การปรับปรุงอัลกอริทึมการสร้างกรณีทดสอบแบบเอ็นเวย์

นายกิติภูมิ ชัยสุวรรณ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2549
ISBN 974-14-2503-1

IMPROVEMENT OF ALGORITHM FOR N-WAY TEST CASE GENERATION

Mr. Kitipoom Chaisuwan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering
Department of Computer Engineering
Chulalongkorn University
Academic Year 2006

ISBN 974-14-2503-1

หัวข้อวิทยานิพนธ์	การปรับปรุงอัลกอริทึมการสร้างกรณีทดสอบแบบเอ็นเวย์
โดย	นายกิติภูมิ ชัยสุวรรณ
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.อรรถสิทธิ์ สุรฤกษ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.อรรถสิทธิ์ สุรฤกษ์)

..... กรรมการ
(อาจารย์ นครทิพย์ พร้อมพูล)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. อานนท์ รุ่งสว่าง)

กิติภูมิ ชัยสุวรรณ : การปรับปรุงอัลกอริทึมการสร้างกรณีทดสอบแบบเอ็นเวย์ (IMPROVEMENT OF ALGORITHM FOR N-WAY TEST CASE GENERATION) อาจารย์ที่ปรึกษา : อ. ดร.อรรถสิทธิ์ สุรฤกษ์, 67 หน้า
ISBN 974-14-2503-1

การทดสอบแบบแพร์ไวส์เป็นการทดสอบซอฟต์แวร์ที่ใช้ทดสอบทุกคู่ของค่าที่เป็นไปได้ของแต่ละพารามิเตอร์ ในขณะที่การทดสอบแบบเอ็นเวย์นั้น จะทำการสร้างกรณีทดสอบที่ใช้ทดสอบเฉพาะกรณีที่จำเป็นเท่านั้น ในงานวิจัยนี้ ผู้วิจัยนำเสนอการปรับปรุงอัลกอริทึมไอพีโอ เพื่อให้สามารถนำมาใช้ทดสอบซอฟต์แวร์แบบเอ็นเวย์ได้ ประเด็นที่จะทำการปรับปรุงอัลกอริทึม มุ่งเน้นไปที่จำนวนกรณีทดสอบที่ทำการสร้างออกมา โดยการทดสอบในงานวิจัยนี้ จะเปรียบเทียบจำนวนกรณีทดสอบกรณีที่พารามิเตอร์ของระบบมีความสัมพันธ์เป็นแบบแพร์ไวส์ ระหว่างจำนวนกรณีทดสอบที่สร้างขึ้นจากอัลกอริทึมที่นำเสนอ กับจำนวนกรณีทดสอบที่สร้างขึ้นจากวิธีการวิเคราะห์ค่าขอบเขต วิธีการทดสอบสภาพทนทาน วิธีการทดสอบกรณีเลวร้ายที่สุด และอัลกอริทึมไอพีโอ ส่วนในกรณีที่พารามิเตอร์ของระบบมีความสัมพันธ์กันแบบเอ็นเวย์จะเปรียบเทียบจำนวนกรณีทดสอบที่สร้างขึ้นจากอัลกอริทึมไอพีโอที่นำเสนอ กับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี ซึ่งผลการทดสอบ โดยเฉลี่ยแล้วจำนวนกรณีทดสอบที่สร้างขึ้นจากอัลกอริทึมที่นำเสนอมีจำนวนน้อยกว่าหรือเท่ากับจำนวนกรณีทดสอบที่สร้างจากวิธีเออีทีจีด้วยความเชื่อมั่น 99 เปอร์เซ็นต์

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต กิติภูมิ ชัยสุวรรณ
สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่ออาจารย์ที่ปรึกษา
ปีการศึกษา 2549

4770222221 : MAJOR SOFTWARE ENGINEERING

KEY WORD : SOFTWARE TESTING / PAIR-WISE TESTING / N-WAY TESTING / COMBINATION / IPO ALGORITHM / AETG

KITIPOOM CHAISUWAN : IMPROVEMENT OF ALGORITHM FOR N-WAY TEST CASE GENERATION

THESIS ADVISOR : ATHASIT SURARERKS, Ph.D., 67 pp.

ISBN 974-14-2503-1

Pair-wise testing is a kind of software testing that combines all pairs of input values of each parameter while n-way testing generates only the necessary test cases. The thesis proposes an improvement version of In-Parameter-Order algorithm (IPO) for n-way test case generation. The concept of improvement focuses on the number of generating test cases. For pair-wise testing, the proposed algorithm is compared with Boundary Value Analysis, Robustness Testing, Worst-case Testing and IPO. For n-way testing, the proposed algorithm is compared with AETG (Automatic Efficient Test Generator). The result of statistical test at 99 percent confidence interval shows that number of test cases generated by the proposed algorithm is less than or equal to the number of test cases obtained from AETG.

Department Computer Engineering Student's signature .....

Field of study Software Engineering Advisor's signature .....

Academic year 2006

กิตติกรรมประกาศ

ข้าพเจ้าใคร่ขอกราบขอบพระคุณอาจารย์ ดร.อรรถสิทธิ์ สุรฤกษ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ที่กรุณาแนะนำให้แนวคิด ความรู้ คำปรึกษา ความช่วยเหลือต่างๆ ตลอดจนดูแลการทำวิทยานิพนธ์ของข้าพเจ้าจนสำเร็จลุล่วงลงได้ด้วยดี

ขอกราบขอบพระคุณ รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์ อาจารย์ นครทิพย์ พร้อมพูล และผู้ช่วยศาตราจารย์ ดร. อานนท์ รุ่งสว่าง ซึ่งเป็นคณะกรรมการสอบวิทยานิพนธ์ที่ได้สละเวลาและให้คำแนะนำต่างๆ ที่เป็นประโยชน์อย่างยิ่งต่อการจัดทำวิทยานิพนธ์ฉบับนี้

ขอขอบคุณ อาจารย์ทุกท่าน ที่ได้ประสิทธิ์ประสาทวิชาให้ข้าพเจ้า รวมถึงชี้แนะสิ่งดีๆ ตลอดเวลาที่ข้าพเจ้าได้ศึกษาเล่าเรียนระดับมหาบัณฑิต

สุดท้ายนี้ ข้าพเจ้าใคร่ขอกราบขอบพระคุณบิดา มารดาของข้าพเจ้า ที่คอยให้กำลังใจ และสนับสนุนด้านการเงินแก่ข้าพเจ้า และขอขอบคุณเพื่อนๆ ทุกคน ที่ให้ความช่วยเหลือในทุกๆ ด้านจนสามารถทำวิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วง

สารบัญ

หน้า

สารบัญตาราง

สารบัญภาพ

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การทดสอบซอฟต์แวร์ (software testing) เป็นส่วนมีความสำคัญมากในวัฏจักรการพัฒนาซอฟต์แวร์ (Software Develop Life Cycle: SDLC) แต่การทดสอบซอฟต์แวร์ต้องเสียค่าใช้จ่ายสูงมาก ซึ่งในการพัฒนาซอฟต์แวร์แต่ละครั้ง ต้องเสียค่าใช้จ่ายเพื่อทดสอบซอฟต์แวร์สูงประมาณ 30% ถึง 50% ของค่าใช้จ่ายในการพัฒนาซอฟต์แวร์ทั้งหมด [1] และต้องใช้เวลาในการทดสอบซอฟต์แวร์สูงประมาณ 40% ของวัฏจักรการพัฒนาซอฟต์แวร์ [2] โดยจะทำการทดสอบที่ขั้นตอนใดนั้นขึ้นอยู่กับรูปแบบของขั้นตอนการพัฒนาซอฟต์แวร์ ที่ผู้พัฒนาซอฟต์แวร์เลือกมาใช้ในการพัฒนาซอฟต์แวร์ จากงานวิจัยของดาวิด เอ็ม โคเฮน (David M. Cohen) และคณะ [3] ได้กล่าวว่าการสร้างกรณีทดสอบเป็นสิ่งที่ยากและเสียค่าใช้จ่ายสูง ต้องใช้เวลาหลายเดือนในการสร้างกรณีทดสอบ ซึ่งการทดสอบซอฟต์แวร์เป็นการค้นหาข้อผิดพลาด (error) ที่มีอยู่ในซอฟต์แวร์ออกมาให้มากที่สุด [4] เพื่อนำข้อผิดพลาดที่เกิดขึ้นนั้นมาให้ผู้ที่ทำการพัฒนาซอฟต์แวร์ ได้ทำการแก้ไข และปรับปรุงซอฟต์แวร์ โดยที่การทดสอบซอฟต์แวร์มีความสำคัญในหลายๆ ด้านของการพัฒนาซอฟต์แวร์ เช่น ช่วยลดข้อผิดพลาด ที่เกิดขึ้นในแต่ละขั้นตอนการพัฒนาซอฟต์แวร์ ทำให้สามารถพัฒนาซอฟต์แวร์ได้ครบถ้วน และถูกต้องตรงตามความต้องการของผู้ใช้งาน ยิ่งไปกว่านั้นการทดสอบซอฟต์แวร์ในแต่ละขั้นตอนของการพัฒนาซอฟต์แวร์ จะช่วยลดค่าใช้จ่ายในการบำรุงรักษาซอฟต์แวร์ (software maintenance)

