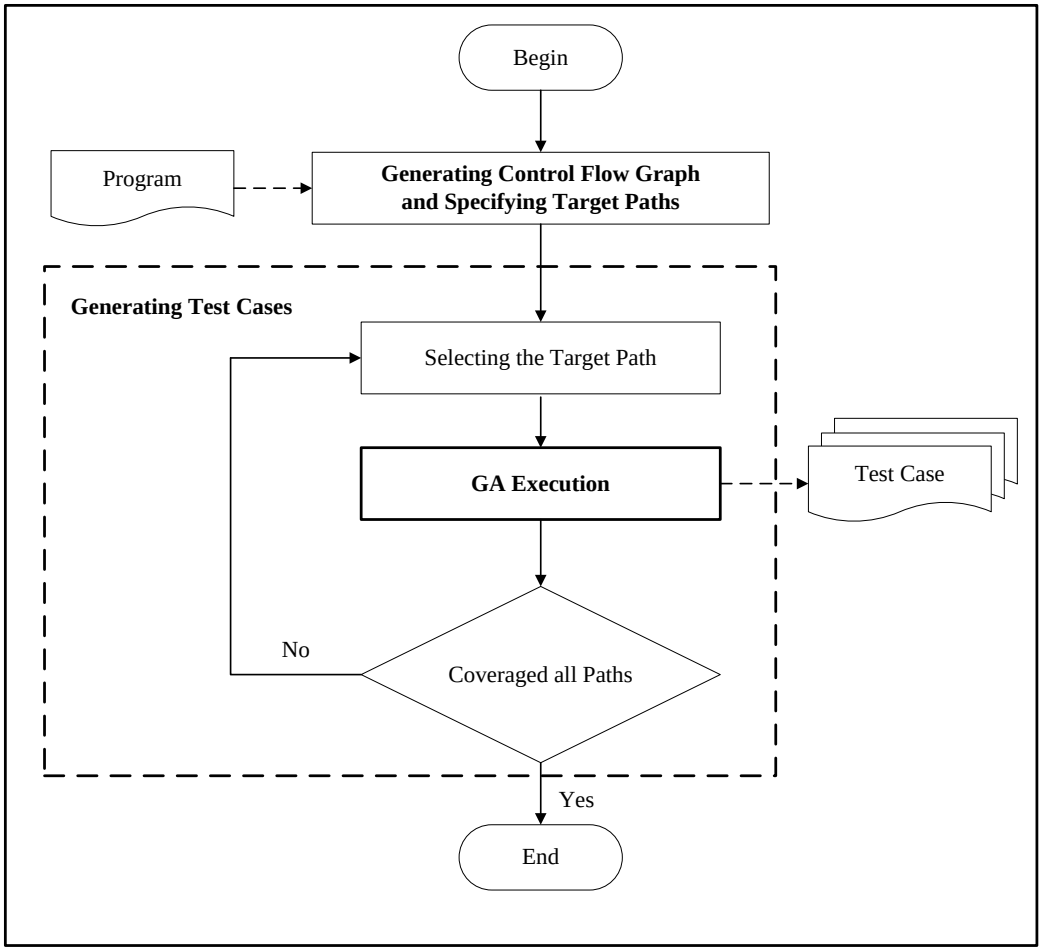
วิธีการดำเนินการวิจัย

3.1 การวิเคราะห์และออกแบบขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

ขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม แสดงดังภาพที่ 3-1 ประกอบด้วย 2 ขั้นตอนหลัก คือ

1) การสร้างกราฟควบคุมการไหลและระบุเส้นทางเป้าหมาย (Generating Control Flow Graph and Specifying Target Paths)

2) การสร้างกรณีทดสอบ (Generating Test Cases)



ภาพที่ 3-1 แผนภาพแสดงกรอบแนวคิดขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

3.2 การสร้างกราฟควบคุมการไหลและระบุเส้นทางเป้าหมาย (Generating Control Flow Graph and Specifying Target Paths)

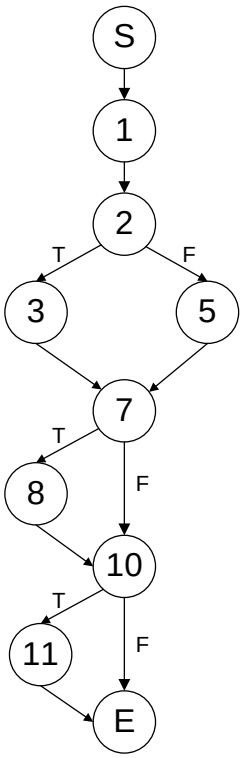
ขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมจะเริ่มจากการสร้างกราฟควบคุมการไหลของโปรแกรมที่ต้องการหากรณีทดสอบและทำการพิจารณาเส้นทางการทำงานของโปรแกรมทั้งหมดจากกราฟควบคุมการไหลเพื่อที่จะระบุเส้นทาง (Paths) สำหรับการสร้างกรณีทดสอบสำหรับแต่ละเส้นทาง ซึ่งเส้นทางที่ระบุจะต้องครอบคลุมทุกๆ เงื่อนไขการทำงานของโปรแกรม โดยขั้นตอนนี้จะประกอบด้วย 2 ขั้นตอนย่อย คือ ขั้นตอนการสร้างกราฟควบคุมการไหลและขั้นตอนการระบุเส้นทางเป้าหมาย

1) ขั้นตอนการสร้างกราฟควบคุมการไหล (Generating Control Flow Graph)

การสร้างกราฟควบคุมการไหลจะพิจารณาจากรหัสต้นฉบับ (Source Code) ของโปรแกรมที่ต้องการทดสอบ โดยกราฟควบคุมการไหลถูกสร้างขึ้นตามหลักการที่ได้นำเสนอโดย Amman และ Offutt [8] ภาพที่ 3-2 แสดงตัวอย่างรหัสต้นฉบับของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min) ที่ต้องการทดสอบ ภาพที่ 3-3 แสดงกราฟควบคุมการไหลของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min)

```
1    Dim a , b , c , max , min As Double
2    If a > b Then
3          max = a : min = b
4    Else
5          max = b : min = a
6    End If
7    If max < c Then
8          max = c
9    End If
10   If max > c Then
11         min = c
12   End If
```

ภาพที่ 3-2 ตัวอย่างรหัสต้นฉบับของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min)



ภาพที่ 3-3 กราฟควบคุมการไหลของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min)

2) ขั้นตอนการระบุเส้นทางเป้าหมาย (Specifying Target Paths)

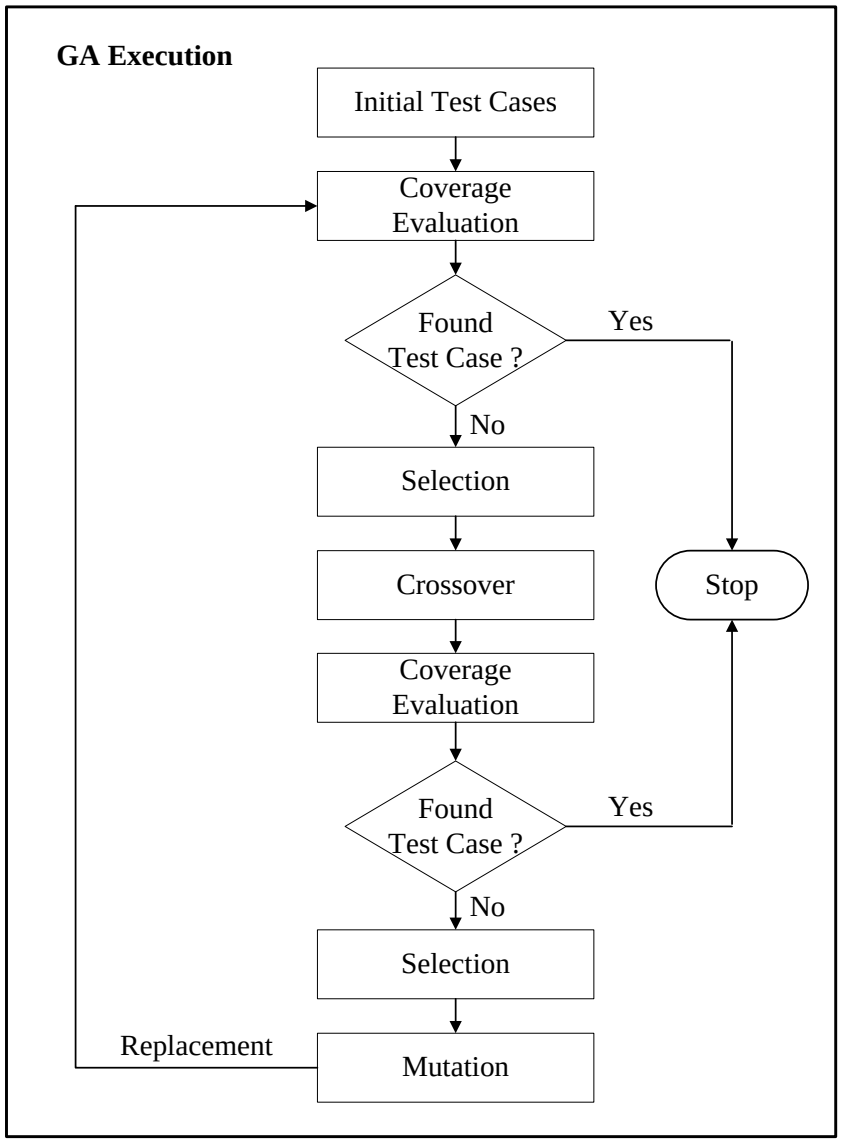
กราฟควบคุมการไหลจากขั้นตอนที่ 1 จะถูกนำมาใช้เพื่อระบุเส้นทางเป้าหมาย (Target Paths) ที่ใช้ในการหากรณีทดสอบสำหรับแต่ละเส้นทาง สำหรับเส้นทางเป้าหมายในงานวิจัยนี้จะพิจารณาในระดับการครอบคลุมเส้นทาง (Path Coverage - PC) [8] ซึ่งจะพิจารณาการทดสอบให้ครอบคลุมทุกๆ เส้นทางการทำงาน ตัวอย่างกราฟควบคุมการไหลในภาพที่ 3-3 สามารถระบุเส้นทางเป้าหมายสำหรับใช้ในการทดสอบได้ทั้งหมด 8 เส้นทาง ดังภาพที่ 3-4

Path 1	S	1	2T	3	7T	8	10T	11	E
Path 2	S	1	2T	3	7T	8	10F	E	
Path 3	S	1	2T	3	7F	10T	11	E	
Path 4	S	1	2T	3	7F	10F	E		
Path 5	S	1	2F	5	7T	8	10T	11	E
Path 6	S	1	2F	5	7T	8	10F	E	
Path 7	S	1	2F	5	7F	10T	11	E	
Path 8	S	1	2F	5	7F	10F	E		

ภาพที่ 3-4 เส้นทางเป้าหมายทั้งหมดของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min)

3.3 การสร้างกรณีทดสอบ (Generating Test Cases)

การสร้างกรณีทดสอบจะถูกดำเนินการโดยใช้หลักการขั้นตอนวิธีเชิงพันธุกรรมโดยเริ่มจากการเลือกเส้นทางเป้าหมายที่ต้องการพิจารณาเพื่อหากรณีทดสอบ (Select Considering Target Path) แล้วเข้าสู่กระบวนการทำงานของขั้นตอนปฏิบัติการวิธีเชิงพันธุกรรม (GA Execution) เพื่อค้นหากรณีทดสอบสำหรับเส้นทางเป้าหมายที่กำลังพิจารณา ภาพที่ 3-5 แสดงขั้นตอนปฏิบัติการขั้นตอนวิธีเชิงพันธุกรรม โดยขั้นตอนการเลือกเส้นทางเป้าหมายและการดำเนินการขั้นตอนปฏิบัติการขั้นตอนวิธีเชิงพันธุกรรม จะถูกกระทำซ้ำๆ จนกระทั่งครบทุกเส้นทางเป้าหมาย ในแต่ละรอบของการทำงานซ้ำ เส้นทางเป้าหมายที่ถูกเลือกจะเรียกว่า เส้นทางเป้าหมายที่กำลังพิจารณา (Traversal Path)



ภาพที่ 3-5 GA Execution

จากภาพที่ 3-5 แสดงขั้นตอนการประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรมในกระบวนการสร้างกรณีทดสอบซึ่งมีกระบวนการต่างๆ ดังนี้

ขั้นตอนที่ 1 การสร้างกลุ่มกรณีทดสอบต้นกำเนิด (Initial Test Cases)

กลุ่มกรณีทดสอบต้นกำเนิด (กลุ่มประชากรต้นกำเนิด) จะถูกสร้างขึ้นเพื่อเป็นจุดเริ่มต้นสำหรับการปรับเปลี่ยนตามหลักการถ่ายทอดลักษณะทางพันธุกรรมให้กับกลุ่มกรณีทดสอบรุ่นต่อไป โดยกลุ่มกรณีทดสอบต้นกำเนิดจะถูกสร้างขึ้นโดยใช้วิธีการแบ่งส่วนที่สมดุล (Equivalence Partitioning) [8] โดยข้อมูลนำเข้าของโปรแกรมที่ต้องการทดสอบแต่ละตัวจะถูกแบ่งส่วนออกเป็นกลุ่มๆ กรณีทดสอบต้นกำเนิดหนึ่งกรณี ได้จากการนำส่วนใดส่วนหนึ่งของข้อมูลนำเข้าทุกตัวมารวมกัน ดังนั้นหนึ่งกรณีทดสอบต้นกำเนิด (กลุ่มประชากรต้นกำเนิด) หมายถึงหนึ่งชุด

ข้อมูลทดสอบ ซึ่งจำนวนตัวแปรของหนึ่งชุดข้อมูลทดสอบเท่ากับจำนวนโครโมโซมของหนึ่งประชากร โดยที่โครโมโซมของหนึ่งประชากร (ข้อมูลนำเข้า) อาจมีชนิดของข้อมูล (Data Type) ที่แตกต่างกัน ซึ่งการแบ่งส่วนที่สมดุลสำหรับแต่ละชนิดข้อมูลก็แตกต่างกันด้วย ตัวอย่างเช่น

1) การแบ่งส่วนที่สมดุลสำหรับข้อมูลประเภทจำนวนเต็ม (Integer) และจำนวนทศนิยม (Double) สามารถแบ่งส่วนของข้อมูลออกเป็น 3 กลุ่ม คือ มากกว่า , น้อยกว่า และเท่ากับศูนย์
2) การแบ่งส่วนที่สมดุลสำหรับข้อมูลประเภทรหัสอักขระ (Character) สามารถแบ่งส่วนของข้อมูลออกเป็น 2 กลุ่ม คือ รหัสอักขระ และค่าว่าง
3) การแบ่งส่วนที่สมดุลสำหรับข้อมูลประเภทข้อความ (String) สามารถแบ่งส่วนของข้อมูลออกเป็น 2 กลุ่ม คือ ข้อความ และค่าว่าง
4) การแบ่งส่วนที่สมดุลสำหรับข้อมูลประเภทค่าความจริง (Boolean) สามารถแบ่งส่วนของข้อมูลออกเป็น 2 กลุ่ม คือ ค่าจริง (True) และค่าเท็จ (False)

โดยการกำหนดกลุ่มของกรณีทดสอบเริ่มต้น จะกำหนดตามหลักการ Each Choice (EC) ที่ถูกนำเสนอโดย Amman และ Offutt [8] โดยหลักการนี้กำหนดให้แต่ละส่วนที่ถูกแบ่งจะต้องถูกใช้อย่างน้อย 1 ครั้งในการทดสอบ

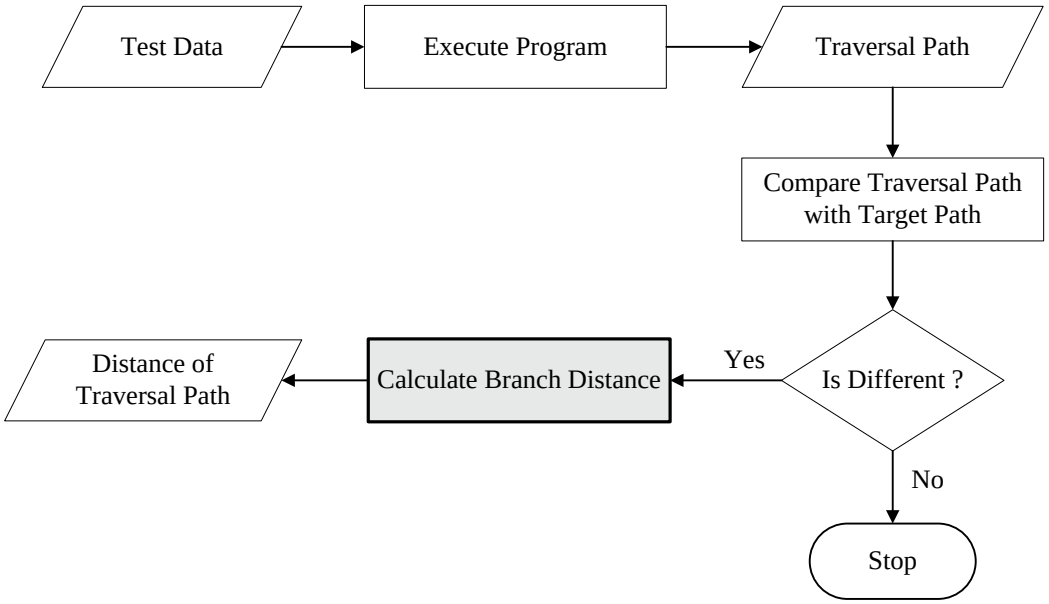
ตัวอย่างโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min) ซึ่งมีข้อมูลนำเข้าคือ a, b และ c โดยทั้งสามเป็นข้อมูลประเภทจำนวนทศนิยม (Double) จากหลักการแบ่งส่วนที่สมดุลสามารถสามารถแบ่งส่วน (Partition) ข้อมูลนำเข้าแต่ละตัวออกเป็น 3 กลุ่ม คือ มากกว่า, น้อยกว่า และเท่ากับศูนย์ เมื่อกำหนดกลุ่มของชุดข้อมูลเริ่มต้นตามหลักการ Each Choice (EC) จะได้ชุดข้อมูลที่ใช้สำหรับโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min) ประกอบด้วย 3 ชุดข้อมูล คือ {a < 0.0 , b < 0.0 , c < 0.0}, {a = 0.0 , b = 0.0 , c = 0.0} และ {a > 0.0 , b > 0.0 , c > 0.0} จากนั้นทำการสุ่มค่าของข้อมูลที่สอดคล้องกับช่วงของข้อมูลที่กำหนดไว้ ตัวอย่างการสุ่มค่าข้อมูลแสดงดังตารางที่ 3-1

ตารางที่ 3-1 กลุ่มกรณีทดสอบต้นกำเนิด สำหรับการทดสอบโปรแกรมหาค่าสูงสุดและต่ำสุด

ตัวแปร / ชุดข้อมูล	a	b	c
1	-2.2	-1.4	-4.5
2	0.0	0.0	0.0
3	3.7	2.4	5.3

ขั้นตอนที่ 2 การประเมินค่าความครอบคลุม (Coverage Evaluation)

การประเมินค่าความครอบคลุมจะพิจารณาจากการประเมินค่าความครอบคลุมเส้นทางเป้าหมายของกลุ่มกรณีทดสอบ สำหรับงานวิจัยนี้การประเมินค่าความครอบคลุมใช้วิธีการคำนวณหาค่า Distance [2] ของคำสั่งที่เป็นเงื่อนไขในการทำงานของโปรแกรม ตามทฤษฎี Korel's distance function ดังแสดงในตารางที่ 2-2 โดยขั้นตอนการประเมินค่าความครอบคลุม แสดงดังภาพที่ 3-6



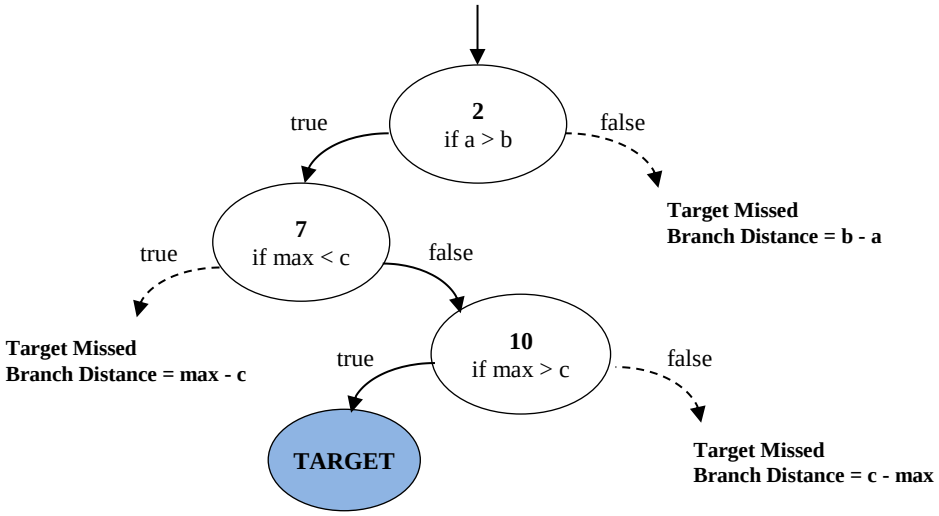
ภาพที่ 3-6 ขั้นตอนการประเมินค่าความครอบคลุม

จากภาพที่ 3-6 ขั้นตอนจะเริ่มต้นจากการนำชุดข้อมูลทดสอบ (Test Data) ที่ได้จากขั้นตอนก่อนหน้า มาปฏิบัติกับโปรแกรมที่ต้องการทดสอบ (Execute Program) โดยชุดข้อมูลแต่ละชุดจะให้เส้นทางการทำงาน (Traversal Path) ของข้อมูลชุดนั้น จากนั้นนำเส้นทางการทำงานมาเปรียบเทียบกับเส้นทางเป้าหมาย (Target Path) โดยจะทำการเปรียบเทียบเฉพาะส่วนของโปรแกรมที่เป็นคำสั่งเงื่อนไขการตัดสินใจ (Decision) ในกรณีที่ส่วนที่เป็นคำสั่งเงื่อนไขการตัดสินใจของเส้นทางการทำงานไม่แตกต่างกับเส้นทางเป้าหมายที่กำลังพิจารณา จะไม่ทำการคำนวณค่า Branch Distance แต่ในกรณีที่ส่วนที่เป็นคำสั่งเงื่อนไขการตัดสินใจของเส้นทางการทำงาน แตกต่างจากเส้นทางเป้าหมายที่กำลังพิจารณา จะทำการคำนวณค่า Branch Distance ตามประเภทของคำสั่งที่เป็นเงื่อนไขตามทฤษฎีของ Korel's distance function ดังแสดงในตารางที่ 2-2