นักวิจัยจำนวนมาก ได้ให้ความสนใจขั้นตอนในการสร้างกรณีทดสอบ (test case generation) ให้มีจำนวนกรณีทดสอบน้อยที่สุด และสามารถทำการทดสอบได้ครอบคลุมทุกกรณีที่สามารถเกิดขึ้นในการใช้ซอฟต์แวร์นั้นๆ เพื่อลดค่าใช้จ่ายในการทำการทดสอบซอฟต์แวร์ เนื่องจากการมีกรณีทดสอบมากจะทำให้ค่าใช้จ่ายที่ใช้ในการทดสอบเพิ่มมากขึ้นตามไปด้วย และต้องใช้เวลาในการทดสอบมากขึ้น เช่น งานวิจัยของคัว ชุง ไถ (Kuo-Chung Tai) และหยู เล่ย (Yu Lei) [11] ได้นำเสนออัลกอริทึมที่ใช้ในการสร้างกรณีทดสอบที่ครอบคลุมเฉพาะพารามิเตอร์ที่มีความสัมพันธ์แบบแพร์ไวส์ (pair-wise) ซึ่งจำนวนกรณีทดสอบน้อยกว่าการใช้วิธีเออีทีจี (Automatic Efficient Test Generator: AETG) [5] โดยที่ วิธีเออีทีจี คือ วิธีที่ใช้ในการสร้างกรณีทดสอบสำหรับทดสอบโปรแกรมประยุกต์มากมาย เช่น ระบบของเบลคอร์ (Bellcore System) เป็นต้น และงานวิจัยของโซลเหมิน ไมตี (Soumen Maity) และคณะ [6] ได้ทำการปรับปรุงอัลกอริทึมที่ใช้ในการสร้างกรณีทดสอบของคัว ชุง ไถ และหยู เล่ย ให้มีกรณีทดสอบน้อยลงกว่าเดิม และการสร้างกรณีทดสอบแบบอื่นๆ เช่น ซูซาน กอร์ (Susan Khor) และปิเตอร์ โกรโกโน (Peter Grogono) [7] ได้ใช้ขั้นตอนวิธีเชิงพันธุกรรม (genetic algorithm)

และการวิเคราะห์แนวคิดเชิงรูปนัย (formal concept analysis) ออกแบบกิ่งของกรณีทดสอบที่ครอบคลุมการทดสอบนั้นๆ อลัน ดับเบิลยู วิลเลียม (Alan W. Williams) และโรเบิร์ต แอล โปรเบิร์ต (Robert L. Probert) [8] ทำการหากรณีทดสอบสำหรับทดสอบการประกอบกันของส่วนประกอบของระบบเครือข่าย (network components) โดยมองแต่ละส่วนประกอบของระบบเครือข่ายนั้น มีความสัมพันธ์กันแบบแพร์ไวส์ เป็นต้น

จากงานวิจัยของดาวิด เอ็ม โคเฮน และคณะ พบว่าความสัมพันธ์ของพารามิเตอร์ที่นำมาใช้ในซอฟต์แวร์ไม่จำเป็นต้องมีความสัมพันธ์แบบแพร์ไวส์เท่านั้น ซึ่งในงานวิจัยนี้จะทำการปรับปรุงอัลกอริทึมของคัว ชุง ไถ และหยู เล่ย ให้มีความเหมาะที่สุด (optimization) และครอบคลุมกรณีที่พารามิเตอร์แต่ละตัวมีความสัมพันธ์กันนอกเหนือจากแบบแพร์ไวส์

1.2 วัตถุประสงค์ของการวิจัย

งานวิจัยนี้ มีวัตถุประสงค์เพื่อปรับปรุงวิธีในการสร้างกรณีทดสอบ ซึ่งสามารถสร้างกรณีทดสอบที่ครอบคลุมระบบที่พารามิเตอร์มีความสัมพันธ์กันทั้งแบบแพร์ไวส์ และเอ็นเวย์ (n-way) ซึ่งสามารถกำหนดความสัมพันธ์ของแต่ละพารามิเตอร์ได้ โดยความสัมพันธ์ของแต่ละพารามิเตอร์นั้นแบ่งออกเป็นกลุ่ม ซึ่งพารามิเตอร์ในแต่ละกลุ่มนั้นอาจจะมีความสัมพันธ์ หรือไม่มีความสัมพันธ์กันเลยก็ได้

1.3 ขอบเขตของการวิจัย

1) อัลกอริทึมใหม่ที่ออกแบบ สามารถกำหนดความสัมพันธ์ของพารามิเตอร์ให้มีความสัมพันธ์กันแบบเอ็นเวย์ได้

2) พารามิเตอร์แต่ละพารามิเตอร์สามารถกำหนดอยู่ในกลุ่มความสัมพันธ์ได้มากกว่าหนึ่งกลุ่มความสัมพันธ์

3) ปรับปรุงอัลกอริทึมของคัว ชุง ไถ และหยู เล่ย [9, 10] ที่สามารถสร้างกรณีทดสอบเฉพาะพารามิเตอร์ที่จะสร้างกรณีทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์ ให้สามารถสร้างกรณีทดสอบให้กับพารามิเตอร์ที่มีความสัมพันธ์กันแบบเอ็นเวย์ได้ด้วย

4) ทำการเปรียบเทียบจำนวนกรณีทดสอบระหว่างจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมใหม่ วีธีวิเคราะห์ค่าขอบเขต วิธีการทดสอบสภาพทนทาน และวิธีการทดสอบกรณีเลวร้ายที่สุด เมื่อพารามิเตอร์ที่นำมาสร้างกรณีทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์

5) ทำการเปรียบเทียบขั้นตอนที่ใช้ในการสร้างกรณีทดสอบระหว่างอัลกอริทึมใหม่ กับอัลกอริทึมไอพีโอของคัว ชุง ไถ และหยู เล่ย [9, 10]

6) ทำการเปรียบเทียบจำนวนกรณีทดสอบที่สร้างขึ้นระหว่างจำนวนกรณีทดสอบที่สร้างด้วยอัลกอริทึมใหม่ อัลกอริทึมไอพีโอขอลคัว ชุง ไถ และหยู เล่ย [9, 10] และอัลกอริทึม

ของโซลเหมิน ไมตี้ [6] เมื่อพารามิเตอร์ที่นำมาสร้างกรณีทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์

7) ทำการเปรียบเทียบจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมใหม่กับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมของระบบเออีทีจี [5] เมื่อพารามิเตอร์ที่นำมาสร้างกรณีทดสอบมีความสัมพันธ์กันแบบเอ็นเวย์

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1) ศึกษาทฤษฎีพื้นฐานของการสร้างกรณีทดสอบ และวิธีการทดสอบซอฟต์แวร์แบบต่างๆ โดยเฉพาะการทดสอบซอฟต์แวร์ในระบบที่ใช้พารามิเตอร์ต่างๆกำหนดการทำงาน
2) ศึกษาวิธีการสร้างกรณีทดสอบ โดยศึกษาจากงานวิจัยที่มีอยู่เดิม
3) ปรับปรุงอัลกอริทึมจากงานวิจัยเดิม
4) ออกแบบขั้นตอนวิธีการสร้างกรณีทดสอบที่สามารถกำหนดความสัมพันธ์ของแต่ละพารามิเตอร์ได้
5) ทดสอบวิธีการที่นำเสนอ
6) เปรียบเทียบผลที่ได้จากการทดลอง
7) วิเคราะห์ผลการทดลอง
8) สรุปผลและเรียบเรียงวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับจากการวิจัย

สามารถนำวิธีการสร้างกรณีทดสอบนี้ ไปใช้ในการสร้างกรณีทดสอบที่ครอบคลุมทุกกรณีที่อาจเกิดขึ้นจากการใช้ซอฟต์แวร์ โดยซอฟต์แวร์ที่พัฒนาขึ้นนั้นต้องทำการทดสอบด้วยการใส่ค่าที่เป็นไปได้ของแต่ละพารามิเตอร์เข้าไป และพารามิเตอร์แต่ละพารามิเตอร์ที่นำไปใช้ในการทดสอบ สามารถมีความสัมพันธ์ที่แตกต่างกันได้ทั้งแบบแพร์ไวส์ และแบบเอ็นเวย์

1.6 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง "An improvement of n-way test case generation algorithm" โดย กิติภูมิ ชัยสุวรรณ และอรรถสิทธิ์ สุรฤกษ์ ในงานประชุมทางวิชาการ 10th National Computer Science and Engineering Conference (NCSEC2006) ณ มหาวิทยาลัยขอนแก่น ระหว่างวันที่ 25-27 ตุลาคม 2549

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

การทดสอบซอฟต์แวร์นั้น เป็นกระบวนการที่ใช้ทดสอบการทำงานของซอฟต์แวร์ [4] ว่ามีข้อผิดพลาดที่ใด โดยการทดสอบซอฟต์แวร์ประกอบด้วย

1) การทดสอบฟังก์ชัน (functional testing)

จะทำการทดสอบค่าขอบเขต (boundary value testing) ทดสอบชั้นการสมมูล (equivalence class testing) ทดสอบตารางตัดสินใจ (decision table-based testing) และทดสอบฟังก์ชันย้อนหลัง (retrospective on functional testing) เป็นต้น

2) การทดสอบโครงสร้าง (structure testing)

จะทำการทดสอบวิถี (path testing) ทดสอบกระแสข้อมูล (data flow testing) และทดสอบโครงสร้างย้อนหลัง (retrospective on structural testing) เป็นต้น

3) การทดสอบการรวมระบบ (integration and system testing)

จะทำการทดสอบลำดับขั้นของการทดสอบ (levels of testing) ทดสอบการรวมระบบ (integration testing) ทดสอบระบบ (system testing) และทดสอบการโต้ตอบ (interaction testing)

4) การทดสอบเชิงอ็อบเจกต์ (object-oriented testing)

จะทำการทดสอบประเด็นทั่วไปในการทดสอบเชิงอ็อบเจกต์ (issues in object-oriented testing) ทดสอบคลาส (class testing) ทดสอบการรวมเชิงอ็อบเจกต์ (object-oriented integration testing) ทดสอบส่วนต่อประสานกราฟิกกับผู้ใช้ (GUI testing) และทดสอบระบบเชิงอ็อบเจกต์ (object-oriented system testing) เป็นต้น

2.1.1 การทดสอบซอฟต์แวร์ (software testing)

การทดสอบซอฟต์แวร์ [4, 11] เป็นกระบวนการประเมินซอฟต์แวร์เพื่อตรวจสอบว่าซอฟต์แวร์ทำงานเป็นไปตามความต้องการของซอฟต์แวร์หรือไม่

การทดสอบซอฟต์แวร์แบ่งระดับของการทดสอบออกเป็น 3 ระดับ คือ การทดสอบระดับหน่วย (unit testing) การทดสอบแบบบูรณาการ (integration testing) และการทดสอบระบบ (system testing) การทดสอบแต่ละระดับมีรายละเอียด ดังนี้

2.1.1.1 การทดสอบระดับหน่วย

การทดสอบระดับหน่วย เป็นการทดสอบความสามารถของโมดูล ซึ่งเป็นหน่วยที่เล็กที่สุดของการออกแบบซอฟต์แวร์ การทดสอบระดับหน่วย จะทำหลังจากพัฒนาโมดูลนั้นเสร็จ

2.1.1.2 การทดสอบแบบบูรณาการ

การทดสอบแบบบูรณาการ เป็นการทดสอบการทำงานร่วมกันของโมดูลที่ผ่านการทดสอบระดับหน่วยมาแล้ว การทดสอบแบบบูรณาการมีหลายวิธี เช่น การรวมจากบนลงล่าง (top-down approach) การรวมจากล่างขึ้นบน (bottom-up approach) การรวมแบบแซนด์วิช (sandwich approach) และการรวมแบบบิ๊กแบง (big-bang approach) เป็นต้น

2.1.1.3 การทดสอบระบบ

การทดสอบระบบ เป็นการทดสอบว่าซอฟต์แวร์สามารถทำงานได้ตามความต้องการของผู้ใช้งาน และข้อกำหนดของซอฟต์แวร์หรือไม่ การทดสอบระบบประกอบด้วย 5 การทดสอบ ดังนี้

1) การทดสอบฟังก์ชัน (functional testing) เป็นการทดสอบที่ทดสอบการทำงานของซอฟต์แวร์ให้เป็นไปตามความต้องการของผู้ใช้งานด้วยเทคนิคแบบแบล็กบ็อกซ์ (black box) ซึ่งการทดสอบแบบแบล็กบ็อกซ์ไม่สนใจกระบวนการทำงานภายในของซอฟต์แวร์

2) การทดสอบความกดดัน (stress testing) เป็นการทดสอบซอฟต์แวร์ภายใต้สภาวะการใช้งานอย่างหนัก เพื่อตรวจหาความสามารถสูงสุดของซอฟต์แวร์ภายใต้ทรัพยากรที่กำหนด

3) การทดสอบประสิทธิภาพ (performance testing) เป็นการทดสอบประสิทธิภาพซอฟต์แวร์ตามที่ระบุในข้อกำหนดของซอฟต์แวร์ การทดสอบประสิทธิภาพตรวจสอบจากเวลาตอบสนองการทำงานของซอฟต์แวร์ การทดสอบประสิทธิภาพจะทำงานที่สภาวะการใช้งานปกติ

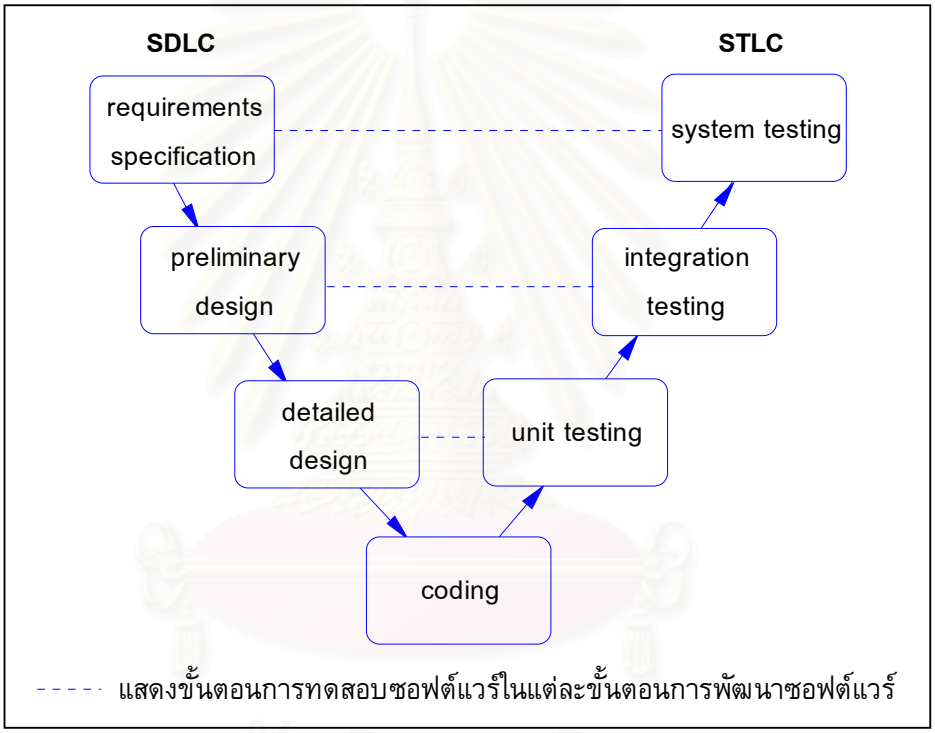
4) การทดสอบความปลอดภัย (security testing) เป็นการทดสอบระบบรักษาความปลอดภัยของซอฟต์แวร์จากผู้ไม่มีสิทธิใช้งานซอฟต์แวร์

5) การทดสอบพื้นหลัง (background testing) เป็นการทดสอบซอฟต์แวร์ด้านความสามารถการรองรับรายการทำงานหลากหลายรายการในเวลาเดียวกัน

2.1.2 ระดับของการทดสอบ (level of testing)

ระดับของการทดสอบซอฟต์แวร์นั้น ส่วนมากจะใช้แบบจำลองวี (v-model) [2] ซึ่งเป็นแบบจำลองที่ใช้กันอย่างแพร่หลายในหลายโครงการ (projects) ซึ่งในทุกๆ ขั้นตอน

ของวัฏจักรการทดสอบซอฟต์แวร์ (Software Testing Life Cycle: STLC) ในแบบจำลองนั้นมีการตอบกลับจากบางกิจกรรมในวัฏจักรการพัฒนาซอฟต์แวร์ โดยแบบจำลองวีนั้นขยายออกมาจากแบบจำลองวอเตอร์ฟอล (waterfall model) [12, 1] ซึ่งในแต่ละขั้นตอน (phases) ของวัฏจักรการพัฒนาซอฟต์แวร์ ด้วยแบบจำลองวี หรือการทดสอบแบบจำลองพื้นฐาน (model based testing) [13] ซึ่งการทดสอบซอฟต์แวร์ด้วยแบบจำลองนี้ สามารถทำขั้นตอนการพัฒนา และการทดสอบไปพร้อมๆกันได้ โดยสามารถใช้ทดสอบหน่วยเล็กๆ (small unit) ส่วนประกอบของส่วนเล็กๆ (collections of units) หรือฟังก์ชันทั่วไปของระบบ (entire system) ดังนั้น แบบจำลองนี้สามารถนำมาใช้ช่วยในการทดสอบการทำงานของแต่ละขั้นตอนได้ โดยไม่ต้องรอให้ถึงขั้นตอนการพัฒนาโปรแกรม (coding) ดังรูปที่ 2.1



รูปที่ 2.1 แบบจำลองวี

จากความสัมพันธ์ของลำดับขั้นของการทดสอบระหว่างการทดสอบฟังก์ชัน และการทดสอบโครงสร้าง จะเห็นว่าการทดสอบโครงสร้าง เหมาะสมที่จะทดสอบในระดับหน่วย (unit level) ขณะที่การทดสอบฟังก์ชันเหมาะสมที่จะทดสอบในขั้นตอนระบบ (system level)

2.1.3 การทดสอบค่าขอบเขต (boundary value testing)

เป็นวิธีการในการหากรณีทดสอบ โดยการดูจากขอบเขตของแต่ละค่าในแต่ละพารามิเตอร์ ที่จะนำมาใช้ในซอฟต์แวร์ทั้งค่าที่เป็นไปได้ และค่าที่เป็นไปไม่ได้ของแต่ละ

พารามิเตอร์ ซึ่งวิธีการหากรณีทดสอบที่ใช้ขอบเขตของแต่ละค่าในแต่ละพารามิเตอร์ที่นิยมใช้กันมีดังนี้

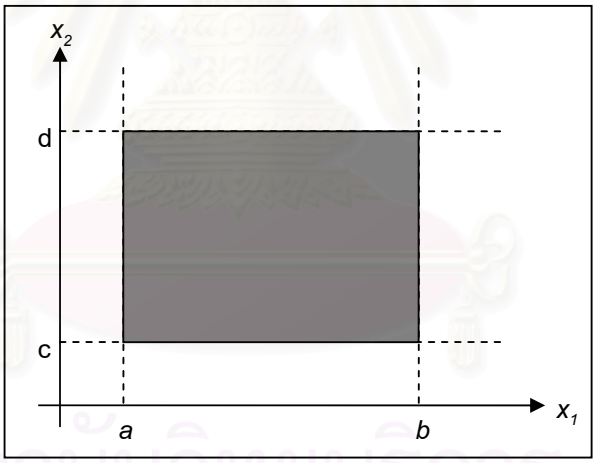
2.1.3.1 การวิเคราะห์ค่าขอบเขต (Boundary Value Analysis: BVA)