ตัวอย่างเช่น จากภาพที่ 3-2 รหัสต้นฉบับของโปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min) เมื่อกำหนดให้เส้นทางที่ 3 เป็นเส้นทางเป้าหมายสำหรับการสร้างกรณีทดสอบคือ เส้นทาง

Path 3 : S , 1 , 2T , 3 , 7F , 10T , 11 , E

สามารถเขียนกราฟควบคุมการไหลแสดงเส้นทางเป้าหมายที่ต้องการทดสอบได้ ดังภาพที่ 3-7



ภาพที่ 3-7 กราฟควบคุมการไหลแสดงเส้นทางเป้าหมาย

เงื่อนไขการทำงานสำหรับเส้นทางเป้าหมาย Path 3 ทั้งหมดแสดงดังตารางที่ 3-2

ตารางที่ 3-2 Predicate Path ที่ต้องพิจารณาจากภาพที่ 3-7

Predicate Path	Predicate	Branch Distance
2T	a > b	b - a
7F	max < c	max - c
10T	max > c	c - max

ตัวอย่างชุดข้อมูลที่ได้จากการสุ่มกรณีทดสอบต้นกำเนิดในตารางที่ 3-1 ถูกนำมาพิจารณากับเส้นทางเป้าหมายที่กำหนด ดังนี้

ชุดข้อมูลที่ 1 : a = -2.2 , b = -1.4 , c = -4.5

เส้นทางสำหรับชุดข้อมูลที่ 1 คือ TR(1) = S , 1 , 2F , 5 , 7F , 10T , 11 , E

เมื่อพิจารณาพบว่า Predicate ที่ต้องพิจารณาคือ 2F

Distance($TR(1)_1$) = b – a = (-1.4) – (-2.2) = 0.8

ดังนั้น Distance(TR(1)) = Distance($TR(1)_1$) = 0.8

ชุดข้อมูลที่ 2 : a = 0.0 , b = 0.0 , c = 0.0

เส้นทางสำหรับชุดข้อมูลที่ 2 คือ TR(2) = S , 1 , 2F , 5 , 7F , 10F , E

เมื่อพิจารณาพบว่า Predicate ที่ต้องพิจารณาคือ 2F และ 10F

Distance($TR(2)_1$) = b – a = **0.0** - **0.0** = **0.0**

Distance($TR(2)_2$) = c – max = **0.0** - **0.0** = **0.0**

ดังนั้น Distance(TR(2)) = Distance($TR(2)_1$) + Distance($TR(2)_2$) = 0.0 + 0.0 = 0.0

ชุดข้อมูลที่ 3 : a = 3.7 , b = 2.4 , c = 5.3

เส้นทางสำหรับชุดข้อมูลที่ 3 คือ TR(3) = S , 1 , 2T , 3 , 7T , 8 , 10F , E

เมื่อพิจารณาพบว่า Predicate ที่ต้องพิจารณาคือ 7T และ 10F

Distance($TR(3)_1$) = max – c = 3.7 – 5.3 = -1.6

Distance($TR(3)_2$) = c – max = 5.3 – 5.3 = 0.0

ดังนั้น Distance(TR(3)) = Distance($TR(3)_1$)+Distance($TR(3)_2$) = (-1.6)+0.0 = -1.6

ค่าความครอบคลุมที่ได้จากการพิจารณาคำสั่งที่เป็นเงื่อนไขการตัดสินใจของแต่ละชุดข้อมูลจะถูกนำไปพิจารณาเพื่อเข้าสู่กระบวนการคัดเลือกทางพันธุกรรมต่อไปเพื่อสร้างเป็นชุดข้อมูลหรือกลุ่มกรณีทดสอบเพื่อทำการค้นหาชุดของข้อมูลที่เป็นกรณีทดสอบที่มีความเหมาะสมกับเส้นทางเป้าหมายที่กำลังพิจารณา โดยกรณีที่ชุดข้อมูลใดให้เส้นทางเหมือนกับเส้นทางเป้าหมายที่กำลังพิจารณาแสดงว่าชุดข้อมูลนั้นเป็นชุดข้อมูลที่ครอบคลุมเส้นทางเป้าหมายที่กำลังพิจารณา ในกรณีนี้จะเป็นกรณีที่พบชุดข้อมูลทดสอบ (เงื่อนไข Found Test Case เป็นจริง) จะมีผลให้ขั้นตอนปฏิบัติการขั้นตอนวิธีเชิงพันธุกรรมยุติ แต่ในกรณีที่ไม่สามารถหากรณีทดสอบ ขั้นตอนปฏิบัติการขั้นตอนวิธีเชิงพันธุกรรมจะยุติเมื่อครบจำนวนรอบที่กำหนดไว้

ขั้นตอนที่ 3 การคัดเลือก (Selection)

จากการดำเนินการในขั้นตอนที่ 2 จะได้ค่า Branch Distance สำหรับแต่ละชุดข้อมูล ซึ่งการคัดเลือกจะกระทำการคัดเลือกชุดข้อมูลเพื่อนำมาเป็นประชากรรุ่นพ่อแม่ โดยเลือกชุดข้อมูลที่ให้ค่าความเหมาะสมน้อย ซึ่งหมายถึงชุดข้อมูลที่ให้ค่า Branch Distance น้อยที่สุด

จากตัวอย่างข้างต้นเมื่อเรียงลำดับค่า Distance จากน้อยไปมากแล้ว จะทำให้ได้ชุดของข้อมูลใหม่ ดังนี้

ตารางที่ 3-3 ตัวอย่างการเรียงลำดับค่า Branch Distance ใหม่จากน้อยไปมาก

ตัวแปร / ชุดข้อมูล	a	b	c	Distance
1	3.7	2.4	5.3	**-1.6**
2	0.0	0.0	0.0	**0.0**
3	-2.2	-1.4	-4.5	**0.8**

ซึ่งชุดของข้อมูลที่ได้ใหม่นี้จะเป็นประชากรรุ่นใหม่เพื่อนำไปใช้ในการหากรณีทดสอบสำหรับเส้นทางเป้าหมายที่กำลังพิจารณาต่อไป

ขั้นตอนที่ 4 การไขว้เปลี่ยน (Crossover)

การไขว้เปลี่ยนจะดำเนินการโดยเลือกชุดข้อมูลที่ให้ค่า Branch Distance น้อยที่สุด 2 ลำดับแรก เนื่องจากจำนวนตัวแปรหรือจำนวนโครโมโซมของชุดข้อมูลมีจำนวนไม่มาก ดังนั้นการทำการไขว้เปลี่ยนจึงใช้วิธีการไขว้เปลี่ยนแบบจุดเดียว (Single-Point Crossover) ซึ่งจะทำให้ได้ชุดข้อมูลใหม่ 2 ชุดข้อมูล ตัวอย่างเช่น ชุดข้อมูลที่ถูกเลือกมาคือ $\{x_1 , x_2 , x_3 , x_4\}$ และ $\{y_1 , y_2 , y_3 , y_4\}$ จะทำการไขว้เปลี่ยนชุดข้อมูลทั้งสอง ณ ตำแหน่งที่ 4 ดังนั้นจะได้ชุดข้อมูลใหม่ คือ $\{x_1 , x_2 , x_3 , \mathbf{y_4}\}$ และ $\{y_1 , y_2 , y_3 , \mathbf{x_4}\}$

จากการดำเนินการในขั้นตอนที่ 3 จะได้ชุดข้อมูลใหม่มาเป็นประชากรต้นกำเนิด สำหรับงานวิจัยนี้จะทำการสุ่มตำแหน่งที่จะทำการไขว้เปลี่ยนมา 1 ค่า ตัวอย่างเช่น สุ่มได้ตัวเลข 2 ดังนั้นจากชุดข้อมูลดังตารางที่ 3-3 และทำการไขว้เปลี่ยนแบบจุดเดียวในตำแหน่งที่ 2 ซึ่งทำให้ได้ชุดของข้อมูลใหม่ ดังตารางที่ 3-4

ตารางที่ 3-4 แสดงการไขว้เปลี่ยนแบบจุดเดียวที่ตำแหน่งที่ 2

ตำแหน่งที่ทำการไขว้เปลี่ยน

ชุดข้อมูล / ตัวแปร	a	b	c
1	3.7	0.0	0.0
2	0.0	2.4	5.3
3	-2.2	-1.4	-4.5

หลังจากทำการไขว้เปลี่ยนจะได้กลุ่มประชากรใหม่ดังตารางที่ 3-4 ซึ่งจะใช้เป็นกลุ่มประชากรต้นกำเนิดเพื่อนำมาพิจารณากับเส้นทางเป้าหมายที่กำลังพิจารณาต่อไป โดยจะเข้าสู่ขั้นตอนที่ 2 การประเมินค่าความครอบคลุม (Coverage Evaluation) อีกครั้ง ซึ่งในการประเมินค่าความครอบคลุม ในกรณีที่พบชุดข้อมูลทดสอบจะมีผลให้ขั้นตอนปฏิบัติการในขั้นตอนวิธีเชิงพันธุกรรมยุติ แต่ในกรณีที่ไม่พบชุดข้อมูลทดสอบจะไปดำเนินการในขั้นตอนถัดไป

จากข้อมูลในตารางที่ 3-4 เมื่อพิจารณาเส้นทางที่เกิดขึ้นกับข้อมูลขุดที่ 1 พบว่าเป็นเส้นทางเดียวกันกับเส้นทางเป้าหมาย ดังนั้นจึงมีผลให้ขั้นตอนปฏิบัติการในขั้นตอนวิธีเชิงพันธุกรรมยุติ และกรณีทดสอบสำหรับเส้นทางเป้าหมาย S , 1 , 2T , 3 , 7F , 10T , 11 , E คือ a = 3.7 , b = 0.0 และ c = 0.0

ขั้นตอนที่ 5 การกลายพันธุ์ (Mutation)

การกลายพันธุ์จะกระทำโดยเลือกชุดข้อมูลที่ให้ค่า Branch Distance น้อยที่สุดเพื่อทำการกลายพันธุ์แบบจุดเดียว (Single-Point Mutation) ซึ่งจะทำให้ได้ชุดข้อมูลใหม่ 1 ชุดข้อมูล ตัวอย่างเช่น ชุดข้อมูลที่ถูกเลือกมาคือ $\{x_1 , x_2 , x_3 , x_4\}$ จะทำการกลายพันธุ์ ณ ตำแหน่งที่ 2 ด้วยค่า x_m ดังนั้นจะได้ชุดข้อมูลใหม่ คือ $\{x_1 , \mathbf{x_m} , x_3 , x_4\}$

โดยที่ตำแหน่งหรือโครโมโซมที่จะกระทำการกลายพันธุ์ จะได้จากการคัดเลือกโดยวิธีการสุ่ม เนื่องจากจำนวนตัวแปรหรือโครโมโซมของชุดข้อมูลมีจำนวนไม่มาก ดังนั้นจึงดำเนินการกลายพันธุ์เพื่อสร้างชุดข้อมูลใหม่โดยใช้การกลายพันธุ์แบบจุดเดียว โดยจะทำการสุ่มตำแหน่งที่จะทำการกลายพันธุ์มา 1 ตำแหน่ง แล้วเลือกชุดข้อมูลที่ให้ค่าความเหมาะสมน้อย ซึ่งหมายถึงชุดข้อมูลที่ให้ค่า Branch Distance น้อยที่สุดเพื่อเป็นชุดข้อมูลที่ใช้ในการกลายพันธุ์

จากการดำเนินการในขั้นตอนที่ 3 จะได้ชุดข้อมูลใหม่มาเป็นประชากรต้นกำเนิด สำหรับงานวิจัยนี้จะทำการสุ่มตำแหน่งที่จะทำการกลายพันธุ์มา 1 ค่า ตัวอย่างเช่น สุ่มได้ตัวเลข 3 ดังนั้นเมื่อทำการกลายพันธุ์ชุดข้อมูลจากตารางที่ 3-3 โดยใช้การกลายพันธุ์แบบจุดเดียวในตำแหน่งที่ 3 จะทำให้ได้ชุดของข้อมูลใหม่ ดังตารางที่ 3-5 ซึ่งจะเปลี่ยนชุดข้อมูลที่ 1 ในตำแหน่งที่ 3 จากข้อมูลเดิม 5.3 เป็น 7.6

ตารางที่ 3-5 แสดงการกลายพันธุ์แบบจุดเดียวที่ตำแหน่งที่ 3

ตำแหน่งที่ทำการกลายพันธุ์

ชุดข้อมูล / ตัวแปร	a	b	c
1	3.7	2.4	7.6
2	0.0	0.0	0.0
3	-2.2	-1.4	-4.5

หลังจากการกลายพันธุ์จะได้กลุ่มประชากรใหม่ดังตารางที่ 3-5 ซึ่งจะใช้เป็นกลุ่มประชากรต้นกำเนิดเพื่อนำมาพิจารณากับเส้นทางเป้าหมายที่กำลังพิจารณาต่อไป โดยจะเข้าสู่ขั้นตอนที่ 2 การประเมินค่าความครอบคลุม (Coverage Evaluation) อีกครั้ง ซึ่งในการประเมินค่าความครอบคลุม ในกรณีที่พบชุดข้อมูลทดสอบจะมีผลให้ขั้นตอนปฏิบัติการในขั้นตอนวิธีเชิงพันธุกรรมยุติ แต่ในกรณีที่ไม่พบชุดข้อมูลทดสอบจะไปดำเนินการในขั้นตอนถัดไปเรื่อยๆจนกว่าจะพบชุดข้อมูลทดสอบหรือครบจำนวนรอบการทำงานที่กำหนดไว้

บทที่ 4

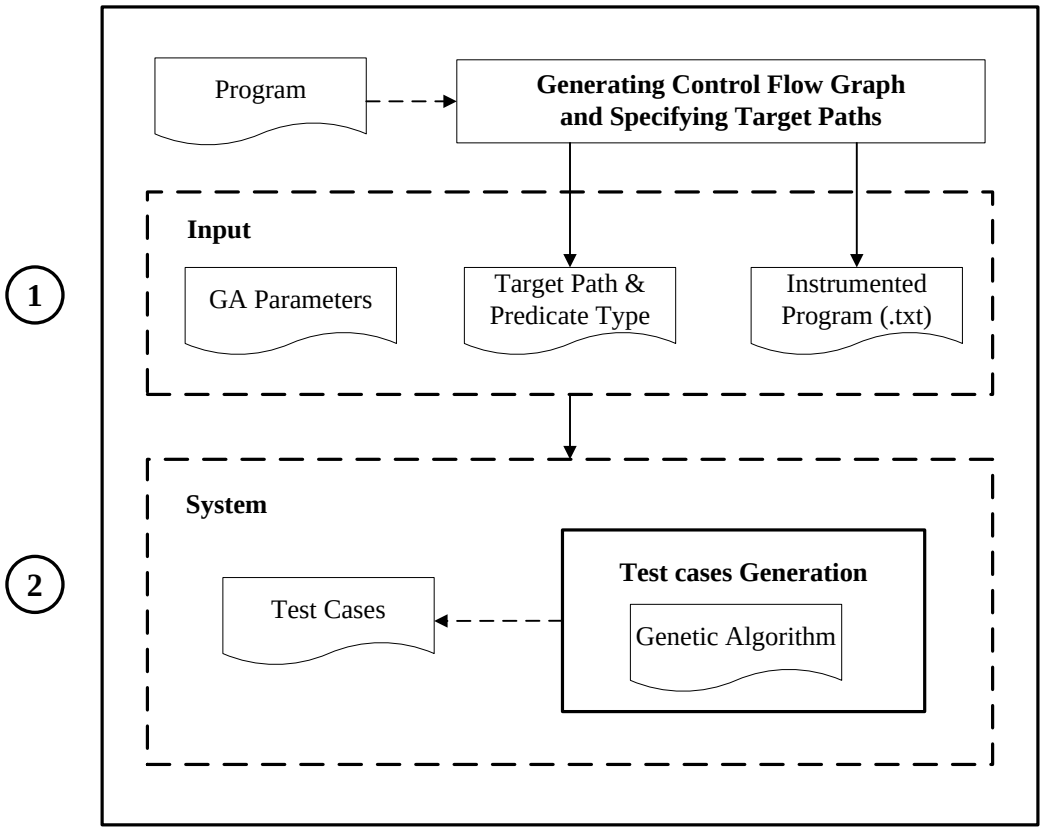
ผลการดำเนินงานวิจัย

4.1 การออกแบบและพัฒนาเครื่องมือต้นแบบสำหรับสร้างกรณีทดสอบ

การพัฒนาเครื่องมือเพื่อใช้ในการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม พัฒนาโดยใช้ Microsoft Visual Studio 2010 และใช้ภาษา Visual Basic ในการพัฒนา โดยเครื่องมือที่พัฒนาขึ้นอยู่ในรูปแบบของวินโดว์แอพลิเคชั่น (Windows Application) ผู้วิจัยได้ออกแบบและพัฒนาให้รองรับหลักการทำงานของขั้นตอนที่ได้นำเสนอ รายละเอียดของเครื่องมือที่พัฒนาขึ้นมีดังนี้

4.1.1 กรอบการทำงานของเครื่องมือที่พัฒนา

เครื่องมือที่พัฒนาขึ้นมีส่วนการทำงานหลักๆ 2 ส่วน แสดงดังภาพที่ 4-1 มีรายละเอียด ดังนี้



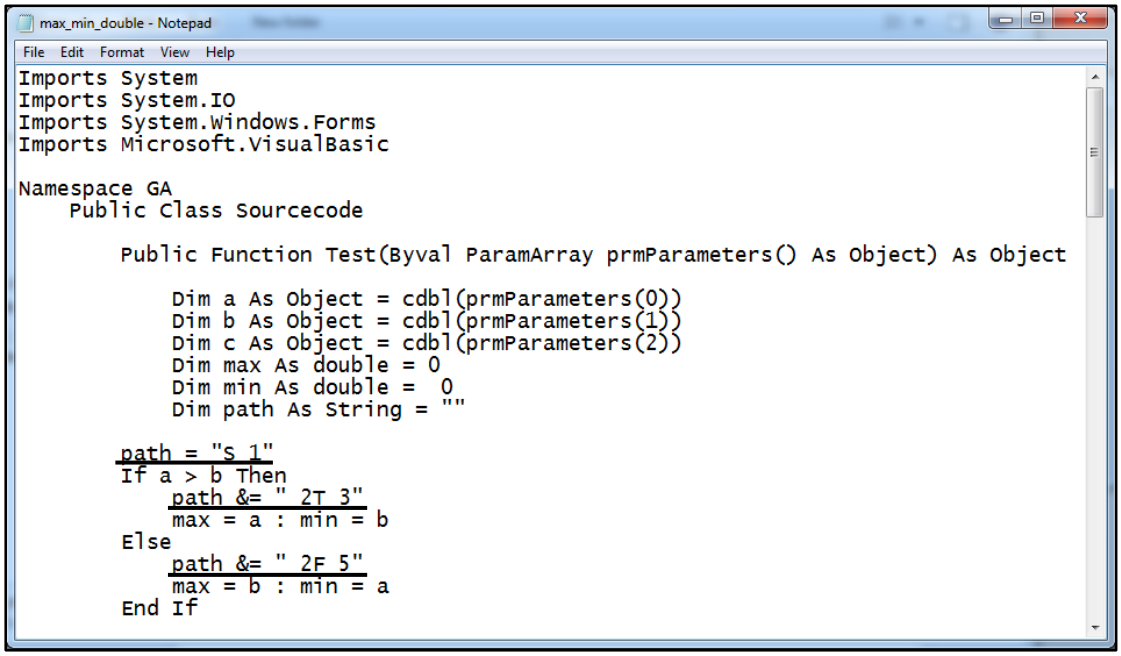
ภาพที่ 4-1 แสดงกรอบการทำงานของเครื่องมือที่พัฒนา

ส่วนที่ 1 Input เป็นส่วนของข้อมูล Input เข้าสู่ระบบ ซึ่งเป็นข้อมูลที่มาจากการสร้างกราฟควบคุมการไหลและระบุเส้นทางเป้าหมายจากรหัสโปรแกรมต้นฉบับ โดยจะแบ่งข้อมูล Input ออกเป็น 3 ส่วนคือ

1) การกำหนดค่าตั้งต้นของขั้นตอนวิธีเชิงพันธุกรรม (GA Parameters)

2) การระบุเส้นทางเป้าหมายและ Predicate Type สำหรับเส้นทางที่กำลังทดสอบ (Target Path & Predicate Type) โดยในการหากรณีทดสอบสำหรับแต่ละเส้นทางเป้าหมายใดจะต้องมีการระบุ Predicate Type สำหรับเส้นทางเป้าหมายนั้นก่อนทุกครั้ง

3) โปรแกรมต้นฉบับที่ถูกสอดแทรกเพิ่มเติมคำสั่ง (Instrumented Program) เพื่อระบุเส้นทางที่ได้จากกราฟควบคุมการไหลทั้งหมด ซึ่งโปรแกรมนี้ถูกจัดเก็บในรูปแบบ Text File (.txt) ตัวอย่างโปรแกรมต้นฉบับและคำสั่งที่ถูกเพิ่ม แสดงดังภาพที่ 4-2