การวิเคราะห์ค่าขอบเขต [4] สนใจที่ขอบเขตของค่าของข้อมูลนำเข้า (input) ซึ่งจะนำไปสร้างกรณีทดสอบ แนวคิดเบื้องต้นของการวิเคราะห์ค่าขอบเขตเป็นการทดสอบค่าของข้อมูลนำเข้าที่มีค่าต่ำสุด (min) ค่าสูงกว่าค่าขอบเขตล่าง (min+) ค่าระหว่างกลางของข้อมูลนำเข้า (nominal หรือ nom) ค่าต่ำกว่าค่าขอบเขตบน (max-) และค่าสูงสุด (max)

สมมติ ความสัมพันธ์ของฟังก์ชัน F กับสองตัวแปร X_1 และ X_2 ซึ่งมีช่วงค่าของข้อมูลนำเข้าของตัวแปร ดังนี้

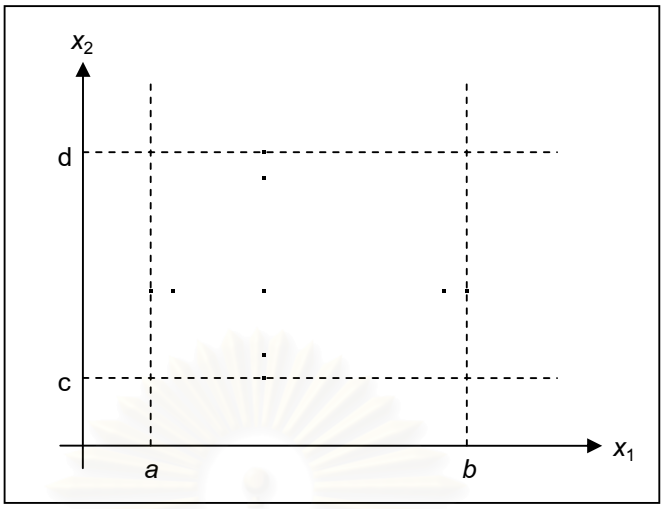
$$a \leq X_1 \leq b$$
$$c \leq X_2 \leq d$$

พื้นที่ค่าของข้อมูลนำเข้าในฟังก์ชัน F แสดงดังรูปที่ 2.2



รูปที่ 2.2 แสดงขอบเขตของค่าที่ใช้ได้ของพารามิเตอร์ X_1 และ X_2

จากรูปที่ 2.2 พื้นที่แรเงาแสดงถึงขอบเขตของค่าที่ใช้ได้ที่จะนำเข้ามาใช้เป็นค่าของพารามิเตอร์ในฟังก์ชัน F คือ พารามิเตอร์ X_1 มีขอบเขตของค่าที่ใช้ได้ตั้งแต่ค่า a ถึงค่า b และพารามิเตอร์ X_2 มีขอบเขตของค่าที่ใช้ได้ตั้งแต่ค่า c ถึงค่า d และพื้นที่ส่วนที่อยู่ภายนอกแสดงถึงค่าที่ใช้ไม่ได้หากนำมาใช้เป็นค่าของพารามิเตอร์ ในฟังก์ชัน F



รูปที่ 2.3 แสดงจุดที่ถูกเลือกเป็นกรณีทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต

จากรูปที่ 2.3 จะได้วิธีการทดสอบ ดังนี้

{ <X_1nom,X_2min>, <X_1nom,X_2min+>, <X_1nom,X_2nom>, <X_1nom,X_2max->, <X_1nom,X_2max>, <X_1min,X_2nom>, <X_1min+,X_2nom>,<X_1nom,X_2nom>, <X_1max-,X_2nom>, <X_1max,X_2nom> }

จะเห็นได้ว่ากรณีทดสอบ <X_1nom,X_2nom> มีซ้ำกัน 2 กรณีทดสอบ ดังนั้นจะมีกรณีทดสอบสำหรับกรณี 2 พารามิเตอร์นี้ทั้งหมด 9 กรณีทดสอบ

สำหรับฟังก์ชันที่มีพารามิเตอร์ n ตัว จะมีจำนวนกรณีทดสอบที่สร้างด้วยวิธีการวิเคราะห์ค่าขอบเขต เป็น $4n+1$ กรณีทดสอบ

วิธีการวิเคราะห์ค่าขอบเขตจะใช้ได้ดีในกรณีที่ตัวแปรแต่ละตัวที่นำมาทำการหากรณีทดสอบไม่มีความสัมพันธ์กัน

ตัวอย่างที่ 2.1

แสดงกรณีทดสอบสำหรับปัญหาสามเหลี่ยม ซึ่งกำหนดให้ใส่ค่าด้านแต่ละด้านของสามเหลี่ยมเข้าไป เพื่อให้แสดงผลออกมาเป็นประเภทของสามเหลี่ยมนั้นๆ

เงื่อนไขสำหรับปัญหาสามเหลี่ยม

- ค่าที่เป็นไปได้ของด้านแต่ละด้านของสามเหลี่ยม ต้องเป็นค่าตัวเลข (integer) มีค่าตั้งแต่ 1 ถึง 200
- ผลรวมของสองด้านต้องมากกว่าด้านที่เหลือจึงเป็นสามเหลี่ยม
- ด้านทั้งสามด้านเท่ากัน ได้ผลลัพธ์เป็น *สามเหลี่ยมด้านเท่า*
- ถ้ามีสองด้านเท่ากันแต่ไม่เท่ากับด้านที่สาม ได้ผลลัพธ์เป็น *สามเหลี่ยมหน้าจั่ว*

ตารางที่ 2.1 แสดงกรณีทดสอบของปัญหาสามเหลี่ยมพร้อมทั้งผลลัพธ์ที่คาดว่าจะเกิดขึ้นเมื่อใส่ค่าที่เป็นไปได้ให้กับด้านแต่ละด้านของสามเหลี่ยม ด้วยวิธีการวิเคราะห์ค่าขอบเขตตามเงื่อนไขที่ได้กำหนดไว้

ตารางที่ 2.1 กรณีทดสอบปัญหาสามเหลี่ยม ด้วยวิธีการวิเคราะห์ค่าขอบเขต

กรณีทดสอบ	ด้านที่ 1	ด้านที่ 2	ด้านที่ 3	ผลลัพธ์ที่คาดหวัง
1	100	100	1	สามเหลี่ยมหน้าจั่ว
2	100	100	2	สามเหลี่ยมหน้าจั่ว
3	100	100	100	สามเหลี่ยมด้านเท่า
4	100	100	199	สามเหลี่ยมหน้าจั่ว
5	100	100	200	ไม่เป็นสามเหลี่ยม
6	100	1	100	สามเหลี่ยมหน้าจั่ว
7	100	2	100	สามเหลี่ยมหน้าจั่ว
8	100	100	100	สามเหลี่ยมด้านเท่า
9	100	199	100	สามเหลี่ยมหน้าจั่ว
10	100	200	100	ไม่เป็นสามเหลี่ยม
11	1	100	100	สามเหลี่ยมหน้าจั่ว
12	2	100	100	สามเหลี่ยมหน้าจั่ว
13	100	100	100	สามเหลี่ยมด้านเท่า
14	199	100	100	สามเหลี่ยมหน้าจั่ว
15	200	100	100	ไม่เป็นสามเหลี่ยม

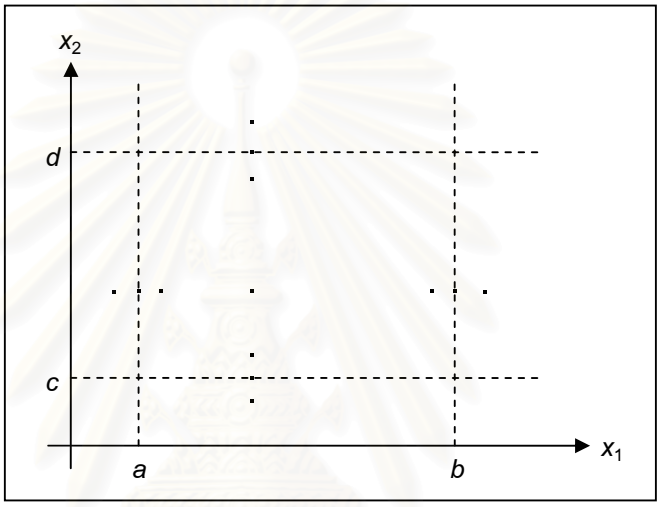
2.1.3.2 การทดสอบสภาพทนทาน (robustness testing)

การทดสอบสภาพทนทาน [4] ได้มีการเพิ่มเติมจุดที่สนใจจากวิธีการวิเคราะห์ค่าขอบเขต โดยนำค่าที่ใช้ไม่ได้ของพารามิเตอร์แต่ละตัวมาใช้เป็นกรณีทดสอบ ได้แก่ ค่ามากกว่าค่าขอบเขตบน (max+) และค่าน้อยกว่าค่าขอบเขตล่าง (min-) เพื่อที่จะได้รู้ถึงผลลัพธ์ที่จะเกิดขึ้นหากนำค่าที่อยู่นอกเหนือค่าที่ใช้ได้มาใช้ในระบบนั้นๆ

กรณีทดสอบสภาพทนทานได้เพิ่มเติมจุดที่ถูกเลือกเป็นกรณีทดสอบจากวิธีการวิเคราะห์ค่าขอบเขตในรูปที่ 2.3 เป็นดังรูปที่ 2.4

สิ่งที่การทดสอบสภาพทนทานได้ให้ความสนใจมาก ไม่ใช่ค่าที่นำเข้าไปในฟังก์ชัน *F* แต่ให้ความสนใจต่อผลที่คาดว่าจะเกิดขึ้นจากฟังก์ชัน *F* ซึ่งการทดสอบลักษณะนี้ จะบอกได้ว่า อะไรจะเกิดขึ้นหากใช้ค่าที่มากหรือน้อยเกินไป เช่น ถ้าน้ำหนักที่ลิฟท์บรรทุกมากเกินไปจะเกิดภาระเกิน (overload) หรือถ้าจำนวนวันในเดือนพฤษภาคมมี 32 วัน จะแสดงข้อความเพื่อเตือนถึงข้อผิดพลาด เป็นต้น

เป้าหมายหลักของการทดสอบสภาพทนทานนั้นให้ความสำคัญที่การจับข้อผิดพลาดที่อาจเกิดขึ้น



รูปที่ 2.4 แสดงจุดที่ถูกเลือกเป็นกรณีทดสอบด้วยวิธีการทดสอบสภาพทนทาน

จากรูปที่ 2.4 จะได้วิธีการทดสอบ ดังนี้

{ <X_1nom,X_2min->, <X_1nom,X_2min>, <X_1nom,X_2min+>, <X_1nom,X_2nom>,
<X_1nom,X_2max->,<X_1nom,X_2max>, <X_1nom,X_2max+>, <X_1min-,X_2nom>,
<X_1min,X_2nom>, <X_1min+,X_2nom>, <X_1nom,X_2nom>, <X_1max-,X_2nom>,
<X_1max,X_2nom>, <X_1max+,X_2nom> }

จากตัวอย่างที่ 2.1 นำมาทดสอบปัญหาสามเหลี่ยมด้วยกรณีทดสอบจากวิธีการทดสอบสภาพทนทานทำให้มีกรณีทดสอบเพิ่มขึ้นมาจากการทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต ดังตารางที่ 2.2

ตารางที่ 2.2 กรณีทดสอบปัญหาสามเหลี่ยม ด้วยวิธีการทดสอบสภาพทนทาน

กรณีทดสอบ	ด้านที่ 1	ด้านที่ 2	ด้านที่ 3	ผลลัพธ์ที่คาดหวัง
1	100	100	0	แสดงข้อผิดพลาด
2	100	100	1	สามเหลี่ยมหน้าจั่ว
3	100	100	2	สามเหลี่ยมหน้าจั่ว
4	100	100	100	สามเหลี่ยมด้านเท่า
5	100	100	199	สามเหลี่ยมหน้าจั่ว
6	100	100	200	ไม่เป็นสามเหลี่ยม
7	100	100	201	แสดงข้อผิดพลาด
8	100	0	100	แสดงข้อผิดพลาด
9	100	1	100	สามเหลี่ยมหน้าจั่ว
10	100	2	100	สามเหลี่ยมหน้าจั่ว
11	100	100	100	สามเหลี่ยมด้านเท่า
12	100	199	100	สามเหลี่ยมหน้าจั่ว
13	100	200	100	ไม่เป็นสามเหลี่ยม
14	100	201	100	แสดงข้อผิดพลาด
15	0	100	100	แสดงข้อผิดพลาด
16	1	100	100	สามเหลี่ยมหน้าจั่ว
17	2	100	100	สามเหลี่ยมหน้าจั่ว
18	100	100	100	สามเหลี่ยมด้านเท่า
19	199	100	100	สามเหลี่ยมหน้าจั่ว
20	200	100	100	ไม่เป็นสามเหลี่ยม
21	201	100	100	แสดงข้อผิดพลาด

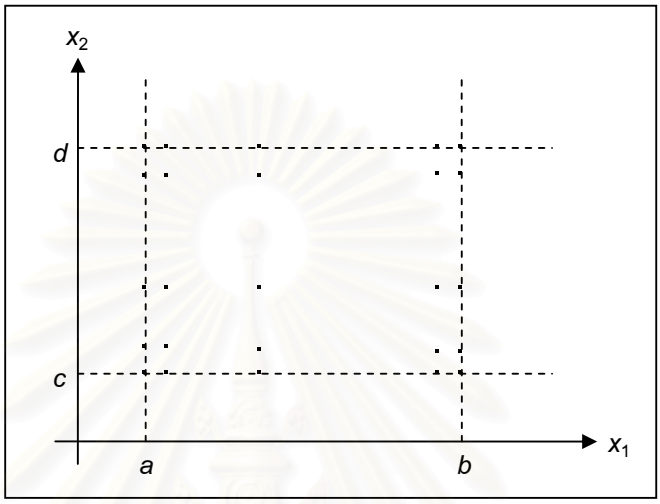
2.1.3.3 การทดสอบกรณีเลวร้ายที่สุด (worst-case testing)

การทดสอบกรณีเลวร้ายที่สุด [4] ให้ความสนใจต่อผลที่เกิดจากพารามิเตอร์มากกว่าหนึ่งตัวมีค่าของข้อมูลนำเข้าเป็นที่สุด คือ ค่าของข้อมูลนำเข้าน้อยที่สุด ค่าของข้อมูลนำเข้าสูงที่สุด เช่น พารามิเตอร์ *A* มีค่าของข้อมูลนำเข้าสูงสุด และพารามิเตอร์ *B* มีค่าของข้อมูลนำเข้าสูงสุดในกรณีทดสอบเดียวกัน เป็นต้น

กรณีทดสอบที่สร้างขึ้นด้วยวิธีการวิเคราะห์ค่าขอบเขตดังที่ได้กล่าวไปแล้วเป็นเซตย่อย (subset) ของกรณีทดสอบที่สร้างขึ้นด้วยวิธีการทดสอบกรณีเลวร้ายที่สุด และจำนวน

กรณีทดสอบสำหรับฟังก์ชันที่มี n พารามิเตอร์ ที่สร้างขึ้นด้วยวิธีการทดสอบกรณีเลวร้ายที่สุดมีทั้งหมด 5^n กรณีทดสอบ

รูปแบบทั่วไปของการหากรณีทดสอบด้วยวิธีการทดสอบกรณีเลวร้ายที่สุดสามารถดูได้จากการวิเคราะห์ค่าขอบเขต



รูปที่ 2.5 แสดงจุดที่ถูกเลือกเป็นกรณีทดสอบด้วยวิธีการทดสอบกรณีเลวร้ายที่สุด

จากรูปที่ 2.5 จะได้วิธีการทดสอบ ดังนี้

{ <X_1min,X_2max>, <X_1min,X_2max->, <X_1min,X_2nom>, <X_1min,X_2min+>,
<X_1min,X_2min>, <X_1min+,X_2max>, <X_1min+,X_2max->,<X_1min+,X_2nom>,
<X_1min+,X_2min+>, <X_1min+,X_2min>, <X_1nom,X_2max>, <X_1nom,X_2max->,
<X_1nom,X_2nom>, <X_1nom,X_2min+>, <X_1nom,X_2min>, <X_1max-,X_2max>,
<X_1max-,X_2max->, <X_1max-,X_2nom>, <X_1max-,X_2min+>,<X_1max-,X_2min>,
<X_1max,X_2max>, <X_1max,X_2max->, <X_1max,X_2nom>, <X_1max,X_2min+>,
<X_1max,X_2min> } □

2.1.3.4 การทดสอบค่าพิเศษ (special value testing)

การทดสอบค่าพิเศษ [4] เป็นการทดสอบซอฟต์แวร์ที่มีการใช้กันอย่างแพร่หลาย โดยเป็นวิธีที่จะใช้การทดสอบฟังก์ชัน ซึ่งเป็นวิธีที่จะใช้ความคิดของผู้ทำการทดสอบและมีความเป็นรูปแบบน้อยมาก ดังนั้นการทดสอบค่าพิเศษจะทำได้ก็ต่อเมื่อผู้ทำการทดสอบมีความรู้ และประสบการณ์ในลักษณะที่เกี่ยวข้องกับซอฟต์แวร์ที่กำลังพัฒนา

ผลที่ได้จากการทดสอบค่าพิเศษจะขึ้นอยู่กับความสามารถของผู้ทำการทดสอบ คือ หากผู้ทำการทดสอบมีประสบการณ์ในลักษณะเดียวกันกับซอฟต์แวร์ที่กำลังพัฒนา ก็จะใช้กรณีทดสอบจำนวนน้อยในการทดสอบซอฟต์แวร์ที่กำลังพัฒนา และหากผู้ทำการทดสอบมี

ประสบการณ์ที่เกี่ยวข้องกับซอฟต์แวร์ที่กำลังพัฒนาน้อย อาจต้องใช้กรณีทดสอบจำนวนมาก เพื่อให้สามารถทดสอบซอฟต์แวร์ให้ครอบคลุมที่กรณีของซอฟต์แวร์ที่กำลังพัฒนา

2.1.3.5 การทดสอบอย่างสุ่ม (random testing)

การทดสอบอย่างสุ่ม [4] เป็นการทดสอบที่น่าศึกษามากที่สุด และอยู่ในเชิงสถิติ ซึ่งการทดสอบอย่างสุ่มจะใช้เครื่องมือในการหากรณีทดสอบออกมาโดยมีแนวคิดพื้นฐานมากจากการวิเคราะห์ค่าขอบเขต และการหากรณีทดสอบนี้ ต้องระวังอคติที่อาจจะเกิดขึ้นในขณะทำการทดสอบ เช่น ผู้ทำการทดสอบอาจใช้ความรู้สึกส่วนตัวในการเลือกกรณีทดสอบ ทำให้ผลที่ได้จากการทดสอบไม่ถูกต้อง เป็นต้น

ปัญหาที่สำคัญในการสร้างกรณีทดสอบด้วยวิธีการทดสอบอย่างสุ่ม คือ จะต้องใช้กรณีทดสอบจำนวนเท่าใดจึงทดสอบซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

2.1.4 อัลกอริทึมเชิงละโมบ (greedy algorithm)

อัลกอริทึมเชิงละโมบ [14] เป็นวิธีการแก้ไขปัญหาอีกวิธีหนึ่งซึ่งใช้กับปัญหาการหาค่าเหมาะที่สุด โดยที่ผลเฉลยของปัญหาหาได้จากลำดับของการตัดสินใจ แต่จะตัดสินใจด้วยการใช้เกณฑ์ที่ให้ผลเหมาะที่สุดจากสภาพของปัญหา ณ ขณะนั้น ได้เป็นผลเฉลยบางส่วน แล้วค่อยลดขนาดของปัญหาลงเพื่อตัดสินใจต่อ เพื่อได้ผลเฉลยบางส่วนที่มีขนาดใหญ่ด้วยวิธีการเดียวกัน กระทำเช่นนี้ไปเรื่อยๆ จนได้ผลเฉลยที่สมบูรณ์ ดังตัวอย่างที่ 2.2