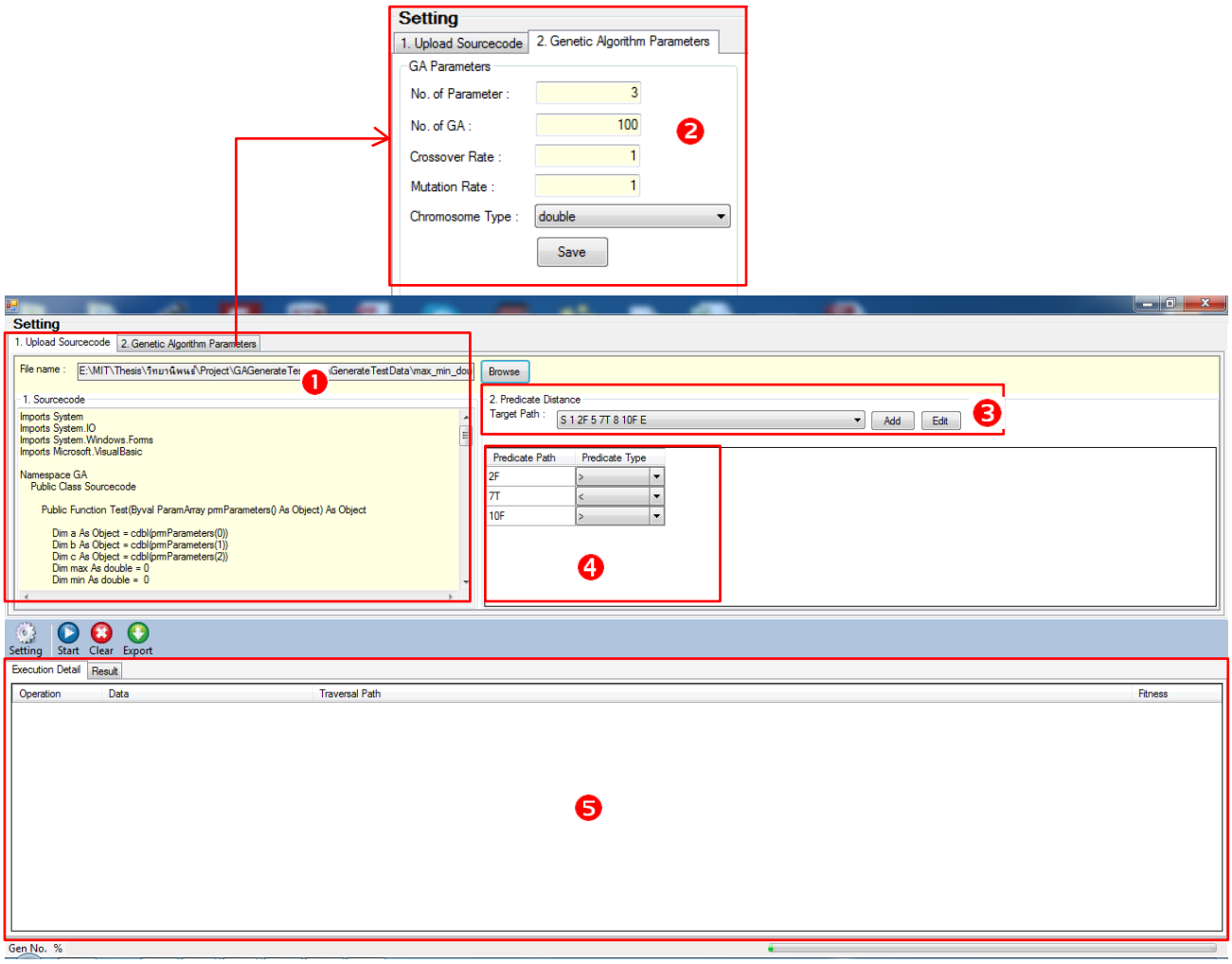


ภาพที่ 4-2 ตัวอย่างรหัสโปรแกรมต้นฉบับในรูปแบบ Text File

ส่วนที่ 2 System เป็นส่วนการทำงานภายในระบบ ซึ่งเป็นส่วนที่พัฒนาขึ้นเพื่อทำการสร้างกรณีทดสอบให้กับโปรแกรมที่ Input เข้าสู่ระบบ โดยใช้ขั้นตอนวิธีเชิงพันธุกรรมในการดำเนินการสร้างกรณีทดสอบ

4.1.2 การออกแบบส่วนติดต่อกับผู้ใช้

เครื่องมือที่พัฒนาขึ้นมีส่วนติดต่อผู้ใช้งานและส่วนการแสดงผลต่างๆ ดังภาพที่ 4-3

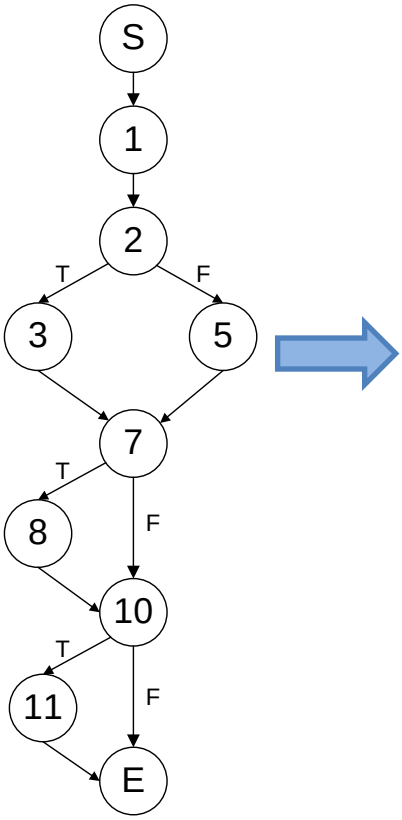


ภาพที่ 4-3 หน้าจอการทำงานในส่วนการติดต่อกับผู้ใช้

จากภาพที่ 4-3 หน้าจอการทำงานในส่วนการติดต่อกับผู้ใช้จะแบ่งออกเป็น 5 ส่วน ซึ่งสัมพันธ์กับขั้นตอนการสร้างกรณีทดสอบ ดังนี้

ขั้นตอนที่ 1 การอัพโหลดไฟล์ (Upload File)

ขั้นตอนการอัพโหลดไฟล์รหัสต้นฉบับเพื่อให้เครื่องมือทำการสร้างกรณีทดสอบ โดยใช้หน้าจอส่วนของเครื่องมือที่พัฒนาหมายเลข ❶ ซึ่งประเภทของไฟล์ที่ใช้ในการอัพโหลดเพื่อทดสอบจะต้องเป็น Text File (.txt) ซึ่งเป็นไฟล์ที่ถูกจัดเตรียมไว้โดยนำไฟล์โปรแกรมต้นฉบับภาษา Visual Basic มาสอดแทรกเพิ่มเติมคำสั่งเพื่อระบุเส้นทางตามกราฟควบคุมการไหลในลักษณะข้อความ (String) ตัวอย่างการสอดแทรกเพิ่มเติมคำสั่งเพื่อระบุเส้นทางที่ต้องการทดสอบแสดงกังภาพที่ 4-4




```
1     Dim a , b , c , max , min As Integer
      Dim path As String
      path = "S 1"
2     If a > b Then
      path &= " 2T 3"
3           max = a : min = b
4     Else
      path &= " 2F 5"
5           max = b : min = a
6     End If
7     If max < c Then
      path &= " 7T 8"
8           max = c
      Else
      path &= " 7F"
9     End If
10    If max > c Then
      path &= " 10T 11"
      Else
      path &= " 10F"
11          min = c
12    End If
      path &= " E"
```

ภาพที่ 4-4 ตัวอย่างไฟล์โปรแกรมหาค่าสูงสุดและต่ำสุด (Max - Min) ที่มีการระบุเส้นทางตามกราฟควบคุมการไหล

ขั้นตอนที่ 2 การกำหนดค่าตั้งต้นในกระบวนการขั้นตอนวิธีเชิงพันธุกรรม

สามารถกำหนดค่าตั้งต้นให้กับการดำเนินการในกระบวนการขั้นตอนวิธีเชิงพันธุกรรมโดยใช้หน้าจอส่วนของเครื่องมือที่พัฒนาหมายเลข ❷ ซึ่งค่าตั้งต้นที่สามารถกำหนดได้ มีดังนี้

1) Number of Parameters คือ จำนวนของพารามิเตอร์หรือตัวแปรที่ต้องการหากรณีทดสอบ
2) Number of Generations คือ จำนวนสูงสุดของรุ่นประชากรที่กำหนด เป็นค่าที่แสดงถึงจำนวนรอบที่จะใช้ในการประมวลผลเพื่อการค้นหาคำตอบ
3) Crossover Probability คือ อัตราความน่าจะเป็นที่จะเกิดการไขว้เปลี่ยนของกลุ่มประชากร ซึ่งจะมีค่าอยู่ระหว่าง 0 ถึง 1 หากกำหนดอัตราการไขว้เปลี่ยนเป็น 1 แสดงว่าจะเกิดกระบวนการไขว้เปลี่ยนในทุกรอบ ส่วนกรณีที่กำหนดอัตราการไขว้เปลี่ยนเป็น 0 แสดงว่าไม่มีโอกาสเกิดกระบวนการไขว้เปลี่ยน
4) Mutation Probability คือ อัตราความน่าจะเป็นที่จะเกิดการกลายพันธุ์ จะมีค่าอยู่ระหว่าง 0 ถึง 1 หากกำหนดอัตราการกลายพันธุ์เป็น 1 แสดงว่าจะเกิดกระบวนการกลายพันธุ์หลังจากการไขว้เปลี่ยนทุกครั้ง ส่วนกรณีที่กำหนดอัตราการกลายพันธุ์เป็น 0 แสดงว่าไม่มีโอกาสเกิดกระบวนการไขว้เปลี่ยน
5) Chromosome Type คือ ประเภทของข้อมูล Input ที่ใช้สำหรับสร้างกรณีทดสอบซึ่งเป็นประเภทหรือชนิดของข้อมูล เช่น จำนวนเต็ม (Integer), จำนวนทศนิยม (Double), รหัสอักขระ (Character), ข้อความ (String) เป็นต้น

ขั้นตอนที่ 3 การกำหนดเส้นทางเป้าหมายที่ต้องการสร้างกรณีทดสอบ

เส้นทางเป้าหมายทั้งหมดสำหรับใช้ในการสร้างกรณีทดสอบระบุลงในเครื่องมือที่พัฒนาขึ้นโดยใช้หน้าจอส่วนของเครื่องมือที่พัฒนาหมายเลข ❸

ขั้นตอนที่ 4 การระบุ Predicate Type ให้กับคำสั่งที่เป็นเงื่อนไข

ระบุ Predicate type สำหรับคำสั่งที่เป็นเงื่อนไขโดยใช้หน้าจอส่วนของเครื่องมือที่พัฒนาหมายเลข ❹

ขั้นตอนที่ 5 การสร้างกรณีทดสอบ

เมื่อระบุข้อมูลที่จำเป็นลงในเครื่องมือที่พัฒนาครบถ้วนแล้ว ก็จะเข้าสู่กระบวนการสร้างกรณีทดสอบ โดยเครื่องมือที่พัฒนาขึ้นจะทำการสร้างกรณีทดสอบให้กับโปรแกรมที่ได้ระบุไว้ในขั้นตอนที่ 1 และจะทำการสร้างกรณีทดสอบสำหรับเส้นทางเป้าหมายที่กำลังพิจารณาซึ่งกำหนดไว้ในขั้นตอนที่ 3 โดยระบบจะทำการค้นหากรณีทดสอบที่มีความเหมาะสมให้กับเส้นทางที่กำลังพิจารณาจนกว่าจะพบคำตอบหรือจนกว่าจะครบเงื่อนไขการกำหนดค่าตั้งต้นที่ได้กำหนดไว้ในขั้นตอนที่ 2 โดยหน้าจอส่วนที่แสดงผลการสร้างกรณีทดสอบใช้หน้าจอส่วนของเครื่องมือที่พัฒนาหมายเลข ❺

ปุ่มการทำงานที่จำเป็นสำหรับการสร้างกรณีทดสอบ แสดงดังภาพที่ 4-5 ซึ่งแต่ละปุ่มมีรายละเอียดการทำงาน ดังนี้

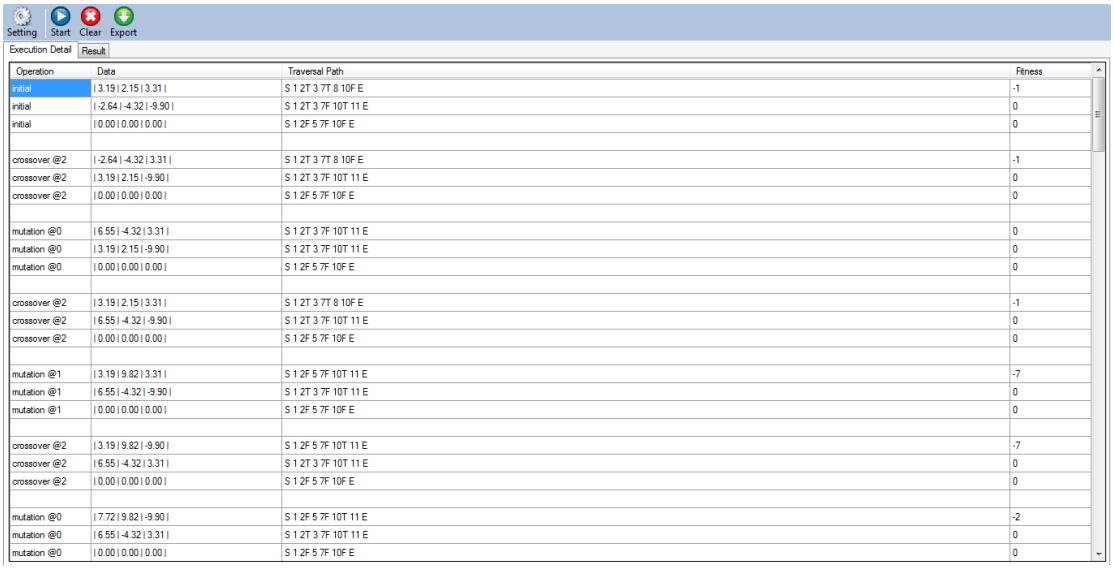
1. **Setting** : ซ่อน / แสดง ส่วนของการตั้งค่าการใช้งาน
2. **Start** : เริ่มการประมวลผล
3. **Clear** : ล้างหน้าจอแสดงผลการทำงานของโปรแกรม
4. **Export** : ส่งออกผลการทำงานของโปรแกรมในรูปแบบไฟล์ MS Excel

ภาพที่ 4-5 ปุ่มการทำงาน

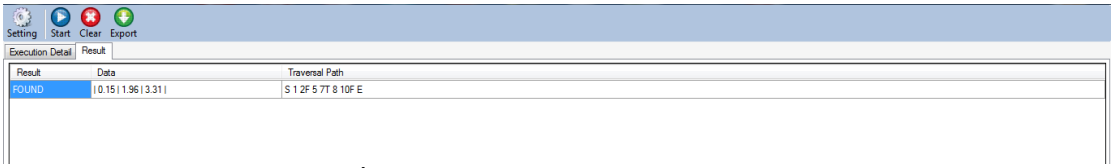
เมื่อระบุเส้นทางเป้าหมายที่ต้องการทดสอบ แล้วกดปุ่ม Start เครื่องมือจะทำการค้นหากรณีทดสอบที่เหมาะสมกับเส้นทางเป้าหมายที่ระบุ โดยภาพที่ 4-6 แสดงหน้าจอรายละเอียดการทำงานของขั้นตอนวิธีเชิงพันธุกรรม โดยจะแสดงรายละเอียดต่างๆ ดังนี้

1) Operation คือ กระบวนการในขั้นตอนวิธีเชิงพันธุกรรม ได้แก่ Initial, Crossover และ Mutation
2) Data คือ ชุดของข้อมูลที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ
3) Traversal Path คือ เส้นทางที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ
4) Fitness คือ ค่าความเหมาะสมที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ

ภาพที่ 4-7 แสดงหน้าจอแสดงผลการหากรณีทดสอบ



ภาพที่ 4-6 หน้าจอแสดงรายละเอียดการการปฏิบัติการดำเนินการทางพันธุกรรมในการหากรณีทดสอบสำหรับเส้นทางเป้าหมายที่ระบุ (Execution Detail)



ภาพที่ 4-7 หน้าจอแสดงผลการหากรณีทดสอบ

4.2 ผลการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมจากเครื่องมือที่พัฒนา

ทดสอบการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมด้วยเครื่องมือที่พัฒนาโดยใช้โปรแกรมเพื่อเป็นกรณีศึกษา 4 โปรแกรม ซึ่งมีข้อมูลนำเข้าที่แตกต่างกัน ดังนี้

1) โปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด ข้อมูลเข้าประเภทจำนวนทศนิยม (Double)
2) โปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม ข้อมูลเข้าประเภทจำนวนเต็ม (Integer)
3) โปรแกรมคำนวณหาค่ากลางของข้อมูล ข้อมูลเข้าประเภทรหัสอักขระ (Character)
4) โปรแกรมตรวจสอบข้อความ Palindrome ซึ่งเป็นข้อความที่สะกดเหมือนเดิมจากหน้าไปหลังหรือจากหลังไปหน้า ข้อมูลเข้าประเภทข้อความ (String)

จากผลการทดลอง พบว่าขั้นตอนวิธีเชิงพันธุกรรมสามารถนำมาประยุกต์ใช้ในการสร้างกรณีทดสอบในกระบวนการพัฒนาซอฟต์แวร์ได้ ซึ่งในการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม กำหนดรอบในการทำงานเพื่อค้นหากรณีทดสอบที่ครอบคลุมเส้นทางที่ต้องการทดสอบจำนวน 100 รอบ, กำหนดอัตราความน่าจะเป็นในการไขว้เปลี่ยนเป็น 1 และกำหนดอัตราความน่าจะเป็นในการกลายพันธุ์เป็น 1 สามารถแสดงผลได้ดังตารางที่ 4-1 – ตารางที่ 4-5

ตารางที่ 4-1 แสดงเส้นทางสำหรับการทดสอบโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุดมีเส้นทางเป้าหมายทั้งหมด 8 เส้นทาง มีเส้นทางที่สามารถหากรณีทดสอบที่ได้จำนวน 6 เส้นทาง ไม่สามารถหากรณีทดสอบได้จำนวน 2 เส้นทาง คิดเป็น 75 %

ตารางที่ 4-1 กรณีทดสอบที่ได้จากโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Test Case	Target Path	Result	Input Result		
			a	b	c
1	S 1 2T 3 7T 8 10T 11 E	Not Found	-	-	-
2	S 1 2T 3 7T 8 10F E	Found	3.05	1.64	3.63
3	S 1 2T 3 7F 10T 11 E	Found	8.76	3.74	2.85
4	S 1 2T 3 7F 10F E	Found	9.77	0.13	9.77
5	S 1 2F 5 7T 8 10T 11 E	Not Found	-	-	-
6	S 1 2F 5 7T 8 10F E	Found	2.48	5.96	7.35
7	S 1 2F 5 7F 10T 11 E	Found	2.20	8.75	1.39
8	S 1 2F 5 7F 10F E	Found	0.0	0.0	0.0

เส้นทางที่ไม่สามารถหากรณีทดสอบได้สำหรับโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด มีทั้งสิ้น 2 เส้นทาง เมื่อพิจารณาจะพบว่าเส้นทางเป้าหมายเส้นทางที่ 1 และเส้นทางเป้าหมายที่ 5 เป็นเส้นทางที่ไม่สามารถเข้าถึงได้ (infeasible path) เนื่องจากทั้งสองเส้นทางนี้กำหนดให้ทั้งเงื่อนไข $max < c$ และ $max > c$ เป็นจริงทั้งคู่ซึ่งเป็นไปไม่ได้

ตารางที่ 4-2 แสดงเส้นทางสำหรับการทดสอบโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยมมีเส้นทางเป้าหมายทั้งหมด 8 เส้นทาง มีเส้นทางที่สามารถหากรณีทดสอบที่ได้จำนวน 4 เส้นทาง ไม่สามารถหากรณีทดสอบได้จำนวน 4 เส้นทาง คิดเป็น 50 %

ตารางที่ 4-2 กรณีทดสอบที่ได้จากโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Test Case	Target Path	Result	Input Result		
			a	b	c
1	S 1 2 3 4T 5 9F 18 E	Not Found	-	-	-
2	S 1 2 3 4T 5 9T 10T 11 E	Found	9	9	9
3	S 1 2 3 4T 5 9T 10F 12T 13 E	Found	4	2	3
4	S 1 2 3 4T 5 9T 10F 12F 15 E	Found	3	3	2
5	S 1 2 3 4F 7 9F 18 E	Found	-5	-4	-8
6	S 1 2 3 4F 7 9T 10T 11 E	Not Found	-	-	-
7	S 1 2 3 4F 7 9T 10F 12T 13 E	Not Found	-	-	-
8	S 1 2 3 4F 7 9T 10F 12F 15 E	Not Found	-	-	-

เส้นทางที่ไม่สามารถหากรณีทดสอบได้สำหรับโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม มีทั้งสิ้น 4 เส้นทาง เมื่อพิจารณาจะพบว่าเส้นทางเป้าหมายเส้นทางที่ 1 เป็นเส้นทางที่ไม่สามารถเข้าถึงได้ (infeasible path) เนื่องจากทั้งเส้นทางนี้กำหนดให้ค่าตัวแปร Is_Triangle ในเส้นทาง 4T เป็นจริง ดังนั้นจึงเป็นไปไม่ได้ที่จะมีเส้นทาง 9F เกิดขึ้น เช่นเดียวกันสำหรับเส้นทางเป้าหมายเส้นทางที่ 6, 7 และเส้นทางเป้าหมายที่ 8 เป็นเส้นทางที่ไม่สามารถเข้าถึงได้ (infeasible path) เนื่องจากทั้งสามเส้นทางนี้กำหนดให้ค่าตัวแปร Is_Triangle ในเส้นทาง 4F เป็นเท็จ ดังนั้นจึงเป็นไปไม่ได้ที่จะมีเส้นทาง 9T เกิดขึ้น

ตารางที่ 4-3 แสดงเส้นทางสำหรับการทดสอบโปรแกรมคำนวณหาค่ากลางของข้อมูล มีเส้นทางเป้าหมายทั้งหมด 6 เส้นทาง มีเส้นทางที่สามารถหากรณีทดสอบที่ได้จำนวน 6 เส้นทาง คิดเป็น 100 %

ตารางที่ 4-3 กรณีทดสอบที่ได้จากโปรแกรมคำนวณหาค่ากลางของข้อมูล

Test Case	Target Path	Result	Input Result		
			a	b	c
1	S 1 2T 3 4 5 6T 7 E	Found	U	c	l
2	S 1 2T 3 4 5 6F 8T 9 11 E	Found	Y	y	l
3	S 1 2T 3 4 5 6F 8F 10 11 E	Found	P	n	‘ ‘
4	S 1 2F 6T 7 E	Found	N	I	V
5	S 1 2F 6F 8T 9 11 E	Found	q	J	i
6	S 1 2F 6F 8F 10 11 E	Found	‘ ‘	‘ ‘	‘ ‘

ตารางที่ 4-4 แสดงเส้นทางสำหรับการทดสอบโปรแกรมตรวจสอบข้อความ Palindrome มีเส้นทางเป้าหมายทั้งหมด 10 เส้นทาง มีเส้นทางที่สามารถหากรณีทดสอบที่ได้จำนวน 10 เส้นทาง คิดเป็น 100 %

ตารางที่ 4-4 กรณีทดสอบที่ได้จากโปรแกรมตรวจสอบข้อความ Palindrome

Test Case	Target Path	Result	Input Result
1	S 1 2 3 4 5T 6 15 2 3 15 E	Found	XLJc8WcV1y
2	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 15 E	Found	tYpVpkK5xQ
3	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E	Found	SjSLmRnVrS
4	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E	Found	7fXgbiZuZt
5	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E	Found	0dlMCNFQXQ
6	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E	Found	rnYnrBE6El
7	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	eeNnvnMnkJ
8	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	iSOSOEq9EH
9	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 15 2 3 15 E	Found	wNKQ8Q8PcP
10	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	ororOadjeJ

จำนวนกรณีทดสอบที่สร้างได้สำหรับโปรแกรมตัวอย่างทั้ง 4 โปรแกรมแสดงดังตารางที่ 4-5

ตารางที่ 4-5 จำนวนกรณีทดสอบของโปรแกรมต่างๆ

Program	Path Coverages	Infeasible Paths	Total Test Cases
Calculate Maximum – Minimum	8	2	6
Triangle Problem	8	4	4
Find Middle Character	6	-	6
Palindrome	10	-	10

จากการทดสอบการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมด้วยเครื่องมือที่พัฒนาโดยใช้โปรแกรมกรณีศึกษา 4 โปรแกรม แสดงผลการทดสอบดังตารางที่ 4-5 เมื่อพิจารณาผลการทดลองที่ได้จะพบว่า เครื่องมือที่พัฒนาขึ้นสามารถหากรณีทดสอบสำหรับโปรแกรมตัวอย่างทั้ง 4 ได้

บทที่ 5

การประเมินประสิทธิภาพของกรณีทดสอบ

งานวิจัยนี้เลือกใช้วิธีการทดสอบแบบกลายพันธุ์ (Mutation Testing) ในการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นจากขั้นตอนวิธีเชิงพันธุกรรมที่ได้นำเสนอ โดยโปรแกรมที่ใช้เป็นกรณีศึกษาในงานวิจัยนี้คือโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด, โปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม, โปรแกรมคำนวณหาค่ากลางของข้อมูล และโปรแกรมตรวจสอบข้อความ Palindrome ซึ่งขั้นตอนในการประเมินประสิทธิภาพของกรณีทดสอบมีดังต่อไปนี้

1) ขั้นตอนการสร้างกรณีทดสอบ ถูกดำเนินการโดยใช้กระบวนการขั้นตอนวิธีเชิงพันธุกรรมตามวิธีการที่นำเสนอ จำนวนกรณีทดสอบของแต่ละโปรแกรมแสดงดังตารางที่ 5-1

2) ขั้นตอนการสร้างโปรแกรมกลายพันธุ์ (Mutant Program) ดำเนินการเปลี่ยนแปลงโปรแกรมต้นฉบับ (Original Program) โดยใช้ตัวดำเนินการการกลายพันธุ์ (Mutation Operator) สำหรับภาษา Visual Basic ซึ่งนำเสนอโดย P. Stylianos Yiasemis และ A. S. Andreou [16] จำนวนโปรแกรมกลายพันธุ์ที่ถูกสร้างสำหรับโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุดมีจำนวนเท่ากับ 21, จำนวนโปรแกรมกลายพันธุ์ที่ถูกสร้างสำหรับโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม มีจำนวนเท่ากับ 38, จำนวนโปรแกรมกลายพันธุ์ที่ถูกสร้างสำหรับโปรแกรมคำนวณหาค่ากลางของข้อมูล มีจำนวนเท่ากับ 21 และจำนวนโปรแกรมกลายพันธุ์ที่ถูกสร้างสำหรับโปรแกรมตรวจสอบข้อความ Palindrome มีจำนวนเท่ากับ 21 แสดงดังตารางที่ 5-1

3) นำกรณีทดสอบที่ได้มาดำเนินการกับโปรแกรมต้นฉบับ (Original Program) และโปรแกรมกลายพันธุ์ต่างๆ เพื่อเปรียบเทียบผลลัพธ์ และทำการคำนวณหาคะแนนการกลายพันธุ์ (Mutation Score) เพื่อประเมินประสิทธิภาพของกรณีทดสอบที่ได้ หากคะแนนการกลายพันธุ์มีค่าเข้าใกล้ 1 หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพมาก และหากคะแนนการกลายพันธุ์มีค่าเข้าใกล้ศูนย์หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพค่อนข้างต่ำ

สรุปผลการประเมินประสิทธิภาพของกรณีทดสอบสำหรับโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด, โปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม, โปรแกรมคำนวณหาค่ากลางของข้อมูล และโปรแกรมตรวจสอบข้อความ Palindrome ซึ่งประเมินจากทุกกรณีโดยใช้วิธีการทดสอบแบบกลายพันธุ์ ได้ดังตารางที่ 5-1

ตารางที่ 5-1 สรุปผลการประเมินประสิทธิภาพกรณีทดสอบ

Program	Total Test Cases	Total Mutants	Killed Mutants	Equivalent Mutants	Mutation Score
Calculate Maximum – Minimum	6	21	19	-	0.9
Triangle Problem	4	38	25	9	0.86
Find Middle Character	6	21	21	-	1
Palindrome	10	21	16	4	0.94

ผลการประเมินประสิทธิภาพของกรณีทดสอบของโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด (Calculate Maximum – Minimum) ซึ่งมีโปรแกรมกลายพันธุ์ทั้งหมด 21 Mutants พบว่ามีโปรแกรมกลายพันธุ์ที่ให้ผลลัพธ์แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับหรือโปรแกรมกลายพันธุ์ที่ถูกฆ่า (Killed Mutant) จำนวน 19 Mutants ซึ่งสามารถหาคะแนนการกลายพันธุ์ (Mutation Score) ได้เท่ากับ 0.9 จากผลการทดลองที่ได้ แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากขั้นตอนที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพค่อนข้างสูง

ผลการประเมินประสิทธิภาพของกรณีทดสอบของโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม ซึ่งมีโปรแกรมกลายพันธุ์ทั้งหมด 38 Mutants พบว่ามีโปรแกรมกลายพันธุ์ที่ให้ผลลัพธ์แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับหรือโปรแกรมกลายพันธุ์ที่ถูกฆ่า (Killed Mutant) จำนวน 25 Mutants มีโปรแกรมกลายพันธุ์ที่สมมูล (Equivalent Mutant) จำนวน 9 Mutants ซึ่งสามารถหาคะแนนการกลายพันธุ์ (Mutation Score) ได้เท่ากับ 0.86 จากผลการทดลองที่ได้ แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากขั้นตอนที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพค่อนข้างสูง

ผลการประเมินประสิทธิภาพของกรณีทดสอบของโปรแกรมคำนวณหาค่ากลางของข้อมูล ซึ่งมีโปรแกรมกลายพันธุ์ทั้งหมด 21 Mutants พบว่ามีโปรแกรมกลายพันธุ์ที่ให้ผลลัพธ์แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับหรือโปรแกรมกลายพันธุ์ที่ถูกฆ่า (Killed Mutant) จำนวน 21 Mutants ซึ่งสามารถหาคะแนนการกลายพันธุ์ (Mutation Score) ได้เท่ากับ 1 จากผลการทดลองที่ได้ แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากขั้นตอนที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพค่อนข้างสูง

ผลการประเมินประสิทธิภาพของกรณีทดสอบของโปรแกรมตรวจสอบข้อความ Palindrome ซึ่งมีโปรแกรมกลายพันธุ์ทั้งหมด 21 Mutants พบว่ามีโปรแกรมกลายพันธุ์ที่ให้ผลลัพธ์แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับหรือโปรแกรมกลายพันธุ์ที่ถูกฆ่า (Killed Mutant) จำนวน 16 Mutants มีโปรแกรมกลายพันธุ์ที่สมมูล (Equivalent Mutant) จำนวน 4 Mutants ซึ่ง

สามารถหาคะแนนการกลายพันธุ์ (Mutation Score) ได้เท่ากับ 0.94 จากผลการทดลองที่ได้ แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากขั้นตอนที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพค่อนข้างสูง

สำหรับรายละเอียดการประเมินระสิทธิภาพของกรณีทดสอบของกรณีศึกษา โดยใช้ขั้นตอนวิธีเชิงพันธุกรรมที่ได้นำเสนอในการสร้างกรณีทดสอบ แสดงในภาคผนวก ก

บทที่ 6

บทสรุปและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

การทดสอบซอฟต์แวร์เป็นขั้นตอนที่มีความสำคัญในกระบวนการผลิตซอฟต์แวร์ โดยขั้นตอนหนึ่งที่สำคัญในกระบวนการทดสอบซอฟต์แวร์ คือการคัดเลือกข้อมูลที่นำมาใช้ในการทดสอบ ดังนั้นการนำเสนอขั้นตอนการสร้างกรณีทดสอบที่ดีจะมีส่วนทำให้การทดสอบมีประสิทธิภาพ โดยกรณีทดสอบที่มีประสิทธิภาพจะต้องครอบคลุมคุณลักษณะต่างๆ ของระบบและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไป

วิทยานิพนธ์นี้ได้นำเสนอวิธีการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม โดยการสร้างกราฟควบคุมการไหลจากโปรแกรมรหัสต้นฉบับและทำการระบุเส้นทางที่เป็นไปได้ สำหรับวิธีการค้นหากรณีทดสอบที่มีความเหมาะสมกับแต่ละเส้นทางใช้หลักการของขั้นตอนวิธีเชิงพันธุกรรม โดยเริ่มจากการสร้างกลุ่มประชากรต้นกำเนิดซึ่งจะถูกสร้างขึ้นโดยใช้วิธีการแบ่งส่วนที่สมดุล (Equivalence Partitioning) การประเมินค่าความเหมาะสมจะพิจารณาจากการประเมินค่าความครอบคลุมของชุดข้อมูลที่ใช้ในการทดสอบ สำหรับวิทยานิพนธ์นี้การประเมินค่าความครอบคลุมใช้วิธีการคำนวณหาค่า Distance ของคำสั่งที่เป็นเงื่อนไขในการทำงานของโปรแกรม ตามทฤษฎี Korel’s distance function ขั้นตอนวิธีเชิงพันธุกรรมจะประกอบไปด้วยการดำเนินการ 2 รูปแบบ ได้แก่ การไขว้เปลี่ยน (Crossover) และการกลายพันธุ์ (Mutation) ในกรณีที่กลุ่มประชากรต้นกำเนิดที่ได้จากการสุ่มในการทำงานรอบแรกยังไม่ใช้กรณีทดสอบที่เหมาะสมกับเส้นทางเป้าหมายที่กำลังหากรณีทดสอบ กลุ่มประชากรต้นกำเนิดจะถูกนำเข้าสู่กระบวนการไขว้เปลี่ยนและการกลายพันธุ์เพื่อเกิดเป็นกลุ่มประชากรใหม่ต่อไป ซึ่งอัตราความน่าจะเป็นในการเกิดการไขว้เปลี่ยนและการกลายพันธุ์จะถูกกำหนดไว้ในส่วนของการกำหนดค่าตั้งต้นให้กับขั้นตอนวิธีเชิงพันธุกรรม ซึ่งกระบวนการทางพันธุกรรมจะถูกดำเนินไปเรื่อยๆ โดยเส้นทางเป้าหมายแต่ละเส้นทางจะถูกนำมาพิจารณาเพื่อหาชุดข้อมูลที่เหมาะสมสำหรับการทดสอบเส้นทางนั้นๆ ในกรณีที่ยังไม่มีชุดข้อมูลที่เหมาะสมสำหรับเส้นทางใด เส้นทางนั้นจะถูกดำเนินการพิจารณาต่อไปจนครบจำนวนรอบการทำงานที่ตั้งไว้ ส่วนกรณีที่พบชุดข้อมูลที่เหมาะสมกับเส้นทางเป้าหมายที่กำลังทดสอบ กระบวนการทางพันธุกรรมก็จะจบลง

ในงานวิจัยนี้ได้ทำการประเมินประสิทธิภาพของกรณีทดสอบที่ได้จากวิธีการที่นำเสนอโดยใช้วิธีการทดสอบแบบกลายพันธุ์ (Mutation Testing) ซึ่งจากผลการประเมินประสิทธิภาพของกรณีทดสอบพบว่า กรณีทดสอบที่สร้างจากขั้นตอนวิธีที่นำเสนอมีประสิทธิภาพ นอกจากนี้วิทยานิพนธ์นี้ยังได้พัฒนาเครื่องมือเพื่อช่วยในการสร้างกรณีทดสอบตามวิธีการที่ได้นำเสนอ ซึ่งช่วยให้ขั้นตอนในการสร้างกรณีทดสอบมีความสะดวกและรวดเร็วขึ้น

6.2 ข้อเสนอแนะสำหรับการวิจัยในอนาคต

การนำเสนองานวิจัยนี้เพื่อเป็นแนวทางในการสร้างเครื่องมือสำหรับการสร้างกรณีทดสอบเพื่อใช้ในกระบวนการทดสอบซอฟต์แวร์ แนวทางในการพัฒนาต่อในอนาคตนั้นสามารถพัฒนาต่อได้ในหลากหลายแนวทาง ดังนี้

1. วิทยานิพนธ์นี้ดำเนินการสร้างกรณีทดสอบจากการแปลงรหัสต้นฉบับ (Source Code) ให้อยู่ในรูปของกราฟควบคุมการไหล (Control Flow Graph) เพื่อใช้ในการระบุเส้นทางที่ใช้ในการหากรณีทดสอบ ซึ่งการสร้างกราฟควบคุมการไหลและระบุเส้นทางที่ทดสอบ ผู้ทดสอบจะต้องระบุด้วยตนเอง (Manual) ในอนาคตอาจมีการพัฒนาเครื่องมือให้รองรับการแปลงรหัสต้นฉบับให้อยู่ในรูปของกราฟควบคุมการไหลแบบอัตโนมัติ เพื่อให้เกิดความสะดวกรวดเร็วขึ้น

2. เครื่องมือที่พัฒนาขึ้นในวิทยานิพนธ์นี้ สามารถดำเนินการสร้างกรณีทดสอบสำหรับชนิดข้อมูลบางประเภท ได้แก่ จำนวนเต็ม (Integer), จำนวนทศนิยม (Double), รหัสอักขระ (Character) และข้อความ (String) ซึ่งในอนาคตอาจมีการพัฒนาวิธีการสร้างกรณีทดสอบสำหรับชนิดข้อมูลอื่นเพิ่มเติม

3. เนื่องจากในส่วนของการสร้างกรณีทดสอบ จะพบว่าบางเส้นทางไม่สามารถหากรณีทดสอบได้ ทั้งนี้เนื่องจากเป็นเส้นทางที่ไม่สามารถเข้าถึง (Infeasible Path) งานวิจัยในอนาคตจึงอาจต้องศึกษาหาวิธีการจัดการกับเส้นทางที่ไม่สามารถเข้าถึงนี้

บรรณานุกรม

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli. "Fundamentals of Software Engineering 2nd," *Prentice Hall PTR Upper Saddle River, NJ, USA,* 2002.

[2] B. Korel. "Automated test data generation for programs with procedures," *Proceedings of International Symposium on Software Testing and Analysis*, pages 209-215, 1996.

[3] C. Michael, G. McGraw, and A. Schatz. "Generating software test data by evolution," *IEEE Transactions on Software Engineering,* pp. 1085-1110, 2001.

[4] M. Sarma and R. Mall. "AutomaticTest Case Generation from UML Models," *10th International Conference on Information Technology*, pp. 196-201, 2007.

[5] T. Lerthumpanya and T. Senivongse. "Basis Path Test Suite and Testing Process for WS-BPEL," *WSEAS TRANSACTIONS on COMPUTERS, Issue 5, Vol.7*, pp. 483-496, 2008.

[6] Mark Kevitt. "Best Software Test & Quality Assurance Practices in the project Life-cycle," *School Computer Applications*
Dublin City University, 2008.

[7] Pezze and Young. "Software Testing and Analysis : Process, Principles and Techniques," *Wiley, USA,* 2008.

[8] P. Ammann and J. Offutt. "Introduction to Software Testing," *Cambridge University Press*, 2008.

[9] Xin-SheYang. "Engineering Optimization An Introduction with Metaheuristic Applications," *Wiley Publisher*, 2010.

[10] G. Winter, J. Periaux, M.Galan and P.Cuesta. "Genetic Algorithms in Engineering and Computer Science," *Wiley Publisher*, 1996.

บรรณานุกรม (ต่อ)

[11] P. C. Jorgensen. “Software Testing A Craftsman’s Approach 3rd,” *AuerbachPiblictions, USA,* 2007.

[12] A.J. Offutt, A.Lee, G. Rothermel, R. Untch, and C. Zapf, “An Experimental Determination of Sufficient Mutant Operators,” *ACM Transaction on Software Engineering Methodology* , pp. 99-118, 1996.

[13] L. Guti´errez-Madro˜nal, J. Jos´e Dom´ınguez-Jim´enez and I. Medina-Bulo. “Mutation Testing: Guideline and Mutation Operator Classification,” *The Ninth International Multi-Conference on Computing in the Global Information Technology***,** pp. 171-179, 2014.

[14] P. Ranjan Srivastava and Tai-hoon Kim. “Application of Genetic Algorithm in Software Testing,” *International Journal of Software Engineering and Its Applications, Vol.3, No.4*, 2009.

[15] M. Alzabidi, A. Kumar and A.D. Shaligram. “Automatic Software Structural Testing by Using Evolutionary Algorithms for Test Data Generations,” *International Journal of Computer Science and Network Security, Vol.9, No.4,* 2009.

[16] P. Stylianos Yiasemis และ A. S. Andreou. “Testing Object-Oriented Code Through a Specifications-Based Mutation Engine*,” International Journal on Advances in Software, vol 5 no 3 & 4,* 2012.

ภาคผนวก

ภาคผนวก ก

กรณีศึกษาการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

1. กรณีศึกษา การสร้างกรณีทดสอบโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Program Description

ตารางที่ ก-1 รายละเอียดโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

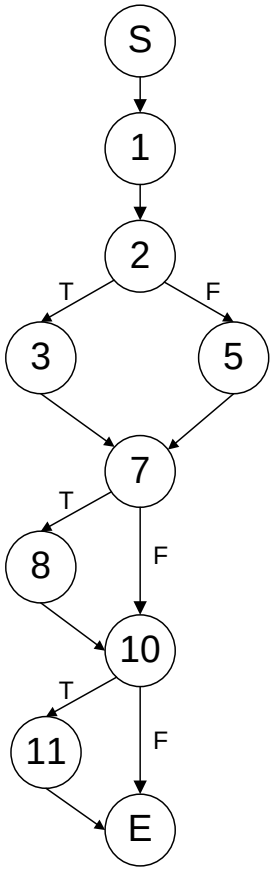
ชื่อโปรแกรม	Calculate Maximum – Minimum
รายละเอียดโปรแกรม	คำนวณค่าสูงสุดและค่าต่ำสุดจากข้อมูลนำเข้า 3 ตัวที่เป็นจำนวนทศนิยม (Double)
จำนวน input	3
ประเภท input	จำนวนทศนิยม (Double)
ประเภท output	จำนวนทศนิยม (Double) - ค่าสูงสุด (Maximum) - ค่าต่ำสุด (Minimum)

Source Code

```
1    Dim a , b , c , max , min As Integer
2    If a > b Then
3          max = a : min = b
4    Else
5          max = b : min = a
6    End If
7    If max < c Then
8          max = c
9    End If
10   If max > c Then
11         min = c
12   End If
```

ภาพที่ ก-1 รหัสต้นฉบับโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Control Flow Graph



ภาพที่ ก-2 กราฟควบคุมการไหลโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Target Path

ตารางที่ ก-2 เส้นทางสำหรับการสร้างกรณีทดสอบโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Path 1	S	1	2T	3	7T	8	10T	11	E
Path 2	S	1	2T	3	7T	8	10F	E	
Path 3	S	1	2T	3	7F	10T	11	E	
Path 4	S	1	2T	3	7F	10F	E		
Path 5	S	1	2F	5	7T	8	10T	11	E
Path 6	S	1	2F	5	7T	8	10F	E	
Path 7	S	1	2F	5	7F	10T	11	E	
Path 8	S	1	2F	5	7F	10F	E		

Genetic Algorithm Parameters

ตารางที่ ก-3 การกำหนดค่าตั้งต้นโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Number of Parameters	3
Number of Generations	100
Crossover Probability	1.0
Mutation Probability	1.0
Chromosome Type	Integer

Test Case Generation

ตารางที่ ก-4 ผลการสร้างกรณีทดสอบโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Test Case	Target Path	Result	Input Result		
			a	b	c
1	S 1 2T 3 7T 8 10T 11 E	Not Found	-	-	-
2	S 1 2T 3 7T 8 10F E	Found	3.05	1.64	3.63
3	S 1 2T 3 7F 10T 11 E	Found	8.76	3.74	2.85
4	S 1 2T 3 7F 10F E	Found	9.77	0.13	9.77
5	S 1 2F 5 7T 8 10T 11 E	Not Found	-	-	-
6	S 1 2F 5 7T 8 10F E	Found	2.48	5.96	7.35
7	S 1 2F 5 7F 10T 11 E	Found	2.20	8.75	1.39
8	S 1 2F 5 7F 10F E	Found	0.0	0.0	0.0

Mutation Testing

ตารางที่ ก-5 สรุปผลการประเมินประสิทธิภาพกรณีทดสอบโปรแกรมคำนวณค่าสูงสุดและค่าต่ำสุด

Program	Total Test Cases	Total Mutants	Killed Mutants	Mutation Score
Calculate Maximum – Minimum	6	21	19	0.9

2. กรณีศึกษา การสร้างกรณีทดสอบโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Program Description