ตัวอย่างที่ 2.2

กำหนดให้มีของขายอยู่ n ชิ้นแต่ละชิ้นมีหมายเลข 1, 2, 3, …, n ของชิ้นที่ i มีน้ำหนัก w_i และมีมูลค่า v_i ลูกค้าคนหนึ่งได้รับเลือกให้หยิบของชิ้นใดก็ได้ใส่ถุงเป้ใบหนึ่งซึ่งรับน้ำหนักได้ไม่เกิน W อยากทราบว่าควรเลือกหยิบชิ้นใดบ้าง ถึงจะได้มูลค่ารวมสูงสุด โดยเราสามารถเฉือนแบ่งของที่ขายออกเป็นสัดส่วนได้ตามต้องการ โดยมูลค่าและน้ำหนักของของชิ้นนั้นแปรตามสัดส่วนที่แบ่งออกมา

กำหนดให้ x_i คือ สัดส่วนของของชิ้นที่ *i* ที่หยิบใส่ถุงเป้ โดยที่ $0 \leq x_i \leq 1$ และ $i = 1, 2, \ldots, n$

ถ้า $x_i = 0$ แสดงว่าไม่ได้เลือกของชิ้นที่ *i* ถ้า $x_i = 1$ แสดงว่าเลือกของชิ้นที่ *i* ทั้งชิ้น ถ้า $x_i = 0.25$ แสดงว่าเลือกของชิ้นที่ *i* โดยเฉือนมา ¼ ชิ้น

ดังนั้น ต้องการหาค่าของ $(x_1, x_2, \ldots, x_n)$ ที่มี

$$\sum_{i=1}^{n} v_i x_i \text{ มากที่สุด}$$

โดยที่ $\sum_{i=1}^{n} v_i x_i \leq W$ และ $0 \leq x_i \leq 1$

การทำงานจะเป็นในลักษณะของวงวนของการตัดสินใจเพื่อเลือก และขยายผลเฉลยไปเรื่อยๆ จนได้ผลเฉลยที่สมบูรณ์ สำหรับปัญหานี้จะขอเสนอเกณฑ์ “ความละโมบ” ในการเลือกดังนี้

1. เลือกของที่มีมูลค่าสูงสุด
2. เลือกของที่มีน้ำหนักน้อยสุด
3. เลือกของที่มีมูลค่าต่อน้ำหนักสูงสุด

สมมติว่ามีถุงรับน้ำหนักได้ 100 มีของอยู่ 5 ชิ้นมีน้ำหนัก มูลค่า และมูลค่าต่อน้ำหนักแสดง ดังนี้

ตารางที่ 2.3 น้ำหนัก มูลค่า และมูลค่าต่อน้ำหนักของสินค้า

	1	**2**	**3**	**4**	**5**
v_i	20	30	66	40	60
w_i	10	20	30	40	50
v_i / w_i	2.0	1.5	2.2	1.0	1.2

แบบที่ 1 จะเลือกชิ้นที่ 3 ตามด้วยชิ้นที่ 5 แล้วก็ชิ้นที่ 4 อีกครึ่งชิ้น จะได้มูลค่ารวม 66+60+20 = 146

แบบที่ 2 จะเลือกชิ้นที่ 1 ตามด้วยชิ้นที่ 2, 3 และ 4 จะได้มูลค่ารวม 20+30+66+40 = 146

แบบที่ 3 จะเลือกชิ้นที่ 3 ตามด้วยชิ้นที่ 1 แล้วก็ชิ้นที่ 2 และชิ้นที่ 5 อีก 80% จะได้มูลค่ารวม 66+20+30+48 = 164 □

จากตัวอย่างที่ 2.2 จะเห็นได้ว่าแบบที่ 3 ให้ผลเฉลยที่มีมูลค่ารวมสูงที่สุด แต่อัลกอริทึมเชิงละโมบนั้นไม่สามารถบอกได้ว่าผลเฉลยเหมาะสมที่สุด ทำให้ผลเฉลยที่ได้จากอัลกอริทึมเชิงละโมบนั้น อาจไม่ใช่ผลเฉลยที่เหมาะสมที่สุดก็ได้

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

จากการศึกษา พบว่าที่ผ่านมามีงานวิจัยจำนวนมากที่นำเสนอวิธีการต่างๆ ซึ่งเกี่ยวข้องกับการสร้างกรณีทดสอบเพื่อสร้างกรณีทดสอบให้มีจำนวนน้อยลง แต่กรณีทดสอบที่สร้างขึ้นมานั้นสามารถทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบซอฟต์แวร์ ดังนี้

2.2.1 งานวิจัย "A Test Generation Strategy for Pairwise Testing" โดยคัว ชุง ไถ และหยูเล่ย

เนื่องจากการทดสอบซอฟต์แวร์ เป็นขั้นตอนที่ใช้ระยะเวลาและค่าใช้จ่ายในการดำเนินการมาก ซึ่งปัญหาดังกล่าวจะมีมากหรือน้อย ขึ้นอยู่กับจำนวนกรณีทดสอบที่ต้องนำมาใช้ในการทดสอบ ในปี 1998 คัว ชุง ไถ และหยูเล่ย [9] ได้นำเสนอวิธีไอพีโอ (In-Parameter-Order: IPO) ขึ้น และได้ทำการปรับปรุงแนวคิดนี้ให้สามารถสร้างกรณีทดสอบได้ดียิ่งขึ้นในปี 2002 [10] พร้อมทั้งได้เสนองานวิจัยนี้ ซึ่งเป็นทางใหม่ในการสร้างกรณีทดสอบให้มีจำนวนกรณีทดสอบน้อยลงเมื่อเทียบกับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี สำหรับทดสอบระบบที่ทำการทดสอบด้วยการใส่พารามิเตอร์เข้าไป โดยที่พารามิเตอร์แต่ละตัวในระบบที่จะทำการทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์ และในแต่ละกรณีทดสอบที่สร้างมานั้นจะต้องครอบคลุมอย่างน้อย 1 คู่ของการจับคู่ระหว่างแต่ละค่าที่เป็นไปได้ของแต่ละพารามิเตอร์ ซึ่งแนวคิดในการทดสอบระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ จะทำการพิจารณาพารามิเตอร์ และค่าที่เป็นไปได้ของแต่ละพารามิเตอร์

แนวคิดไอพีโอ ใช้สำหรับสร้างกรณีทดสอบ เพื่อทดสอบระบบที่มีพารามิเตอร์ตั้งแต่ 2 พารามิเตอร์ขึ้นไป ซึ่งแนวคิดนี้ จะแบ่งขั้นตอนในการสร้างกรณีทดสอบออกเป็น 2 ขั้นตอน โดยในขั้นตอนแรก การเติบโตแนวนอน (horizontal growth) จะทำการหากรณีทดสอบของ 3 พารามิเตอร์แรกก่อน โดยจะทำการจับคู่ค่าที่เป็นไปได้แต่ละค่าของ 2 พารามิเตอร์แรก แล้วจึงเพิ่มพารามิเตอร์ตัวที่ 3 เข้าไป และในกรณีที่กรณีทดสอบที่สร้างขึ้นในขั้นตอนแรก ยังไม่ครอบคลุมทุกกรณีที่ต้องการทดสอบ จะทำขั้นตอนที่ 2 การเติบโตแนวตั้ง (vertical growth) จะทำการสร้างกรณีทดสอบเพิ่ม เพื่อให้กรณีทดสอบทั้งหมดที่สร้างขึ้นสามารถทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบในระบบนั้นๆ ซึ่งรายละเอียดอัลกอริทึมไอพีโอ มีดังนี้

Algorithm: IPO_H

Input: All parameters

Output: Test set

Begin

// T is a test set. But T is also treated as a list with elements in arbitrary order.

{ assume that the domain of p_i contains values v_1, v_2, …, and v_q;

π = { pairs between values of p_i and values of p_1, p_2, …, and p_{i-1} };

// *Card*(T) is a cardinality function that returns the number of elements in a list.

if (*Card*(T) $\leq$ q)

{ **for** $1 \leq j \leq$ *Card*(T), extend the jth test in T by adding value v_j and remove from π pairs covered by extended test;

}

else

{ **for** $1 \leq j \leq q$, extend the *j*th test in T by adding value v_j and remove from π pairs covered by extended test;

for $q \leq j \leq Card(T)$, extend the *j*th test in T by adding one value of p_i such that the resulting test covers the most number of pairs in π, and remove from π pairs covered by extended test;

}

}

End

Algorithm: IPO_V

Input: All parameters

Output: Test set

Begin

let T' be an empty set;

for each pair in π

{ assume that the pair contains value w of p_k, $1 \leq k < i$, and value u of p_i;

if (T' contains a test with "-" as the value of p_k and u as the value of p_i)

modify this test by replacing the "-" with w;

else

add a new test to T' that has w as the value of p_k, u as the value of p_i, and "-" as the value of every other parameter;

};

$T = T \cup T'$;

End

ในงานวิจัยนี้ได้นำเสนอแนวคิดไอพีโอ ซึ่งเป็นวิธีการสร้างกรณีทดสอบสำหรับการทดสอบระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ จากการทดลองพบว่ากรณีทดสอบที่สร้างด้วยวิธีไอพีโอมีจำนวนกรณีทดสอบน้อยกว่าจำนวนกรณีทดสอบที่สร้างด้วยวิธีเออีทีจี [5] และยิ่งไปกว่านั้นวิธีไอพีโอยังง่ายต่อการนำกรณีทดสอบที่สร้างไว้กลับมาใช้ใหม่อีกครั้ง (reusing) หากระบบที่นำมาทำการทดสอบมีการเปลี่ยนแปลงค่าพารามิเตอร์ หรือค่าที่เป็นไปได้ของพารามิเตอร์