ตารางที่ ก-6 รายละเอียดโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

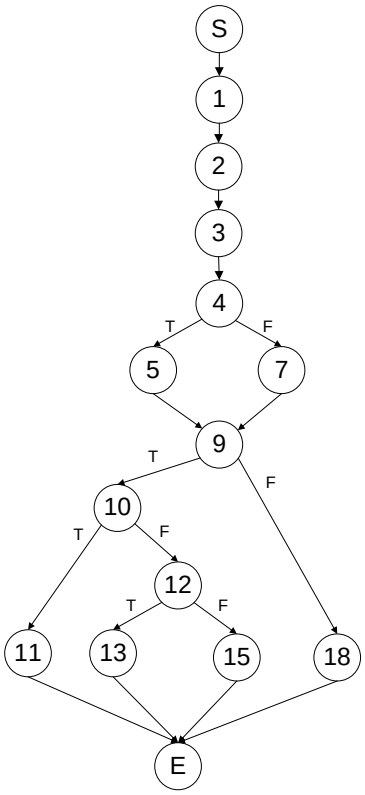
ชื่อโปรแกรม	Triangle Problem
รายละเอียดโปรแกรม	คำนวณหาประเภทของรูปสามเหลี่ยมจากข้อมูลนำเข้า 3 ตัวที่เป็นจำนวนเต็ม (Integer)
จำนวน input	3
ประเภท input	จำนวนเต็ม (Integer)
ประเภท output	ประเภทรูปสามเหลี่ยม (String) - Not a Triangle - Equilateral - Scalene - Isosceles

Source Code

```
1    Dim a, b, c As Integer
2    Dim Is_Triangle As Boolean
3    Dim output As String
4    If (a < b + c) And (b < a + c) And (c < b + a) Then
5        Is_Triangle = True
6    Else
7        Is_Triangle = False
8    End If
9    If Is_Triangle Then
10       If (a = b) And (b = c) Then
11            output = "Equilateral"
12       ElseIf (a <> b) And (b <> c) And (a <> c) Then
13            output = "Scalene"
14       Else
15            output = "Isosceles"
16       End If
17   Else
18       output = "Not a Triangle"
19   End If
```

ภาพที่ ก-3 รหัสต้นฉบับโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Control Flow Graph



ภาพที่ ก-4 กราฟควบคุมการไหลโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Target Path

ตารางที่ ก-7 เส้นทางสำหรับการสร้างกรณีทดสอบโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Path 1	S	1	2	3	4T	5	9F	18	E	
Path 2	S	1	2	3	4T	5	9T	10T	11	E
Path 3	S	1	2	3	4T	5	9T	10F	12T	13	E
Path 4	S	1	2	3	4T	5	9T	10F	12F	15	E
Path 5	S	1	2	3	4F	7	9F	18	E	
Path 6	S	1	2	3	4F	7	9T	10T	11	E
Path 7	S	1	2	3	4F	7	9T	10F	12T	13	E
Path 8	S	1	2	3	4F	7	9T	10F	12F	15	E

Genetic Algorithm Parameters

ตารางที่ ก-8 การกำหนดค่าตั้งต้นโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Number of Parameters	3
Number of Generations	100
Crossover Probability	1.0
Mutation Probability	1.0
Chromosome Type	Integer

Test Case Generation

ตารางที่ ก-9 ผลการสร้างกรณีทดสอบโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Test Case	**Target Path**	**Result**	**Input Result**		
			a	**b**	**c**
1	S 1 2 3 4T 5 9F 18 E	Not Found	-	-	-
2	S 1 2 3 4T 5 9T 10T 11 E	Found	9	9	9
3	S 1 2 3 4T 5 9T 10F 12T 13 E	Found	4	2	3
4	S 1 2 3 4T 5 9T 10F 12F 15 E	Found	3	3	2
5	S 1 2 3 4F 7 9F 18 E	Found	-5	-4	-8
6	S 1 2 3 4F 7 9T 10T 11 E	Not Found	-	-	-
7	S 1 2 3 4F 7 9T 10F 12T 13 E	Not Found	-	-	-
8	S 1 2 3 4F 7 9T 10F 12F 15 E	Not Found	-	-	-

Mutation Testing

ตารางที่ ก-10 สรุปผลการประเมินประสิทธิภาพกรณีทดสอบโปรแกรมคำนวณหาประเภทของรูปสามเหลี่ยม

Program	**Total Test Cases**	**Total Mutants**	**Killed Mutants**	**Mutation Score**
Triangle Problem	4	38	25	0.66

3. กรณีศึกษา การสร้างกรณีทดสอบโปรแกรมคำนวณหาค่ากลางของข้อมูล

Program Description

ตารางที่ ก-11 รายละเอียดโปรแกรมคำนวณหาค่ากลางของข้อมูล

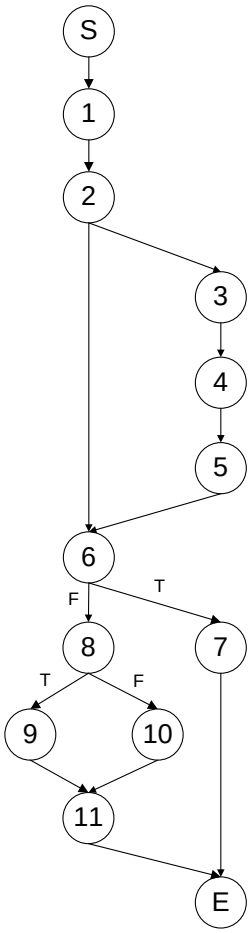
ชื่อโปรแกรม	Find Middle Character
รายละเอียดโปรแกรม	คำนวณหาค่ากลางของข้อมูลจากข้อมูลนำเข้า 3 ตัวที่เป็นรหัสอักขระ (Character)
จำนวน input	3
ประเภท input	รหัสอักขระ (Character)
ประเภท output	รหัสอักขระ (Character)

Source Code

```
1     Dim a , b , c , temp , mid As Char
2     If b > a Then
3          temp = a
4          a = b
5          b = temp
      End If
6     If c > a Then
7          mid = a
      else
8          If c > b Then
9               mid = c
           Else
10              mid = b
11         End If
      End If
```

ภาพที่ ก-5 รหัสต้นฉบับโปรแกรมคำนวณหาค่ากลางของข้อมูล

Control Flow Graph



ภาพที่ ก-6 กราฟควบคุมการไหลโปรแกรมคำนวณหาค่ากลางของข้อมูล

Target Path

ตารางที่ ก-12 เส้นทางสำหรับการสร้างกรณีทดสอบโปรแกรมคำนวณหาค่ากลางของข้อมูล

Path 1	S	1	2T	3	4	5	6T	7	E		
Path 2	S	1	2T	3	4	5	6F	8T	9	11	E
Path 3	S	1	2T	3	4	5	6F	8F	10	11	E
Path 4	S	1	2F	6T	7	E					
Path 5	S	1	2F	6F	8T	9	11	E			
Path 6	S	1	2F	6F	8F	10	11	E			

Genetic Algorithm Parameters

ตารางที่ ก-13 การกำหนดค่าตั้งต้นโปรแกรมคำนวณหาค่ากลางของข้อมูล

Number of Parameters	3
Number of Generations	100
Crossover Probability	1.0
Mutation Probability	1.0
Chromosome Type	Character

Test case Generation

ตารางที่ ก-14 ผลการสร้างกรณีทดสอบโปรแกรมคำนวณหาค่ากลางของข้อมูล

Test Case	**Target Path**	**Result**	**Input Result**		
			a	**b**	**c**
1	S 1 2T 3 4 5 6T 7 E	Found	U	c	l
2	S 1 2T 3 4 5 6F 8T 9 11 E	Found	Y	y	l
3	S 1 2T 3 4 5 6F 8F 10 11 E	Found	P	n	‘ ‘
4	S 1 2F 6T 7 E	Found	N	I	V
5	S 1 2F 6F 8T 9 11 E	Found	q	J	i
6	S 1 2F 6F 8F 10 11 E	Found	‘ ‘	‘ ‘	‘ ‘

ตารางที่ ก-15 สรุปผลการประเมินประสิทธิภาพกรณีทดสอบโปรแกรมคำนวณหาค่ากลางของข้อมูล

Program	**Total Test Cases**	**Total Mutants**	**Killed Mutants**	**Mutation Score**
Find Middle Character	6	21	21	1

4. กรณีศึกษา การสร้างกรณีทดสอบโปรแกรมตรวจสอบข้อความ Palindrome

Program Description

ตารางที่ ก-16 รายละเอียดโปรแกรมตรวจสอบข้อความ Palindrome

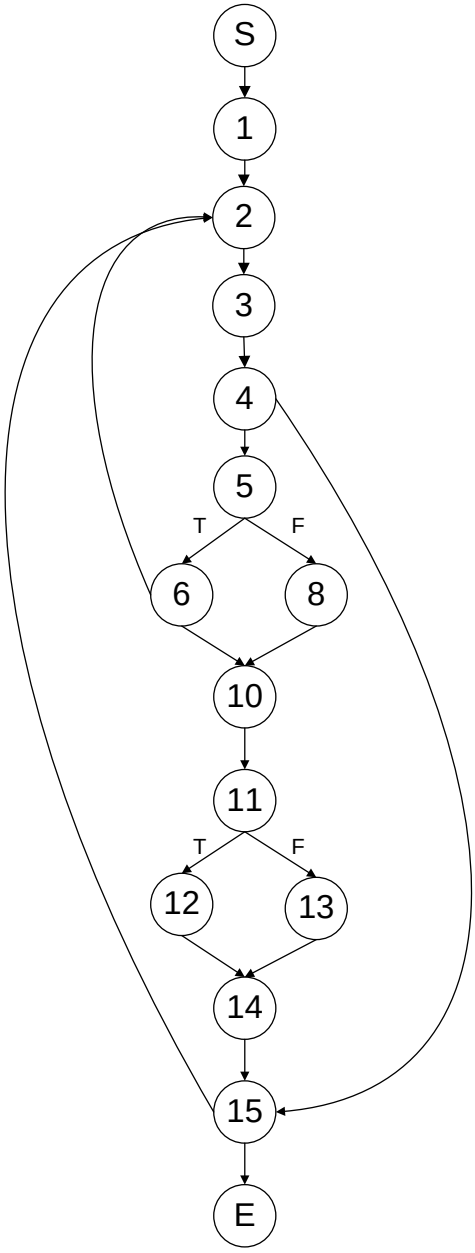
ชื่อโปรแกรม	Palindrome
รายละเอียดโปรแกรม	คำนวณหาคำจากข้อความที่สะกดเหมือนเดิมทั้งจากหน้าไปหลังหรือหลังไปหน้า
จำนวน input	1
ประเภท input	ข้อความ (String)
ประเภท output	ข้อความ (String)

Source Code

```
1     Dim theInputString As String
2     For i As Integer = 1 To theInputString.Length - 1
3          k = i + 1
           j = i - 1
4          While j >= 0 AndAlso k < theInputString.Length
5               If theInputString(j) <> theInputString(k) Then
6                    Exit While
7               Else
8                    j -= 1
                     k += 1
9               End If
10              aPalindrome = theInputString.Substring(j + 1, k - j - 1)
11              If aPalindrome.Length > aLongestPalindrome.Length Then
12                   aLongestPalindrome = aPalindrome
13              End If
14         End While
15    Next
```

ภาพที่ ก-7 รหัสต้นฉบับโปรแกรมตรวจสอบข้อความ Palindrome

Control Flow Graph



ภาพที่ ก-8 กราฟควบคุมการไหลโปรแกรมตรวจสอบข้อความ Palindrome

Target Path

ตารางที่ ก-17 เส้นทางสำหรับการสร้างกรณีทดสอบโปรแกรมตรวจสอบข้อความ Palindrome

Path 1	S 1 2 3 4 5T 6 15 2 3 15 E
Path 2	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 15 E
Path 3	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E
Path 4	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E
Path 5	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E
Path 6	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E
Path 7	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E
Path 8	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E
Path 9	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 15 2 3 15 E
Path 10	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E

Genetic Algorithm Parameters

ตารางที่ ก-18 การกำหนดค่าตั้งต้นโปรแกรมตรวจสอบข้อความ Palindrome

Number of Parameters	1
Number of Generations	100
Crossover Probability	1.0
Mutation Probability	1.0
Chromosome Type	String

Test case Generation

ตารางที่ ก-19 ผลการสร้างกรณีทดสอบโปรแกรมตรวจสอบข้อความ Palindrome

Test Case	Target Path	Result	Input Result
1	S 1 2 3 4 5T 6 15 2 3 15 E	Found	XLJc8WcV1y
2	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 15 E	Found	tYpVpkK5xQ
3	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E	Found	SjSLmRnVrS
4	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E	Found	7fXgbiZuZt
5	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 15 2 3 15 E	Found	0dlMCNFQXQ
6	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5T 6 15 2 3 15 E	Found	rnYnrBE6El
7	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	eeNnvnMnkJ
8	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	iSOSOEq9EH
9	S 1 2 3 4 5T 6 15 2 3 4 5F 8 10 11T 12 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 4 5F 8 10 11F 13 14 15 2 3 15 E	Found	wNKQ8Q8PcP
10	S 1 2 3 4 5F 8 10 11T 12 14 15 2 3 4 5F 8 10 11F 13 14 4 5T 6 15 2 3 15 E	Found	ororOadjeJ

Mutation Testing

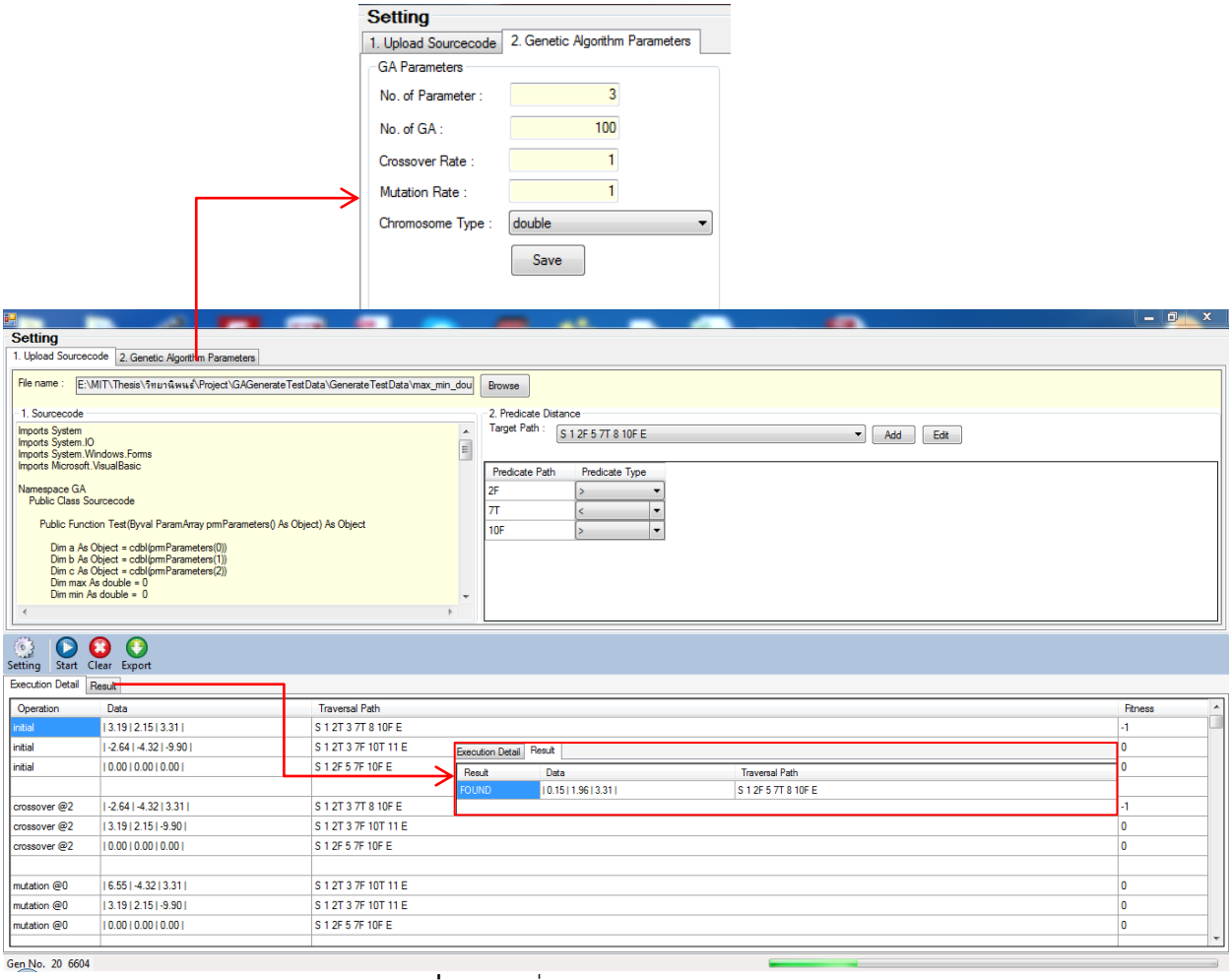
ตารางที่ ก-20 สรุปผลการประเมินประสิทธิภาพกรณีทดสอบโปรแกรมตรวจสอบข้อความ Palindrome

Program	Total Test Cases	Total Mutants	Killed Mutants	Mutation Score
Palindrome	10	21	16	0.76

ภาคผนวก ข

คู่มือการใช้งานเครื่องมือการสร้างกรณีทดสอบ

เครื่องมือสำหรับการสร้างกรณีทดสอบที่พัฒนาขึ้น แสดงดังภาพที่ ข-1 และมีรายละเอียดหน้าจอการทำงานในส่วนต่างๆ ดังนี้

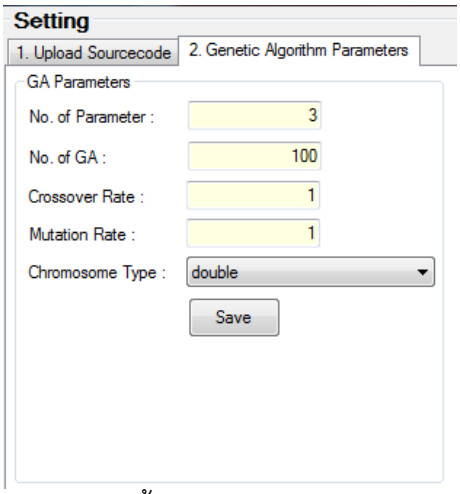


ภาพที่ ข-1 เครื่องมือการสร้างกรณีทดสอบ

2. หน้าจอการตั้งค่า Genetic Algorithm Parameters

เป็นส่วนที่ใช้สำหรับการกำหนดค่าพารามิเตอร์ในกระบวนการขั้นตอนวิธีเชิงพันธุกรรม แสดงดังภาพที่ ข-2 ประกอบไปด้วย

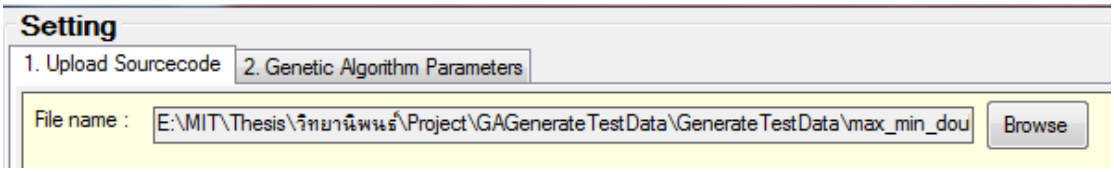
1) No. of Parameter คือ จำนวนข้อมูลนำเข้าที่ต้องการหา
2) No. of GA คือ จำนวน Generation ของกระบวนการขั้นตอนวิธีเชิงพันธุกรรม ซึ่งก็คือจำนวนรอบของการทำงานของกระบวนการขั้นตอนวิธีเชิงพันธุกรรม
3) Crossover Rate คือ อัตราความน่าจะเป็นในการไขว้เปลี่ยน สามารถกำหนดค่าระหว่าง 0 ถึง 1
4) Mutation Rate คือ อัตราความน่าจะเป็นในการกลายพันธุ์ สามารถกำหนดค่าระหว่าง 0 ถึง 1
5) Chromosome Type คือ ชนิดของข้อมูลนำเข้า



ภาพที่ ข-2 หน้าจอการตั้งค่า Genetic Algorithm Parameters

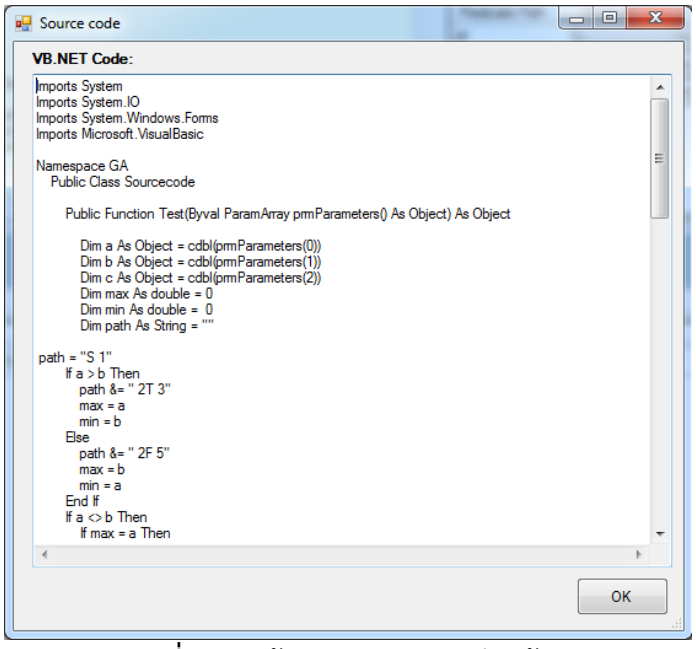
3. หน้าจอการอัพโหลดไฟล์ (Upload File)

เป็นส่วนที่ใช้ในการเลือกไฟล์ซึ่งอยู่ในรูปแบบของไฟล์ข้อความ (Text File) เพื่อใช้อัพโหลดเข้าสู่ระบบ โดยสามารถ Browse เพื่อเลือกที่ตั้งของไฟล์ได้ หน้าจอการอัพโหลดไฟล์แสดงดังภาพที่ ข-3



ภาพที่ ข-3 หน้าจอการอัพโหลดไฟล์ (Upload File)