2.2.2 งานวิจัย “An Improved Test Generation Algorithm for Pair-Wise Testing” โดยโซลเหมิน ไมตี้ และคณะ

ในปี 2005 โซลเหมิน ไมตี้ [6] ได้นำเสนองานวิจัยนี้ โดยทำการปรับปรุงงานวิจัย “A Test Generation Strategy for Pairwise Testing” ของคัว ชุง ไถ และหยู เล่ย ซึ่งในงานวิจัยนี้จะนำเสนอเทคนิคใหม่ที่ใช้สร้างกรณีทดสอบสำหรับทดสอบระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ให้มีจำนวนน้อยลง

แนวคิดในงานวิจัยของโซลเหมิน ไมตี้ และคณะ ได้นำพื้นฐานจากเอ็มโอแอลเอส (Mutually Orthogonal Latin Squares: MOLS) มาใช้ในการพิจารณาเพื่อสร้างกรณีทดสอบสำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ โดยแบ่งวิธีการสร้างกรณีทดสอบออกเป็น 3 กลุ่ม คือ

1. พารามิเตอร์แต่ละพารามิเตอร์มีจำนวนค่าที่เป็นไปได้ 2 ค่า
2. พารามิเตอร์แต่ละพารามิเตอร์มีจำนวนค่าที่เป็นไปได้ *n* ค่า
3. พารามิเตอร์แต่ละพารามิเตอร์มีจำนวนค่าที่เป็นไปได้ไม่เท่ากัน

ตารางที่ 2.4 เปรียบเทียบจำนวนกรณีทดสอบของวิธีใหม่ กับวิธีเออีทีจี และไอพีโอ

ระบบ	เออีทีจี	ไอพีโอ	วิธีของ โซลเหมิน ไมตี้ และคณะ
กรณีศึกษาที่ 1	11	9	9
กรณีศึกษาที่ 2	17	17	15
กรณีศึกษาที่ 3	35	34	24
กรณีศึกษาที่ 4	25	26	24
กรณีศึกษาที่ 5	12	15	10

โดยที่ กรณีศึกษาที่ 1: 4 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า
กรณีศึกษาที่ 2: 13 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า
กรณีศึกษาที่ 3: 61 พารามิเตอร์ (15 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 17 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 29 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)
กรณีศึกษาที่ 4: 75 พารามิเตอร์ (1 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 39 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 35 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)
กรณีศึกษาที่ 5: 100 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า

จากตารางที่ 2.5 แสดงผลการทดลองของงานวิจัยนี้ พบว่าอัลกอริทึมที่ปรับปรุงมาสามารถสร้างกรณีทดสอบได้จำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างด้วยวิธีเออีทีจี และ

อัลกอริทึมไอพีโอ โดยกรณีทดสอบที่สร้างขึ้นนั้นสามารถทดสอบระบบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบเช่นเดียวกัน

2.2.3 งานวิจัย "The AETG System: An Approach to Testing Based on Combinatorial Design" โดย ดาวิด เอ็ม โคเฮน และคณะ

จากปัญหาของการทดสอบซอฟต์แวร์ที่ต้องใช้ระยะเวลา และค่าใช้จ่ายในการดำเนินการสูงมากเมื่อเทียบกับขั้นตอนอื่นในการพัฒนาซอฟต์แวร์ ดังนั้น ในงานวิจัยนี้จึงได้นำเสนอวิธีในการสร้างกรณีทดสอบแบบใหม่ ซึ่งครอบคลุมระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ ทริเปิล และเอ็นเวย์

โดยทางผู้วิจัยได้ทำการพัฒนาระบบเออีทีจี ขึ้นมาในปี 1997 และนำวิธีการใหม่ไปใช้ในกับระบบเบลคอร์ [7] จากการทดลองพบว่ามีบางเงื่อนไขที่ห้ามไม่ให้กรณีทดสอบทำการทดสอบด้วยกรณีทดสอบที่สร้างขึ้น ดังนั้น งานวิจัยนี้ จึงกำหนดเงื่อนไขดังกล่าวเป็นเงื่อนไขบังคับของเออีทีจี (AETG constraints) ขึ้นมาก่อน จากนั้นจึงสร้างกรณีทดสอบขึ้นมาโดยไม่ขัดแย้งกับเงื่อนไขบังคับของเออีทีจี

แนวคิดในการสร้างกรณีทดสอบ จะใช้การประกอบกันทีละตัวค่าที่เป็นไปได้ (values) ของแต่ละพารามิเตอร์ (parameters) โดยค่าที่เป็นไปได้ที่จะนำมาประกอบนั้นจะถูกสุ่ม (random) มาจากค่าที่เป็นไปได้ทุกค่าของพารามิเตอร์นั้นๆ โดยต้องไม่ขัดแย้งกับเงื่อนไขบังคับ โดยที่การประกอบกันของแต่ละค่าที่เป็นไปได้นั้น จะต้องครอบคลุมกรณีที่ต้องการทดสอบได้มากที่สุด และทำการสร้างกรณีทดสอบเพิ่มขึ้น จนสามารถครอบคลุมได้ทุกกรณีที่ต้องการทดสอบ

จากการทดลองสร้างกรณีทดสอบ พบว่าจำนวนกรณีทดสอบที่สร้างได้ในกรณีพารามิเตอร์มีความกันแบบเอ็นเวย์ ส่วนใหญ่จะมีจำนวนมากกว่าการสร้างกรณีทดสอบเมื่อพารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ เพราะว่าในกรณีความสัมพันธ์เอ็นเวย์มีเงื่อนไขมากกว่ากรณีความสัมพันธ์แบบแพร์ไวส์ ทำให้ต้องใช้จำนวนกรณีทดสอบมากกว่าเพื่อทดสอบให้ครบทุกกรณีที่ต้องการ

2.2.4 งานวิจัย "The Automatic Efficient Test Generator (AETG) System" โดย ดาวิด เอ็ม โคเฮน และคณะ

ในงานวิจัยนี้ ได้นำเสนอปัญหาในการสร้างกรณีทดสอบด้วยวิธีการทดสอบหน้าจอ (screen testing) [15] ซึ่งจำนวนกรณีทดสอบที่สร้างจากวิธีการทดสอบหน้าจอ จะมีจำนวนมาก ยกตัวอย่างเช่น ในระบบที่มีพารามิเตอร์ 75 พารามิเตอร์ และแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า จะเป็นไปไม่ได้เลยที่จะใช้วิธีการทดสอบหน้าจอสร้างกรณีทดสอบที่สามารถทดสอบได้ครบทั้ง 2^{75} กรณี ดังนั้น ในงานวิจัยนี้ จึงได้เสนอแนวทางในการแก้ปัญหาดังกล่าวด้วยวิธีเออีทีจี

ซึ่งสามารถสร้างกรณีทดสอบจำนวนไม่มาก แต่สามารถทดสอบได้ครบทุกกรณีที่ต้องการ พร้อมทั้งแสดงผลการเปรียบเทียบกับจำนวนกรณีทดสอบด้วยวิธีแถวลำดับเชิงตั้งฉาก (orthogonal array) ซึ่งผลการทดลองในกรณีศึกษาที่มีพารามิเตอร์ 7 พารามิเตอร์ และแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า เป็นดังนี้

ตารางที่ 2.5 กรณีทดสอบที่สร้างขึ้นด้วยวิธีแถวลำดับเชิงตั้งฉาก

	F1	F2	F3	F4	F5	F6	F7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

ตารางที่ 2.6 กรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี

	F1	F2	F3	F4	F5	F6	F7
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	1
3	1	1	2	2	2	2	2
4	2	1	1	1	2	2	2
5	2	2	2	1	1	1	2
6	1	2	1	2	1	2	2

บทที่ 3

การออกแบบอัลกอริทึมในการสร้างกรณีทดสอบ

ในที่นี้จะเริ่มต้นจากการนำเสนอแนวคิดที่น่าสนใจ โดยประเด็นแรกจะเสนอถึงวิธีการวิเคราะห์เพื่อปรับปรุงอัลกอริทึมไอพีโอ ให้สามารถสร้างกรณีทดสอบได้ด้วยขั้นตอนที่สั้นลงและประเด็นถัดมา พร้อมทั้งนำเสนออัลกอริทึมที่ปรับปรุงจากอัลกอริทึมไอพีโอ เพื่อให้สามารถใช้งานกับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ได้

3.1 แนวคิดในการลดขั้นตอนการสร้างกรณีทดสอบของอัลกอริทึมไอพีโอ

อัลกอริทึมไอพีโอ [9, 10] เป็นอัลกอริทึมที่ใช้สร้างกรณีทดสอบเมื่อทุกพารามิเตอร์ในระบบที่จะทำการทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์เท่านั้น

นิยามที่ 1

ความสัมพันธ์แบบแพร์ไวส์ (pair-wise) *คือ การที่พารามิเตอร์ทุกตัวมีความสัมพันธ์แบบพบกันหมด เมื่อพารามิเตอร์ A มีค่าที่เป็นไปได้ คือ* $\{A_1, A_2, \ldots, A_n\}$ *และพารามิเตอร์ B มีค่าที่เป็นไปได้ คือ* $\{B_1, B_2, \ldots, B_n\}$ *ดังนั้น*