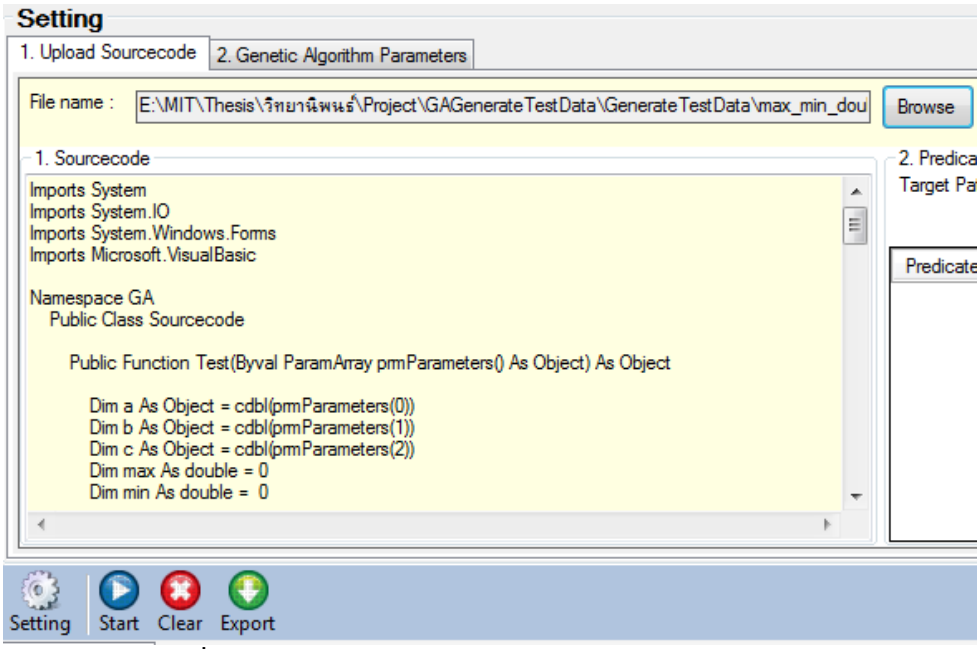
เมื่อทำการเลือกไฟล์ที่ต้องการอัพโหลดเสร็จ ระบบจะปรากฏหน้าจอใหม่ซึ่งแสดงรายละเอียดข้อมูลเพื่อยืนยันไฟล์ที่ต้องการเลือกโดยกดปุ่ม OK ดังภาพที่ ข-3



ภาพที่ ข-4 หน้าจอแสดงรายละเอียดข้อมูล

4. หน้าจอแสดงรหัสโปรแกรมต้นฉบับ (Source Code)

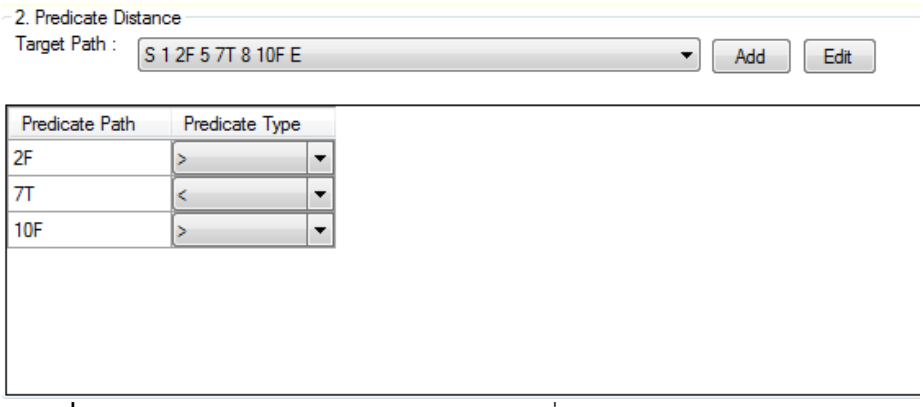
เมื่อยืนยันการเลือกไฟล์ หน้าจอแสดงไฟล์โปรแกรมดังภาพที่ ข-5



ภาพที่ ข-5 หน้าจอแสดงรหัสโปรแกรมต้นฉบับ (Source Code)

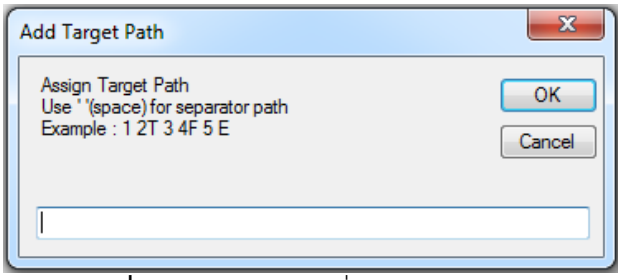
5. หน้าจอการกำหนดเส้นทางเป้าหมายที่ต้องการใช้ในการหากรณีทดสอบ

เป็นส่วนที่ใช้กำหนดเส้นทางเป้าหมายที่ต้องการหากรณีทดสอบ โดยเมื่อระบุเส้นทางเป้าหมาย ระบบจะแสดงเส้นทางที่เป็นคำสั่งเงื่อนไขการตัดสินใจ เพื่อให้ผู้ใช้ทำการระบุประเภทของเงื่อนไข (Predicate Type) ดังภาพที่ ข-6

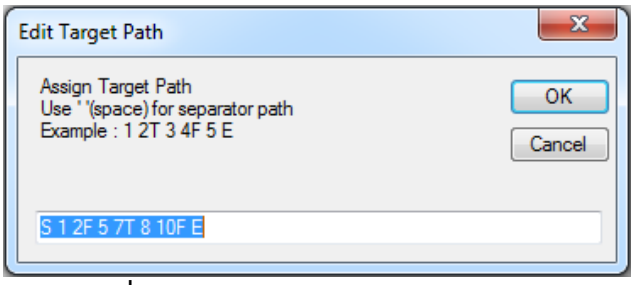


ภาพที่ ข-6 หน้าจอการกำหนดเส้นทางเป้าหมายที่ต้องการใช้ในการหากรณีทดสอบ

สามารถเพิ่มเส้นทางเป้าหมายใหม่หรือแก้ไขเส้นทางเป้าหมายที่เลือกได้ โดยกดที่ปุ่ม Add หรือ Edit โดยจะแสดงหน้าจอดังภาพที่ ข-7 และ ข-8 ตามลำดับ



ภาพที่ ข-7 หน้าจอการเพิ่มเส้นทางเป้าหมาย



ภาพที่ ข-8 หน้าจอการแก้ไขเส้นทางเป้าหมาย

6. ปุ่มการทำงาน

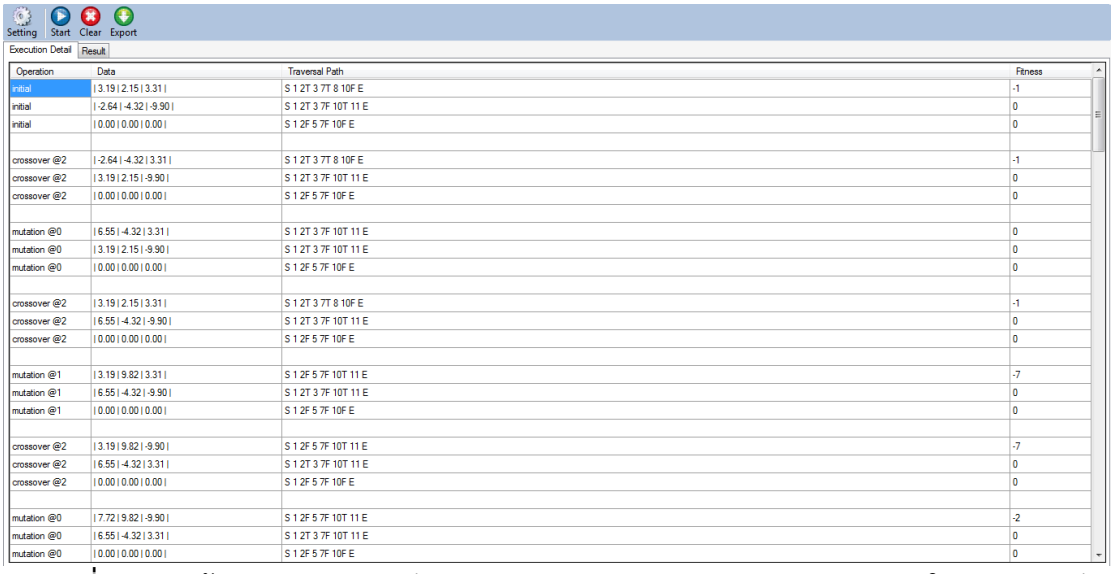
ภาพที่ ข-9 ปุ่มการทำงาน

1. **Setting** : ซ่อน / แสดง ส่วนของการตั้งค่าการใช้งาน
2. **Start** : เริ่มการประมวลผล
3. **Clear** : ล้างหน้าจอแสดงผลการทำงานของโปรแกรม
4. **Export** : ส่งออกผลการทำงานของโปรแกรมในรูปแบบไฟล์ MS Excel

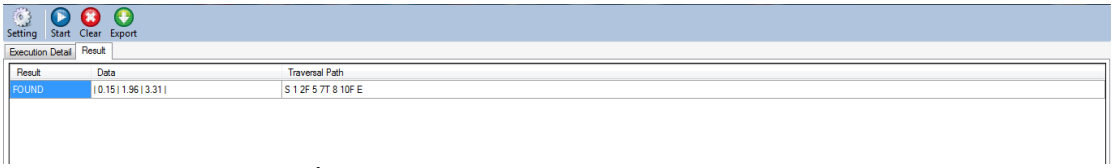
7. หน้าจอแสดงผลการทำงานของโปรแกรม

หน้าจอส่วนนี้จะแสดงรายละเอียดการทำงานของขั้นตอนวิธีเชิงพันธุกรรม โดยจะแสดงรายละเอียดต่างๆ ดังนี้

5) Operation คือ กระบวนการในขั้นตอนวิธีเชิงพันธุกรรม ได้แก่ Initial , Crossover และ Mutation
6) Data คือ ชุดของข้อมูลที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ
7) Traversal Path คือ เส้นทางที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ
8) Fitness คือ ค่าความเหมาะสมที่เกิดขึ้นระหว่างการค้นหากรณีทดสอบ



ภาพที่ ข-10 หน้าจอแสดงรายละเอียดการการปฏิบัติการดำเนินการทางพันธุกรรมในการหากรณีทดสอบสำหรับเส้นทางเป้าหมายที่ระบุ (Execution Detail)



ภาพที่ ข-11 หน้าแสดงผลการหากรณีทดสอบ (Result)

ภาคผนวก ค

รายงานผลงานตีพิมพ์

1.1 รายการผลงานที่ตีพิมพ์

นันทนี ช่วยชู และ สุภาภรณ์ กานต์สมเกียรติ. “การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม.” การประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 10 (NCCIT 2014), 2557.

Nuntanee Chuaychoo and Supaporn Kansomkeat. “Path Coverage Test Case Generation Using Genetic Algorithms.” *2nd Advancement on Information Technology International Conference*, 2015.

1.2 รายการการประชุมวิชาการ

ชื่อเรื่องผลงานที่นำเสนอ	การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม
ชื่อการประชุม	การประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 10 (NCCIT 2014)
วันเดือนปีและสถานที่จัดประชุม	วันที่ 8 – 9 พฤษภาคม 2557 ณ โรงแรมอังสนา ลากูนา จังหวัดภูเก็ต
หน่วยงานที่จัดประชุม	คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ชื่อเรื่องผลงานที่นำเสนอ	Path Coverage Test Case Generation Using Genetic Algorithms
ชื่อการประชุม	2nd Advancement on Information Technology International Conference (ADVCIT 2015)
วันเดือนปีและสถานที่จัดประชุม	วันที่ 3 – 5 ธันวาคม 2558 ณ โรงแรมดีวาน่า พลาซ่า จังหวัดกระบี่
หน่วยงานที่จัดประชุม	Malaysia Technical Scientist Association (MALTESAS) , Universiti Malaysia Perlis (UNIMAP)

1.3 สำเนาต้นฉบับที่ได้รับการยินยอมจากผู้พิมพ์ผลงาน

1.3.1 นันทนี ช่วยชู และ สุภาภรณ์ กานต์สมเกียรติ. “การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม.” การประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 10 (NCCIT 2014), 2557.

The Tenth National Conference on Computing and Information Technology NCCIT2014

การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม
Test Case Generation Using Genetic Algorithms

นันทนี ช่วยชู(Nuntanee Chuaychoo)[1] และ สุภาภรณ์ กานต์สมเกียรติ (Supaporn Kansomkeat)[2]
[1]สาขาวิชาการจัดการเทคโนโลยีสารสนเทศ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์
[2]ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์
[1]nun-ta-nee@hotmail.com, [2]supaporn.k@psu.ac.th

บทคัดย่อ

การทดสอบซอฟต์แวร์ เป็นขั้นตอนหนึ่งที่สำคัญในกระบวนการผลิตซอฟต์แวร์ การทดสอบช่วยให้ซอฟต์แวร์ที่พัฒนามีความถูกต้อง และมีความน่าเชื่อถือเพิ่มมากขึ้น การสร้างกรณีทดสอบเป็นอีกองค์ประกอบหนึ่งที่มีความสำคัญในกระบวนการทดสอบซอฟต์แวร์ คุณภาพของการทดสอบจะขึ้นอยู่กับประสิทธิภาพของกรณีทดสอบ บทความนี้นำเสนอวิธีการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม และได้นำขั้นตอนการสร้างกรณีทดสอบที่ได้นำเสนอไปประยุกต์ใช้กับกรณีศึกษา ผลลัพธ์ที่ได้พบว่า กรณีทดสอบที่สร้างขึ้นตามวิธีการที่นำเสนอนี้มีความเหมาะสมสำหรับการทดสอบซอฟต์แวร์

คำสำคัญ : การทดสอบซอฟต์แวร์, ขั้นตอนวิธีเชิงพันธุกรรม, การสร้างกรณีทดสอบ

Abstract

Software Testing is an important step in the software process. It makes the developed software more accurate and reliable. Generating test cases is a key element in the process of software testing. The quality of testing depends on the effectiveness of test. This paper presents test case generation method using genetic algorithms. The proposed method was applied on a case study. The result shows that the generated test cases are appropriate for testing software.

Keywords : Software Testing, Genetic Algorithms, Test Case Generation

1. บทนำ

ในปัจจุบันความต้องการของการใช้คอมพิวเตอร์และเทคโนโลยีสารสนเทศยังคงมีเพิ่มขึ้นเรื่อยๆ ดังนั้นจึงมีความต้องการซอฟต์แวร์เพื่อใช้งานเฉพาะต่าง ๆ อีกมากมาย การทดสอบซอฟต์แวร์ (Software Testing) [1] เป็นขั้นตอนที่สำคัญในกระบวนการพัฒนาซอฟต์แวร์ที่มีบทบาทและมีความสำคัญมาก การทดสอบเป็นกิจกรรมที่จัดทำขึ้นเพื่อปรับปรุงคุณภาพของซอฟต์แวร์ ค้นหาข้อบกพร่องและลดข้อผิดพลาดจากการทำงานของซอฟต์แวร์ให้เหลือน้อยที่สุด

การกำหนดกรณีทดสอบเป็นสิ่งหนึ่งที่สำคัญในกระบวนการทดสอบซอฟต์แวร์ ขั้นตอนการสร้างกรณีทดสอบที่ดี จะมีส่วนทำให้การทดสอบมีประสิทธิภาพ กรณีทดสอบที่มีประสิทธิภาพจะต้องครอบคลุมคุณลักษณะต่างๆ ของระบบและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไป

ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithms) เป็นวิธีการค้นหาคำตอบที่เหมาะสมโดยใช้หลักการคัดเลือกแบบธรรมชาติจากการจำลองแนวคิดวิวัฒนาการของสิ่งมีชีวิต การนำขั้นตอนวิธีทางพันธุกรรมมาประยุกต์ใช้ในการสร้างกรณีทดสอบสำหรับกระบวนการทดสอบซอฟต์แวร์ จึงเป็นเรื่องที่น่าสนใจ เนื่องจากหลักการทำงานของขั้นตอนวิธีทางพันธุกรรม มีความสอดคล้องกับขั้นตอนการค้นหากรณีทดสอบและมีการปรับเปลี่ยนกรณีทดสอบ จนกระทั่งได้กรณีทดสอบที่เหมาะสมที่สุด สามารถช่วยให้ผู้พัฒนาระบบหรือผู้ทดสอบระบบสามารถทำงานได้ง่ายขึ้น ช่วยลดเวลาและต้นทุนในขั้นตอนการทดสอบซอฟต์แวร์

บทความนี้นำเสนอการสร้างกรณีทดสอบที่สามารถครอบคลุมการทำงานและเงื่อนไขการทำงานของโปรแกรมโดยประยุกต์ใช้ขั้นตอนวิธีเชิงพันธุกรรม

บทความนี้ประกอบด้วยหัวข้อต่างๆ ดังต่อไปนี้คือ หัวข้อที่ 2 เสนอทฤษฎีที่เกี่ยวข้อง หัวข้อที่ 3 เสนอวิธีการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม หัวข้อที่ 4 เสนอกรณีศึกษา และหัวข้อที่ 5 สรุปผลและงานในอนาคต

2. ทฤษฎีที่เกี่ยวข้อง

ในส่วนนี้นำเสนอเกี่ยวกับทฤษฎีที่เกี่ยวข้อง ประกอบไปด้วยการทดสอบซอฟต์แวร์และขั้นตอนวิธีเชิงพันธุกรรม

2.1 การทดสอบซอฟต์แวร์ (Software Testing)

การทดสอบเป็นปัจจัยสำคัญปัจจัยหนึ่งของการประกันคุณภาพซอฟต์แวร์ดังนั้นในกระบวนการพัฒนาซอฟต์แวร์จึงต้องมีขั้นตอนการทดสอบซอฟต์แวร์ สำหรับเทคนิคในการทดสอบซอฟต์แวร์ (Software Testing Technique) ที่นิยมใช้กันแพร่หลายได้แก่

1. Black-Box Testing คือการทดสอบการทำงานของซอฟต์แวร์ที่ไม่สนใจกลไกภายในของระบบ การทำการทดสอบเน้นเพื่อการตรวจสอบผลการทำงานของระบบในแต่ละหน้าที่ตามข้อกำหนดความต้องการ (Requirement Specification) ว่าถูกต้องตามความต้องการหรือไม่

2. White-Box Testing คือการทดสอบการทำงานของซอฟต์แวร์ ซึ่งพิจารณากลไกภายในของระบบโดยจะมุ่งเน้นพิจารณาโครงสร้างภายใน

2.2 ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithms)

ขั้นตอนวิธีเชิงพันธุกรรม เป็นกระบวนการหนึ่งที่อาศัยทฤษฎีการวิวัฒนาการของสิ่งมีชีวิต นำเสนอโดย John Holland [2] สามารถนำมาใช้เพื่อค้นหาคำตอบที่ดีขึ้นและเหมาะสมมากขึ้นโดยผ่านการเรียนรู้ด้วยตัวเอง เปรียบเสมือนการวิวัฒนาการของสิ่งมีชีวิตที่สามารถวิวัฒนาการเพื่อการดำรงชีวิตอยู่ กระบวนการทำงานของขั้นตอนวิธีเชิงพันธุกรรมจะเริ่มจากการสุ่มโครโมโซมขึ้นมาจำนวนหนึ่งเพื่อสร้างเป็นกลุ่มประชากรต้นกำเนิด (Initial Population) และทำการประเมินค่าความเหมาะสม (Fitness Evaluation) จากนั้นจะคัดเลือกประชากร (Selection) มาจำนวนหนึ่งเพื่อใช้เป็นต้นกำเนิดทางสายพันธุ์หรือเป็นกลุ่มประชากรรุ่นพ่อแม่ จากนั้นจะนำประชากรเหล่านี้เข้าสู่กระบวนการปฏิบัติการทางสายพันธุ์ (Genetic Operation) ได้แก่ การไขว้เปลี่ยน (Crossover) และการกลายพันธุ์ (Mutation) ซึ่งจะทำให้เกิดกลุ่มประชากรที่เป็นรุ่นลูก และเมื่อนำประชากรกลุ่มนี้ไปประเมินค่าความเหมาะสมแล้วพบคำตอบที่เหมาะสมกับปัญหา กระบวนการทำงานก็จะจบลง แต่หากคำตอบที่ได้ยังไม่เหมาะสม ก็จะดำเนินต่อไปโดยการนำประชากรรุ่นลูกที่ได้ไปแทนที่ (Replacement) ประชากรรุ่นพ่อแม่ เมื่อทำการแทนที่เสร็จแล้วก็จะนำประชากรกลุ่มใหม่ที่ได้กลับเข้าสู่กระบวนการคัดเลือกใหม่อีกครั้ง ซึ่งกระบวนการเหล่านี้จะดำเนินการซ้ำไปเรื่อยๆ จนกระทั่งได้ประชากรกลุ่มที่เหมาะสม

3. การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

ขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม แสดงดังภาพที่ 1 ประกอบด้วย 2 ขั้นตอนหลัก คือ
1) การสร้างกราฟควบคุมการไหลและระบุเส้นทางเป้าหมาย (Generating Control Flow Graph and Specifying Target Paths)
2) การสร้างกรณีทดสอบ (Generating Test Cases)

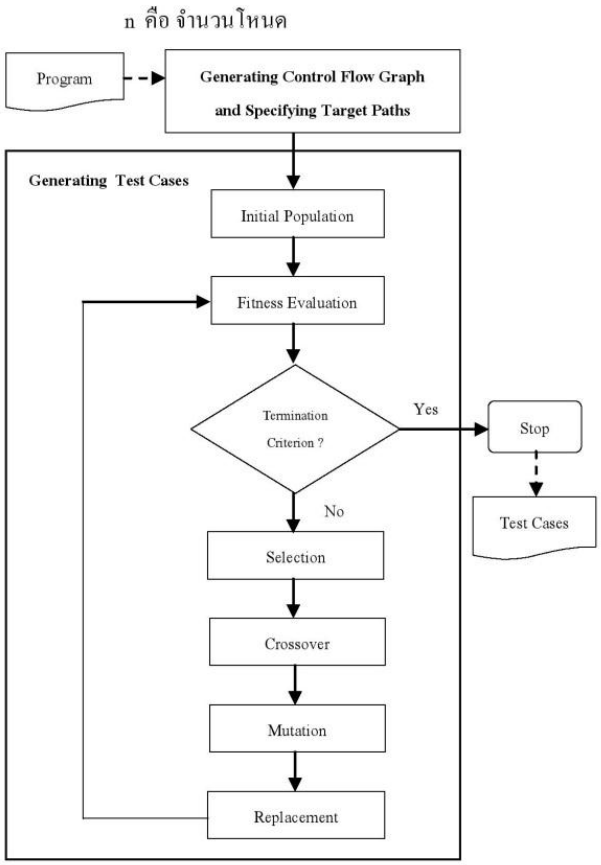
3.1 การสร้างกราฟควบคุมการไหลและระบุเส้นทางเป้าหมาย

การสร้างกราฟควบคุมการไหล พิจารณาจากรหัสต้นทาง (Source Code) ของโปรแกรมที่ต้องการทดสอบ กราฟควบคุมการไหลสร้างขึ้นตามหลักการที่ได้นำเสนอโดย Amman และ Offutt [3] ภาพที่ 2 แสดงตัวอย่างรหัสต้นทางของโปรแกรม Triangle ที่ต้องการทดสอบ ภาพที่ 3 แสดงกราฟควบคุมการไหลของโปรแกรม Triangle

กราฟควบคุมการไหลจากขั้นตอนข้างต้น จะถูกนำมาใช้เพื่อระบุเส้นทางเป้าหมาย โดยจะคำนวณจำนวนเส้นทางจากการคำนวณค่าความซับซ้อนไซโคลมาติก (Cyclomatic Complexity) [4] ซึ่งใช้ในการวัดความซับซ้อนของโปรแกรมเป็นตัวบ่งชี้ว่าโปรแกรมนั้นมีความยากต่อการบำรุงรักษา และการทดสอบมากน้อยเพียงใด โดยสามารถคำนวณได้ดังนี้

$$V(G) = e-n+2 \quad (1)$$

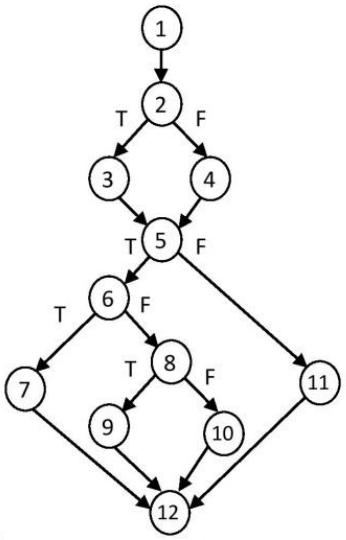
โดยที่ V(G) คือ ค่าความซับซ้อนไซโคลมาติก
e คือ จำนวนเส้นเชื่อมต่อระหว่างโหนด



ภาพที่ 1 : แผนภาพขั้นตอนการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม

```
1    Input (a,b,c) as Integer
2    If (a<b + c) And (b<a + c) And (c<b + a) Then
3       Is_Triangle = True
     Else
4       Is_Triangle = False
     End If
5    If Is_Triangle Then
6       If (a = b) And (b = c) Then
7             output = "Equilateral"
8    ElseIf (a<>b) And (b<>c) And (a<>c) Then
9             output = "Scalene"
        Else
10            output = "Isosceles"
        End If
     Else
11      output = "Not a Triangle"
12   End If
```

ภาพที่ 2 : แสดงตัวอย่างรหัสต้นทางของโปรแกรม Triangle



ภาพที่ 3 : แสดงกราฟควบคุมการไหลของโปรแกรม Triangle

3.2 การสร้างกรณีทดสอบ

การสร้างกรณีทดสอบถูกดำเนินการโดยใช้กระบวนการขั้นตอนวิธีเชิงพันธุกรรม ประกอบด้วยขั้นตอนดังนี้

ขั้นตอนที่ 1 การสร้างกลุ่มประชากรต้นกำเนิด (Initial Population)

กลุ่มประชากรต้นกำเนิดจะถูกสร้างขึ้นโดยใช้วิธีการแบ่งส่วนที่สมดุล (Equivalence Partitioning) ข้อมูลนำเข้าของโปรแกรมที่ต้องการทดสอบแต่ละตัวจะถูกแบ่งส่วนออกเป็นกลุ่มๆ ประชากรต้นกำเนิดหนึ่งประชากร ได้จากการนำส่วนใดส่วนหนึ่งของข้อมูลนำเข้าทุกตัวมารวมกัน ดังนั้นหนึ่งประชากรต้นกำเนิดหมายถึง หนึ่งชุดข้อมูลทดสอบ ซึ่งจำนวนโครโมโซมของหนึ่งประชากรจะเท่ากับจำนวนตัวแปรของหนึ่งชุดข้อมูลทดสอบ

ขั้นตอนที่ 2 การประเมินค่าความเหมาะสม (Fitness Evaluation)

การประเมินค่าความเหมาะสมจะพิจารณาจากการประเมินค่าความครอบคลุมของชุดข้อมูลที่ใช้ในการทดสอบ สำหรับบทความนี้ การประเมินค่าความครอบคลุมใช้วิธีการคำนวณหาค่า distance ของคำสั่งที่เป็นเงื่อนไขในการทำงานของโปรแกรม ตามทฤษฎี Korel's distance function [5] ดังแสดงในตารางที่ 1

ตารางที่ 1 : Korel's distance function

No	Predicate	Distance if path taken
1	A=B	ABS(A-B)
2	A≠B	K
3	A<B	(A-B)+K
4	A≤B	(A-B)
5	A>B	(B-A)+K
6	A≥B	(B-A)
7	X OR Y	MIN(Distance(X),Distance(Y))
8	X AND Y	(Distance(X) + Distance(Y)

ค่าความเหมาะสม สามารถคำนวณจากค่าผลต่างระหว่าง distance ดังนี้

$$\text{Fitness function} = \{ \text{dist(TG)} \} - \{ \text{dist(TR)} \} \quad (2)$$

โดยที่ dist(TG) คือ distance ของเส้นทางเป้าหมาย (Target Path)

dist(TR) คือ distance ของเส้นทางการเดินทางที่เกิดจากชุดข้อมูลทดสอบ (Traversal Path)

ค่าความเหมาะสมที่ได้ สามารถประเมินค่าความครอบคลุมได้ ดังนี้

$$\text{Fitness function} = \begin{cases} 0 & ;\text{covered} \\ \text{otherwise} & ;\text{not covered} \end{cases}$$

กรณีที่ค่าความเหมาะสม มีค่าเท่ากับศูนย์ แสดงว่าชุดข้อมูลทดสอบนั้นสามารถใช้เป็นกรณีทดสอบสำหรับเส้นทางเป้าหมายที่กำลังพิจารณา ในกรณีที่ค่าความเหมาะสม มีค่าไม่เท่ากับศูนย์ แสดงว่าชุดข้อมูลทดสอบนั้นยังไม่ครอบคลุมเส้นทางเป้าหมาย

เส้นทางเป้าหมายต่างๆ จะถูกตรวจสอบ โดยจะทำการตรวจสอบว่าชุดข้อมูลต่างๆ ครอบคลุมทุกเส้นทางเป้าหมายหรือไม่ กรณีที่ครอบคลุมทุกเส้นทางจะสิ้นสุดการทดสอบ

ขั้นตอนที่ 3 การคัดเลือก (Selection)

การคัดเลือกจะกระทำการคัดเลือกชุดข้อมูลเพื่อนำมาเป็นประชากรรุ่นพ่อแม่ โดยเลือกชุดข้อมูลที่ให้ค่าความเหมาะสมน้อย ซึ่งหมายถึงชุดข้อมูลที่ให้ค่าผลต่างระหว่าง distance น้อยที่สุด

ขั้นตอนที่ 4 การไขว้เปลี่ยน (Crossover)

การไขว้เปลี่ยนจะดำเนินการโดยเลือกชุดข้อมูลที่ให้ค่าผลต่างระหว่าง distance น้อยที่สุดมา 2 ชุดข้อมูล และทำการไขว้เปลี่ยน โดยใช้วิธีการไขว้เปลี่ยนแบบจุดเดียว (Single-Point Crossover) ซึ่งจะทำให้ได้ชุดข้อมูลใหม่ 2 ชุดข้อมูล ตัวอย่างเช่น ชุดข้อมูลที่ถูกเลือกมาคือ $\{x_1, x_2, x_3, x_4\}$ และ $\{y_1, y_2, y_3, y_4\}$ จะทำการไขว้เปลี่ยนชุดข้อมูลทั้งสอง ณ ตำแหน่งที่ 4 ดังนั้นจะได้ชุดข้อมูลใหม่ คือ $\{x_1, x_2, x_3, \mathbf{y_4}\}$ และ $\{y_1, y_2, y_3, \mathbf{x_4}\}$

ขั้นตอนที่ 5 การกลายพันธุ์ (Mutation)

การกลายพันธุ์จะกระทำโดยเลือกชุดข้อมูลที่ให้ค่าผลต่างระหว่าง distance น้อยที่สุด เพื่อทำการกลายพันธุ์แบบจุดเดียว (Single-Point Mutation) ซึ่งจะทำให้ได้ชุดข้อมูลใหม่ 1 ชุดข้อมูล ตัวอย่างเช่น ชุดข้อมูลที่ถูกเลือกมาคือ $\{x_1, x_2, x_3, x_4\}$ จะทำการกลายพันธุ์ ณ ตำแหน่งที่ 2 ด้วยค่า x_m ดังนั้นจะได้ชุดข้อมูลใหม่ คือ $\{x_1, \mathbf{x_m}, x_3, x_4\}$

โดยที่ตำแหน่งหรือโครโมโซมที่จะกระทำการไขว้เปลี่ยนในขั้นตอนที่ 4 และการกลายพันธุ์ในขั้นตอนที่ 5 จะได้จากการคัดเลือกโดยวิธีการสุ่ม เนื่องจากจำนวนตัวแปรหรือโครโมโซมของชุดข้อมูลมีจำนวนไม่มาก ดังนั้นการดำเนินการไขว้เปลี่ยนและการกลายพันธุ์จึงมีการกระทำแบบจุดเดียว

ขั้นตอนที่ 6 การแทนที่ (Replacement)

ชุดข้อมูลชุดใหม่ที่ได้จากขั้นตอนที่ 4 และ 5 จะถูกนำไปแทนที่ข้อมูลที่เป็นประชากรรุ่นพ่อแม่ และนำข้อมูลชุดใหม่ไปประเมินค่าความเหมาะสมในขั้นตอนที่ 2 อีกครั้ง

การกระทำขั้นตอนที่ 1 – 6 จะถูกกระทำซ้ำๆ โดยจำนวนรุ่นของประชากรจะเท่ากับจำนวนรอบการทำซ้ำ โดยกำหนดให้มีการทำซ้ำสูงสุด 20 รอบ

4. กรณีศึกษา

บทความนี้ เลือกใช้โปรแกรม Triangle เพื่อแสดงการประยุกต์ใช้ขั้นตอนการหากรณีทดสอบที่ได้นำเสนอ

จากกราฟควบคุมการไหลของโปรแกรม Triangle ดังแสดงในภาพที่ 3 สามารถคำนวณหาค่าความซับซ้อนไซโคลมาติกได้เท่ากับ 5 ดังนั้นเส้นทางในการทดสอบโปรแกรมมี 5 เส้นทาง ซึ่งสามารถระบุเส้นทางเป้าหมายทั้ง 5 เส้นทางได้ดังตารางที่ 2

ตารางที่ 2 : เส้นทางสำหรับทดสอบโปรแกรม Triangle

Path	The description of target path
P1	1-2T-3-5T-6T-7-12
P2	1-2T-3-5T-6F-8T-9-12
P3	1-2T-3-5T-6F-8F-10-12
P4	1-2F-4-5F-11-12
P5	1-2T-3-5F-11-12

กลุ่มประชากรต้นกำเนิด ซึ่งเป็นกลุ่มของชุดข้อมูลเริ่มต้นจะถูกสร้างขึ้นโดยใช้วิธีแยกส่วนที่สมดุล (Equivalence Partitioning) สำหรับโปรแกรม Triangle มีข้อมูลนำเข้าเป็นจำนวนเต็ม 3 จำนวน (a , b , c) โดยจำนวนเต็มสามารถแบ่งส่วน (Partition) ของข้อมูลออกเป็น 3 กลุ่ม คือ มากกว่า, น้อยกว่า และเท่ากับศูนย์

การกำหนดกลุ่มของชุดข้อมูลเริ่มต้น จะกำหนดตามหลักการ Each Choice (EC) ที่ถูกนำเสนอโดย Amman และ Offutt [3] โดยหลักการนี้กำหนดให้แต่ละส่วนที่ถูกแบ่ง จะต้องถูกใช้อย่างน้อย 1 ครั้งในการทดสอบ ดังนั้น ชุดข้อมูลที่ใช้สำหรับโปรแกรม Triangle จึงประกอบด้วย 3 ชุดข้อมูล คือ {a<0,b<0,c<0} , {a=0,b=0,c=0} และ {a>0,b>0,c>0} จากนั้นทำการสุ่มค่าของข้อมูลที่ใช้สำหรับทดสอบแสดงดังตารางที่ 3

ตารางที่ 3 : ชุดข้อมูลที่เบื้องต้น สำหรับการทดสอบโปรแกรม Triangle

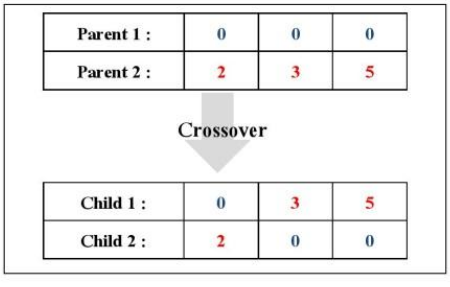
ชุดข้อมูล / ตัวแปร	a	b	c
1	-3	-2	-5
2	0	0	0
3	2	3	5

การประเมินค่าความครอบคลุมของชุดข้อมูลที่ใช้ในการทดสอบถูกกระทำโดยวิธีการคำนวณหาค่า distance ของคำสั่งที่เป็นเงื่อนไขตามทฤษฎี Korel's distance function โดยเส้นทางเป้าหมายแต่ละเส้นทางจะถูกนำมาพิจารณาเพื่อหาชุดข้อมูลที่เหมาะสมสำหรับการทดสอบเส้นทางนั้นๆ ในกรณีที่ยังไม่มีชุดข้อมูลที่เหมาะสมสำหรับเส้นทางใด เส้นทางนั้นจะถูกดำเนินการพิจารณาต่อไป

ตัวอย่างเช่น จากตารางที่ 2 กำหนดให้ P4 เป็นเส้นทางเป้าหมายสำหรับการหากรณีทดสอบ ดังนั้นคำสั่งที่เป็นเงื่อนไขในเส้นทาง P4 ประกอบด้วย {2F ,5F} พิจารณาชุดข้อมูลทั้ง 3 ในตารางที่ 3 นำแต่ละชุดข้อมูลมาคำนวณหาผลต่างระหว่าง distance ของเส้นทางการเดินทางของข้อมูลชุดนั้น กับเส้นทางเป้าหมาย P4 จะพบว่า ผลต่างระหว่าง distance ของเส้นทางการเดินทางของข้อมูลชุดที่ 1 กับเส้นทางเป้าหมาย P4 มีค่าเท่ากับศูนย์ แสดงว่าข้อมูลชุดที่ 1 มีความครอบคลุมเส้นทางเป้าหมาย ดังนั้นกระบวนการหากรณีทดสอบสำหรับเส้นทางเป้าหมาย P4 ก็จะจบลง

ตัวอย่างกำหนดให้ P1 เป็นเส้นทางเป้าหมายสำหรับการหากรณีทดสอบคำสั่งที่เป็นเงื่อนไขในเส้นทาง P1ประกอบด้วย {2T,5T,6T} เมื่อคำนวณหาผลต่างระหว่าง distance โดยใช้ชุดข้อมูลทั้ง 3 ในตารางที่ 3 จะพบว่าไม่มีข้อมูลชุดใดให้ค่าผลต่างระหว่าง distance เท่ากับศูนย์ ดังนั้นแสดงว่ายังไม่มีข้อมูลชุดใดครอบคลุมเส้นทางเป้าหมาย จึงต้องมีการดำเนินการต่อไปโดยการไขว้เปลี่ยนและการกลายพันธุ์

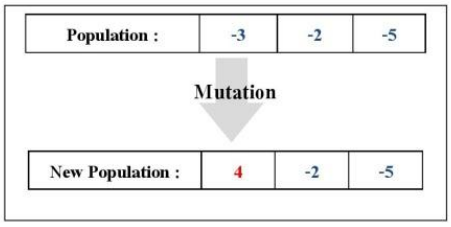
ในการไขว้เปลี่ยน จะเลือกชุดข้อมูลที่ทำให้ค่าผลต่างระหว่าง distance น้อยที่สุด 2 อันดับ และสุ่มเลือกตำแหน่งในการไขว้เปลี่ยนเพื่อให้ได้ข้อมูลชุดใหม่สำหรับการทดสอบในรอบต่อไป ตัวอย่างการไขว้เปลี่ยน แสดงดังภาพที่ 4



ภาพที่ 4 : การไขว้เปลี่ยนชุดข้อมูลแบบจุดเดียว (Single-Point

Crossover)

กรณีที่ทำการไขว้เปลี่ยนแล้ว แต่กรณีทดสอบยังไม่ครอบคลุม จะกระทำการกลายพันธุ์ โดยเลือกชุดข้อมูลที่ให้ค่าผลต่างระหว่าง distance น้อยที่สุด และสุ่มเลือกตำแหน่งในการกลายพันธุ์ ตัวอย่างการกลายพันธุ์แสดงดังภาพที่ 5



ภาพที่ 5 : การกลายพันธุ์ชุดข้อมูลแบบจุดเดียว (Single-Point Mutation)

นำชุดข้อมูลใหม่ที่ได้จากขั้นตอนที่ 4 และ 5 มาเป็นชุดข้อมูลใหม่สำหรับการหากรณีทดสอบในรอบถัดไป

กระบวนการค้นหาชุดข้อมูลสำหรับเส้นทางเป้าหมายจะถูกกระทำซ้ำๆ การทำงานจะจบลงเมื่อสามารถหาชุดข้อมูลสำหรับใช้เป็นกรณีทดสอบสำหรับทุกๆเส้นทางเป้าหมาย ตารางที่ 4 แสดงกรณีทดสอบที่ครอบคลุมเส้นทางเป้าหมายทั้งหมดสำหรับการทดสอบโปรแกรม Triangle

ตารางที่ 4 : กรณีทดสอบที่ครอบคลุมเส้นทางเป้าหมายสำหรับการทดสอบโปรแกรม Triangle

Target Path	Test Case	Input values			No. of GA
		a	b	c	
P1	1	2	2	2	5
P2	2	2	4	5	6
P3	3	4	4	2	7
P4	4	-3	-2	-5	7

ตารางที่ 4 แสดงกรณีทดสอบที่ครอบคลุมเส้นทางทั้งหมดสำหรับการทดสอบโปรแกรม Triangle มีทั้งหมด 4 กรณีทดสอบ จำนวนรอบการดำเนินการขั้นตอนวิธีเชิงพันธุกรรม (No. of GA) เพื่อค้นหากรณีทดสอบที่ครอบคลุมเส้นทางเป้าหมาย P1 ถึง P4 มีจำนวน 5, 6, 7 และ 7 รอบตามลำดับ สำหรับเส้นทางเป้าหมาย P5 ไม่สามารถค้นหากรณีทดสอบที่ครอบคลุมได้ โดยใช้จำนวนรอบในการค้นหา 20 รอบซึ่งเป็นจำนวนรอบสูงสุดที่กำหนดไว้ จากการวิเคราะห์เส้นทาง P5 พบว่าเป็นเส้นทางที่เป็นไปไม่ได้ (infeasible path) ซึ่งเป็นเส้นทางที่ไม่มีกรณีทดสอบใดๆที่สามารถผ่านเส้นทางนี้ได้ ดังนั้นจึงสอดคล้องกับผลที่ได้รับจากขั้นตอนที่ได้นำเสนอ

5. สรุปผลและงานในอนาคต

บทความนี้เสนอการสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมเพื่อคัดเลือกกรณีทดสอบที่เหมาะสมสำหรับการทดสอบซอฟต์แวร์ การดำเนินการเริ่มจากนำรหัสต้นทางของโปรแกรมที่ต้องการทดสอบมาแปลงเป็นกราฟควบคุมการไหล ต่อจากนั้นจะค้นหากรณีทดสอบที่ครอบคลุมทุกเส้นทางการทำงานของโปรแกรมโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม เมื่อนำวิธีการที่ได้นำเสนอไปประยุกต์ใช้กับกรณีศึกษาพบว่า ขั้นตอนวิธีที่นำเสนอสามารถสร้างกรณีทดสอบสำหรับการทดสอบซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

ในอนาคตได้วางแผนที่จะพัฒนาเครื่องมือเพื่อสร้างกรณีทดสอบจากวิธีการที่นำเสนอและนำไปประยุกต์ใช้กับกรณีศึกษาอื่นๆ เพื่อประเมินค่าความถูกต้องของกรณีทดสอบและพัฒนาวิธีการจัดการกับการหากรณีทดสอบสำหรับเส้นทางที่เป็นไปไม่ได้

6. เอกสารอ้างอิง

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli. "Fundamentals of Software Engineering 2nd," *Prentice Hall PTR Upper Saddle River, NJ, USA*, 2002.

[2] G. Winter, J. Periaux, M.Galan and P.Cuesta. "Genetic Algorithms in Engineering and Computer Science," *Wiley Publisher*, 1996.

[3] P. Ammann and J. Offutt. "Introduction to Software Testing," *Cambridge University Press*, 2008.

[4] T. J. McCabe. "A Complexity Measure," *IEEE Transactions on Software Engineering*, pp. 308-320, 1976.

[5] B. Korel. "Automated Test Data Generation," *IEEE Transactions on Software Engineering*, pp.870-879, 1990.

1.3.2 Nuntanee Chuaychoo and Supaporn Kansomkeat. “Path Coverage Test Case Generation Using Genetic Algorithms.” *2nd Advancement on Information Technology International Conference*, 2015.

Jurnal Teknologi

Full Paper

PATH COVERAGE TEST CASE GENERATION USING GENETIC ALGORITHMS

Nuntanee Chuaychoo[1*], Supaporn Kansomkeat[2*]

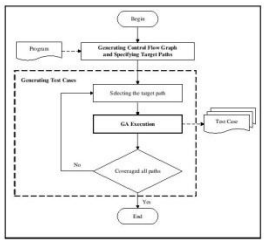
[1] *Department of Management of Information Technology, Faculty of Engineering, Prince of Songkla University, Hat Yai, Songkhla, Thailand*

[2] *Department of Computer Science, Faculty of Science, Prince of Songkla University, Hat Yai, Songkhla, Thailand*

Article history
Received
TBA
Received in revised form
TBA
Accepted
TBA

*Corresponding author
nun-ta-nee@hotmail.com
supaporn.k@psu.ac.th

Graphical abstract



Abstract

Software testing is an important step in the software process. It makes the developed software more accurate and reliable. Generating test cases is a key element in the process of software testing. The quality of testing depends on the test effectiveness. This paper presents a method for generating test cases using genetic algorithms. A test case generation tool is developed with Visual Basic .NET for supporting our method. The proposed method was applied on case studies and mutation testing was used in our experiment for performance evaluation. The results show that the generated test cases are appropriate for software testing.

Keywords: Software Testing; Genetic Algorithms; Test Case Generation

1.0 INTRODUCTION

Software testing is a key step in the software development process. The test is an activity designed to improve software quality. One important step in software testing is the test case selection process. A good test selection contributes to an effective testing. To enable effective testing, the test cases in the set must cover all features that should be tested and the number of test cases should not be too high

Genetic Algorithm (GA) is an optimization algorithm based on evolutionary processes in nature. GA is used to find the right answer with the principle of natural selection to simulate the evolution of life. The GA process finds the answer by exchanging parameters between chromosomes, which will give an improved answer until finally the right answer. There are many methods proposed for generating test cases by applying GA. For example, Jones et al. [1] presented automatic structural software testing using a fitness function of GA based on the Hamming distance. Their structural testing was applied for testing a variety of programs of varying complexity. Pargas et al. [2] presented a technique for automatic test-data generation using GA. They used control dependencies in the program to guide the search for test data to satisfy test requirements. NIE [3] proposed the PSO test case generation algorithm with enhanced exploration ability (EEA-PSO) to restrain the prematurity and reduce the test case generation costs. Mohi-Aldeen et al. [4] used NSA (Negative Selection Algorithm) to automatically generate test cases to satisfy the path coverage of software.

Along with the principles of GA that are consistent with the test cases refinement, the abstract test cases are modified until they meet the targets. In this study, we present a method for generating test cases by using genetic algorithm. In the proposed method, the abstract test cases are generated randomly. Then, they are refined by genetic algorithm for improving the test parameters. In addition, a test case generation tool is developed according to our method for automatically generating test cases. Mutation testing is used to evaluate generated test cases.

2.0 SOFTWARE TESTING

Software testing [5] is an important activity in software quality assurance that shows software reliability. The purpose of software testing is to find errors that may occurred during the software development.

There are two techniques widely used in software testing.

1) Black-Box Testing: This technique is used to test for verify the performance of the duties under the requirements specification, and ignores the internal structure of the program.

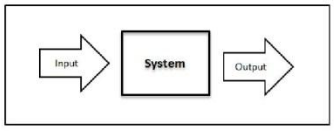


Figure 1 Black-Box Testing

2) White-Box Testing: The testing focuses on the internal program structure. The basic idea of this test is that a crash of the software will be detected. This technique will help to analyze the program to find bugs that may arise from the work of the various commands in the program.

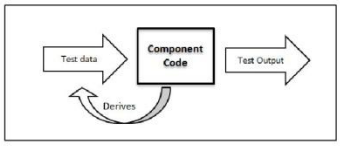


Figure 2 White-Box Testing

3.0 GENETIC ALGORITHM

Genetic algorithm [6] is a process that relies on the theory of evolution of nature offered by John Holland. Genetic algorithm can be used to find better and more affordable solutions through learning by itself.

Figure 3 shows the process of the genetic algorithm [6].

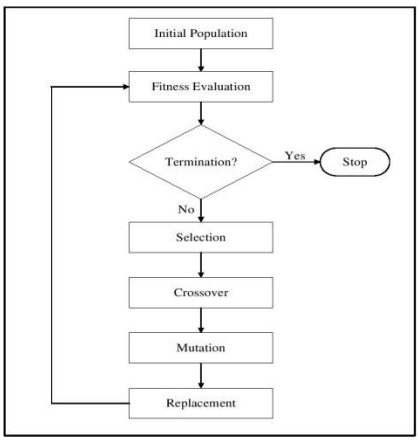


Figure 3 Genetic Algorithm [6]

The main steps of the genetic algorithm can be listed as follows:

1) Initial Population: In this step, initial population of solutions (chromosomes) are randomly produced or manually created from their data type domain.

2) Fitness Evaluation: All chromosomes are evaluated and assigned fitness values. The fitness evaluation is a process used to determine the execute condition in the genetic algorithm process. This process is repeated until a termination condition has been reached.

3) Selection: New population will be selected from the parent population. There are many methods how to select the best chromosome for use in next generation of genetic process such as roulette wheel selection, Boltzman selection, rank selection and some others.

4) Crossover: An importance process of the genetic algorithm is the crossover. The parent population has to switch genes on each chromosome with a switching rate called the probability of crossover. This probability is a real number from 0 to 1 that is randomly generated. In this process, we get a new population that is different from the parent population.

5) Mutation: Mutation is a process of genetic algorithm. Genes in a chromosome will be changed after crossover. This principle helps to create a new answer. The percentage of change is determined by the mutation rate called the probability of mutation.

6) Replacement: The new population after crossover and mutation will be taken to replace the parent population and will become the new parent population.

7) Parameters: The key parameters of genetic algorithm are as following:

a) Population size
b) Probability of crossover
c) Probability of mutation

4.0 CONTROL FLOW GRAPH

Control Flow Graph, CFG [7] is a directed graph used to analyze the function of a computer program. This graph demonstrates the functionality of the software under various conditions. Therefore, the control flow graph analysis can help to test the functionality of the software. A CFG can be built from the program source code. It consists of nodes and edges. A node represents a statement or a basic block of statements. An edge represents the flow of control between nodes. Notations for representing control flow are shown in Figure 4.

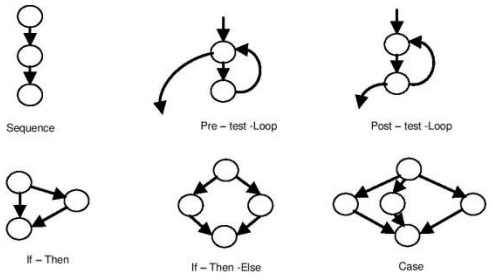


Figure 4 Notions for Representing Control Flow [7]

5.0 TEST CASE GENERATION

In this section, the proposed test cases generating process using genetic algorithm will be described. Our approach consists of two main stages as shown in Figure 5.

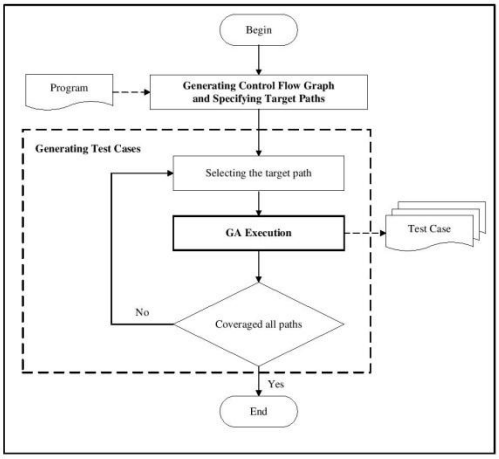


Figure 5 Framework of the proposed approach

5.1 Generating Control Flow Graph and Specifying Target Paths

This procedure consists of two sub-stages:

1) Generating a control flow graph: A control flow graph is generated from a program code. The generated graph consists of nodes connected by edges. Figure 6 shows the control flow graph of Max-Min program that consists of 10 nodes and 12 edges.

```
1   Dim a , b , c , max , min As Double
2   If a > b Then
3           max = a : min = b
4   Else
5           max = b : min = a
6   End If
7   If max < c Then
8           max = c
9   End If
10  If max > c Then
11          min = c
12  End If
```

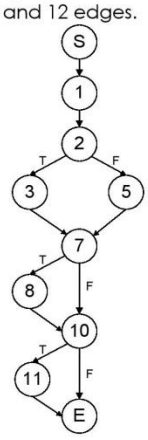


Figure 6 Control Flow Graph of Max-Min Program

2) Specifying Target Paths: The target paths are specified by using the control flow graph generated from the previous step. In this research, the path coverage (PC) is used to specify target paths. The PC requires every path in the tested program has to be executed at least once. Figure 7 shows all target paths of the Max-Min control flow graph in Figure 6.

Path 1	S	1	2T	3	7T	8	10T	11	E
Path 2	S	1	2T	3	7T	8	10F	E	
Path 3	S	1	2T	3	7F	10T	11	E	
Path 4	S	1	2T	3	7F	10F	E		
Path 5	S	1	2F	5	7T	8	10T	11	E
Path 6	S	1	2F	5	7T	8	10F	E	
Path 7	S	1	2F	5	7F	10T	11	E	
Path 8	S	1	2F	5	7F	10F	E		

Figure 7 All Paths of Max-Min CFG

5.2 Generating Test Cases

In this process, test cases are created by using the genetic algorithm. The generation process starts by selecting a target path. This target path is set to be the considering path for finding a test case by using the *Genetic Algorithm Execution* (*GA Execution*). The generation process is repeated until all target paths are considered.

Figure 8 shows the steps of *GA Execution* as following:

Step 1: Initial Test Cases

The input space partitioning approach proposed by Ammann and Offutt [8] is used to generate initial test cases (chromosomes). We consider the data type of each parameter in the program. The number

of partition depends on the data type of the parameter. For example, considering integer or double parameters, the range can be partitioned into three blocks: less than zero, zero, and greater than zero. The initial test cases contain all possible test cases of the considered target path. After all parameters are partitioned into blocks, one block from each parameter is combined to be a test case. We apply the Each Choice (EC) criterion [8] for the combination blocks. The EC requires that each block of parameters must be used in at least one test case.

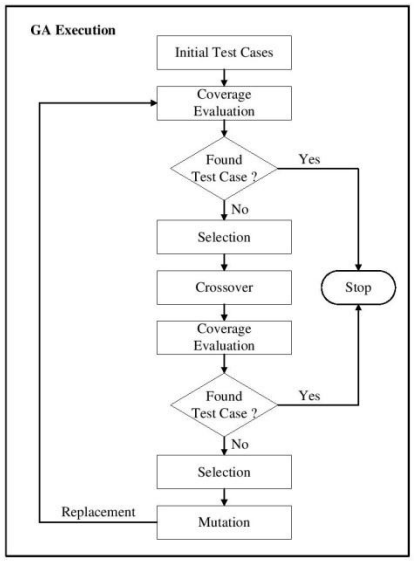


Figure 8 Genetic Algorithm Execution

For example, the Max–Min program has three input parameters (a, b, and c) that are double data types. Each parameter can be partitioned into three blocks ([a<0.0, a=0.0, a>0.0], [b<0.0, b=0.0, b>0.0], and [c<0.0, c=0.0, c>0.0]), so we can combine them into three sets: {a < 0.0 , b < 0.0 , c < 0.0}, {a = 0.0 , b = 0.0 , c = 0.0}, and {a > 0.0 , b > 0.0 , c > 0.0}. A set represents a test case. Then, the test data for each partition of each set are generated randomly. Table I shows the initial test data for the Max–Min program.

Table I Initial Test Data for Max-Min Program

Set	a	B	c
1	-2.2	-1.4	-4.5
2	0.0	0.0	0.0
3	3.7	2.4	5.3

Step 2: Coverage Evaluation

In this research, we use the branch distance calculation proposed by Korel [9] for fitness evaluation. Branch distance is calculated based on the type of branch predicate. Branch predicate and branch distance of the Korel's distance function are shown in Table II. Table III shows all branch predicates of the Max-Min Program.

Table II Korel's Distance Function [9]

Branch Predicate	Branch Distance
a < b	a - b
a <= b	a - b
a > b	b - a
a >= b	b - a
a == b	abs(a-b)
a <> b	abs(a-b)
a && b	a + b
a | | b	min(a , b)

Table III Decision for Max-Min Program

Branch Predicate	Branch Distance
a > b	b - a
max < c	max - c
max > c	c - max

Korel's distance function is used to compute the distance between the target path and the considering path. In this study, the selected target path and an execution paths that traversed by test data of a test case are compared; if they are different then the all branch distances of this execution path are calculated to be the fitness value of the test case. In case they are same then the GA execution process for the target path is finished and the test case is set for the selected target path.

For example, the selected target path is the first path in Figure 7. We found that every execution path of test data sets in Table I is different; therefore all branch distances of every execution paths are calculated as shown in Table IV.

Table IV Branch Distance for Test Sets

Set	A	b	c	Branch Distances
1	-2.2	-1.4	-4.5	0.8
2	0.0	0.0	0.0	0.0
3	3.7	2.4	5.3	-1.6

Step 3: Selection

The selection step is done to select test cases for the evolution process, crossover, and mutation. The sets of test are sorted by descending order of branch distance values. The data sets with small distance values will be selected for the evolution process. For example, the test data sets in Table IV are sorted to be the new sequence sets 3, 2, and 1.

Step 4: Crossover

After sorting in the selection process, the first two data sets are chosen for crossover operation. In this research, we use single-point crossover. Parameters of the chosen two data sets (parent) are exchanged at a random position to produce two new test data sets (child). Figure 9 shows an example for a single point crossover at the first position.

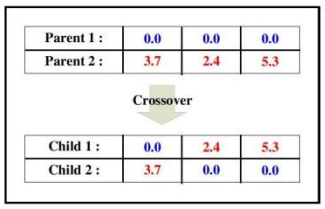


Figure 9 Single Point Crossover

Step 5: Mutation

After sorting in the selection process, mutation is applied to the first data set using single point mutation. In our method, the position for mutation is selected randomly. The value of this position is replaced by another value that is generated randomly based on the range of its data type. Figure 10 shows an example for the single point mutation at first position that changes the value from -2.2 to 4.

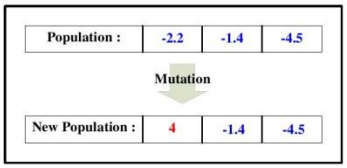


Figure 10 Single Point Mutation

To support our idea, we implemented the application that provides GUI for generating test data. The application is developed by Visual Basic .NET works on Windows base. User can set parameters of the genetic algorithm to find the best solution in generating test sets. The user interface is shown in Figure 11.

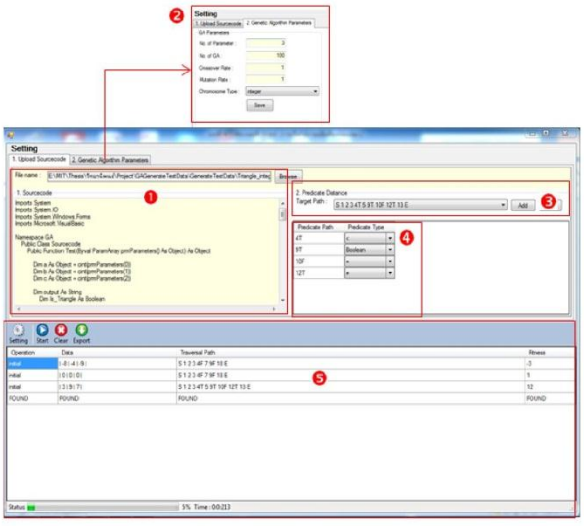


Figure 11 Test Cases Generation GUI

The application contains five parts. Part ❶ is used to upload a program file written in Visual Basic .NET. Part ❷ is used to set parameters for genetic algorithm which are the numbers of the parameters, population size, probability of crossover, probability of mutation and chromosome type. Part ❸ is used to define target paths. Part ❹ is used to specify branch predicates. After execution, the results are shows in part ❺.

6.0 CASE STUDY

In this section, three small programs are used to evaluate our approach. The case study programs are as follows:

1) Calculate Maximum–Minimum

This program receives three numbers that are double types and finds the maximum and minimum numbers. There are three branch predicates in the source code.

2) Triangle Problem

This program receives three sides of triangle that are integer numbers and classifies type of triangle (Not a Triangle, Equilateral, Scalene, and Isosceles). There are four branch predicates in this source code.

3) Calculate Middle Data

This program gets three characters and identifies the character that has the middle value. For example the characters are 'a', 'c' and 'b' then the result is character 'b'. There are three branch predicates in this source code.

Test cases of these three programs are generated by our tool. As stated before, there are three key parameters in the genetic algorithm: population size, probability of crossover, and probability of mutation are set to be 100, 1, and 1, respectively. Probabilities of crossover and mutation are set to be 1 for the execution crossover and mutation operations in every generation which increase the probability to find the appropriate test cases. The results are shown in Table V.

Table V Test Cases for Case Studies Programs

Program	Total Branch Predicates'	Total Target Paths	Found Test cases	Not found Test cases
Calculate Maximum – Minimum	3	8	6	2
Triangle Problem	4	8	4	4
Find Middle Character	3	6	6	0

As shown in Table V, some target paths of the Calculate Maximum-Minimum and Triangle Problem programs cannot generate test cases. Because these target paths are infeasible paths that cannot be traversed by any test cases. Possible causes of infeasible paths are source codes that contain

logically inconsistent predicates or some data that produce an infinity loop in program, and others.

Mutation testing [10] is used to evaluate the effectiveness of the test cases created by our method. Mutation testing starts by making small syntactic changes into the original program. The mutated programs are called *mutants*. Mutants are obtained by applying mutation operators. For example, an arithmetic operator may be changed from addition to subtraction. Mutation analysis is based on the number of killed mutants. The behavior of the original program is different from the mutant called killed mutant. There are cases where it is not possible to find a test case that could kill this mutant called equivalent mutant. The ratio of killed mutants to the total mutants minus the number of equivalent mutants is called *mutation score*.

The test cases evaluation process is carried out as follows:

1) Test cases of case study programs are generated by using our algorithm.
2) Mutants for each case study are created by applying the mutation operators designed by Offutt et al. [11]. As shown in Table VI, there are 21 mutants for calculating the maximum and minimum program, 38 mutants for the triangle problem program, and 21 mutants for the calculation of the middle data program.
3) Test cases are executed with the original program and the mutant programs to calculate mutation scores. If a mutation score is one or near to one, it indicates the effectiveness of the test cases. The evaluation results are shown in Table VI.

Table VI Test Cases Evaluation for Case Studies Programs

Program	Total Test cases	Total Mutants	Killed mutants	Mutation Score
Calculate Maximum – Minimum	6	21	19	0.9
Triangle Problem	4	38	25	0.86
Find Middle Character	6	21	21	1

7.0 CONCLUSIONS

This paper presents a method for generating test cases using genetic algorithm. First, a flow graph is created from the program source code and the paths of the graph are specified by following path coverage criteria. Then, the genetic algorithm is used for generating test cases. Finally, we use mutation testing to evaluate the effectiveness of the test cases. Experimental data show that tests generated by our method have a strong ability to detect faults. In addition, this research has also developed a test case generation tool based on the proposed method. In the future, we intend to improve the tool to automate the whole test process and find solutions for infeasible paths. Moreover, more complex programs are needed for evaluating our method and we need more studies on the optimization algorithms for test case generation, such as hybrid GA and PSO [12,13], which have a higher efficiency than GA algorithms.

Acknowledgement

The authors would like to thank for the support given to this research by the Faculty of Engineering and Faculty of Science, Prince of Songkla University, Thailand.

References

[1] B.F Jones, H.-H Sthamer and D.E. Eyres. 1996. Automatic structural testing using genetic algorithms. *Software Engineering Journal*: 299-306.
[2] R. P. Pargas, M. J. Harrold and R. R. Peck. 1999. Test-Data Generation Using Genetic Algorithms. *Journal of Software Testing, Verification and Reliability I*: 1-19.
[3] P. NIE. 2012. A PSO Test Case Generation Algorithm with Enhances Exploration Ability. *Journal of Computational Information System*: 5785-5793.
[4] S. M. Mohi-Aldeen, R. Mohamad and S. Deris. 2014. Automatic Test Case Generation for Structural Testing Using Negative Selection Algorithm. *International Conference of Recent Trends in Information and Communication Technologies*: 270-280.
[5] C. Ghezzi, M. Jazayeri, and D. Mandrioli. 2002. Fundamentals of Software Engineering 2nd. USA: Prentice Hall PTR Upper Saddle River.
[6] G. Winter, J. Periaux, M.Galan and P.Cuesta. 1996. Genetic Algorithms in Engineering and Computer Science. USA: Wiley Publisher.
[7] P. C. Jorgensen. 2007. Software Testing A Craftsman's Approach 3rd. USA: AuerbachPiblictions.
[8] P. Ammann and J. Offutt. 2008. Introduction to Software Testing. UK: Cambridge University Press.
[9] B. Korel. 1996. Automated test data generation for programs with procedures. *Proceedings of International Symposium on Software Testing and Analysis*: 209-215.
[10] L. Guti´errez-Madro˜nal, J. Jos´e Dom´ınguez-Jim´enez and I. Medina-Bulo. 2014. Mutation Testing: Guideline and Mutation Operator Classification. *The Ninth International Multi-Conference on Computing in the Global Information Technology*: 171-179.
[11] A.J. Offutt, A.Lee, G. Rothermel, R. Untch, and C. Zapf. 1996. An Experimental Determination of Sufficient Mutant Operators. *ACM Transaction on Software Engineering Methodology*. 99-118.
[12] A. Singh, N. Garg and T. Saini. 2014. A hybrid Approach of Genetic Algorithm and Particle Swarm Technique to Software Test Case Generation. *International Journal of Innovations in Engineering and Technology*: 208-214.
[13] S. Singla, D. Kumar, H M Rai and P. Singla. 2011. A hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts. *International Journal of Advanced Science and Technology*: 15-25.

ประวัติผู้เขียน

ชื่อ - สกุล นางสาวนันทนี ช่วยชู

รหัสประจำตัวนักศึกษา 5510121039

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิศวกรรมศาสตรบัณฑิต (วิศวกรรมคอมพิวเตอร์) เกียรตินิยมอันดับสอง	มหาวิทยาลัยวลัยลักษณ์	2553

ประสบการณ์ทำงาน หน่วยวิจัยนวัตกรรมด้านสารสนเทศ มหาวิทยาลัยวลัยลักษณ์
ตำแหน่ง โปรแกรมเมอร์

การตีพิมพ์เผยแพร่ผลงาน

นันทนี ช่วยชู และ สุภาภรณ์ กานต์สมเกียรติ. “การสร้างกรณีทดสอบโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม.” การประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 10 (NCCIT 2014), 2557.

Nuntanee Chuaychoo and Supaporn Kansomkeat. “Path Coverage Test Case Generation Using Genetic Algorithms.” *2nd Advancement on Information Technology International Conference*, 2015.