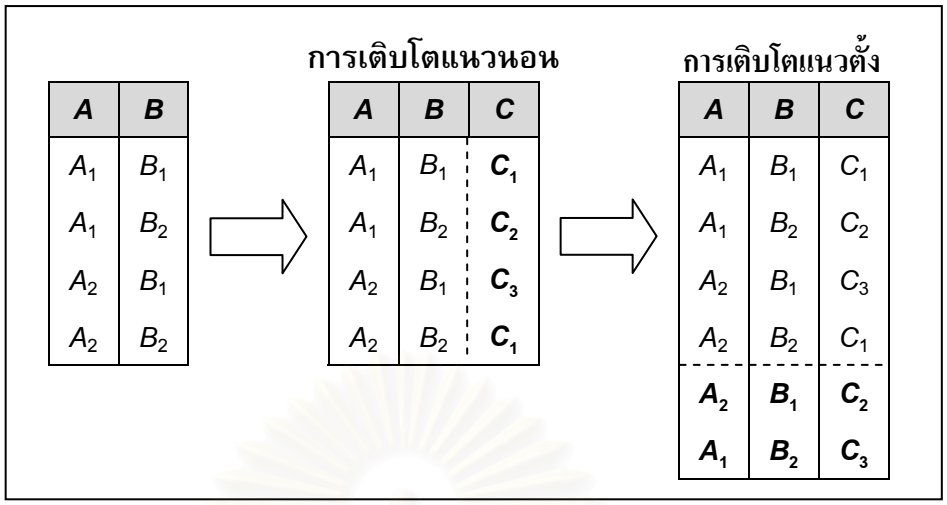
$$A \times B = \{ (A_1,B_1), (A_1,B_2), (A_1,B_3), (A_2,B_1), (A_2,B_2), (A_2,B_3), \ldots , (A_n,B_n) \}$$

นิยามที่ 2

เซตที่ยังไม่ครอบคลุม (uncovered set) *คือ เซตที่มีสมาชิกเป็นคู่ลำดับของค่าที่เป็นไปได้ ซึ่งเกิดจากการพบกันหมดของค่าที่เป็นไปได้ของแต่ละพารามิเตอร์ เมื่อพารามิเตอร์ทุกตัวมีความสัมพันธ์กันแบบแพร์ไวส์ สมมติ พารามิเตอร์ A มีค่าที่เป็นไปได้ คือ* $\{A_1, A_2, \ldots, A_n\}$ *และพารามิเตอร์ B มีค่าที่เป็นไปได้ คือ* $\{B_1, B_2, \ldots, B_n\}$ *ดังนั้น*

เซตที่ยังไม่ครอบคลุม = $\{ (A_1,B_1), (A_1,B_2), (A_1,B_3), (A_2,B_1), (A_2,B_2), (A_2,B_3), \ldots , (A_n,B_n) \}$

ขั้นตอนในการสร้างกรณีทดสอบด้วยอัลกอริทึมไอพีโอ แบ่งออกเป็น 2 ขั้นตอน ได้แก่ ขั้นตอนการเติบโตแนวนอน และขั้นตอนการเติบโตแนวตั้ง ซึ่งขั้นตอนการเติบโตแนวนอนจะสร้างกรณีทดสอบโดยเริ่มจากจับคู่ค่าที่เป็นไปได้ของพารามิเตอร์สองพารามิเตอร์แรกก่อน แล้วจึงทำให้กรณีทดสอบที่สร้างขึ้นสมบูรณ์ด้วยการเพิ่มค่าที่เป็นไปได้ของพารามิเตอร์ในแนวนอน และขั้นตอนการเติบโตแนวตั้ง จะทำการสร้างกรณีทดสอบเพิ่มขึ้นเพื่อให้ครอบคลุมทุกกรณีที่ยังไม่ครอบคลุมด้วยกรณีทดสอบที่ได้จากขั้นตอนการเติมโตแนวนอน ดังรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการสร้างกรณีทดสอบด้วยอัลกอริทึมไอพีโอ

จากแนวคิดการสร้างกรณีทดสอบของอัลกอริทึมไอพีโอ ต่อไปจะแสดงรายละเอียดของอัลกอริทึมไอพีโอที่ใช้ในการสร้างกรณีทดสอบ โดยอัลกอริทึมไอพีโอนี้ จะแบ่งออกเป็น 2 อัลกอริทึมย่อย คือ อัลกอริทึม *IPO_H* สำหรับขั้นตอนการเติบโตแนวนอน และอัลกอริทึม *IPO_V* สำหรับขั้นตอนการเติมโตแนวตั้ง [9, 10]

จากอัลกอริทึมไอพีโอ ในตัวอย่างถัดไปจะแสดงการสร้างกรณีทดสอบ สำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ ด้วยอัลกอริทึมไอพีโอ โดยระบบที่จะนำมาสร้างกรณีทดสอบ มีพารามิเตอร์ 3 ตัว คือ พารามิเตอร์ *A*, *B* และ *C* ซึ่งมีค่าที่เป็นไปได้ ดังตารางที่ 3.1

ตัวอย่างที่ 3.1

สร้างกรณีทดสอบสำหรับทดสอบระบบที่มีพารามิเตอร์ A, B และ C โดยที่ทุกพารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์

ตารางที่ 3.1 ค่าที่เป็นไปได้ของพารามิเตอร์ *A*, *B* และ *C*

A	*B*	*C*
A_1	B_1	C_1
A_2	B_2	C_2
		C_3

กรณีทดสอบที่สร้างขึ้นมาจะครอบคลุมทุก ๆคู่ลำดับในเซตที่ยังไม่ครอบคลุม ซึ่งในตัวอย่างนี้แทนด้วย π จากตารางที่ 3.1 จะได้

π = { (A_1,B_1), (A_1,B_2), (A_2,B_1), (A_2,B_2), (A_1,C_1), (A_1,C_2), (A_1,C_3), (A_2,C_1), (A_2,C_2), (A_2,C_3), (B_1,C_1), (B_1,C_2), (B_1,C_3), (B_2,C_1), (B_2,C_2), (B_2,C_3) }

ขั้นตอนที่ 1 การเติบโตแนวนอน

ในขั้นตอนนี้ จะทำการจับคู่ที่เกิดจากการรวมกันระหว่างสองค่าที่เป็นไปได้ของสองพารามิเตอร์แรก และหลังจากนั้นจะทำการใส่ค่าที่เป็นไปได้ของพารามิเตอร์ถัดไปตามลำดับจนกระทั่งครบทุกค่าที่เป็นไปได้ของพารามิเตอร์ถัดไป

จากตารางที่ 3.1 จะได้คู่ลำดับที่เกิดจากการรวมกันระหว่างสองค่าที่เป็นไปได้ของสองพารามิเตอร์แรก ดังนี้

(A_1,B_1), (A_1,B_2), (A_2,B_1) และ (A_2,B_2)

จากนั้นจะใส่ค่าที่เป็นไปได้ของพารามิเตอร์ถัดไป คือ พารามิเตอร์ C ได้แก่ ค่าที่เป็นไปได้ C_1, C_2 และ C_3 ตามลำดับ

เมื่อใส่ค่าที่เป็นไปได้ของพารามิเตอร์ C จนครบแล้ว จะได้กรณีทดสอบ 3 กรณีทดสอบ ได้แก่ (A_1,B_1,C_1), (A_1,B_2,C_2) และ (A_2,B_1,C_3) จากทั้งสามกรณีทดสอบ จะพิจารณาคู่ลำดับ (A_2,B_2) ที่เหลืออยู่จากการจับคู่ของค่าที่เป็นได้ของพารามิเตอร์สองพารามิเตอร์แรก ซึ่งจะทำการพิจารณาหาค่าที่เป็นไปได้ของพารามิเตอร์ C ที่เหมาะสมกับคู่ลำดับที่เหลืออยู่ โดยพิจารณาจากจำนวนคู่ลำดับใน π ที่สามารถทำการทดสอบได้ครอบคลุมด้วยกรณีทดสอบที่ถูกเติมค่าที่เป็นไปได้ของพารามิเตอร์ C เข้าไปแล้ว

จากการทุกๆการรวมกันของค่าที่เป็นไปได้ของพารามิเตอร์ในกรณีทดสอบ (A_2,B_2,C_1) และ (A_2,B_2,C_2) ซึ่งได้หลังจากเติมค่าที่เป็นไปได้ของพารามิเตอร์ C เข้าไป จะเห็นได้ว่า

(A_2,B_2,C_1) จะครอบคลุม 3 คู่ คือ (A_2,B_2), (A_2,C_1) และ (B_2,C_1)

(A_2,B_2,C_2) จะครอบคลุม 2 คู่ คือ (A_2,B_2) และ (A_2,C_2)

ดังนั้น จะเลือกกรณีทดสอบ (A_2,B_2,C_1) เนื่องจากสามารถนำไปทดสอบได้ครอบคลุมคู่ลำดับใน π มากที่สุด

ในขั้นตอนที่ 1 จะได้กรณีทดสอบทั้งหมด 4 กรณีทดสอบ ได้แก่ (A_1,B_1,C_1), (A_1,B_2,C_2), (A_2,B_1,C_3) และ (A_2,B_2,C_1) โดยที่ทั้ง 4 กรณีทดสอบนั้น ยังไม่ครอบคลุมทุกคู่ลำดับใน π คือ (A_2,C_2), (A_1,C_3), (B_1,C_2) และ (B_2,C_3) ซึ่งจะทำการสร้างกรณีทดสอบเพื่อให้ครอบคลุมคู่ลำดับเหล่านี้ในขั้นตอนถัดไป

ขั้นตอนที่ 2 การเติบโตแนวตั้ง

ในขั้นตอนนี้จะทำการสร้างกรณีทดสอบเพื่อให้ครอบคลุมคู่ลำดับทุกคู่ที่เหลืออยู่ใน π โดยจะพิจารณาคู่ลำดับที่เหลืออยู่ใน π ทีละคู่ ซึ่งจะเริ่มจากคู่ลำดับคู่แรกใน π โดยกำหนดให้สัญลักษณ์ “-” แทนค่าที่เป็นไปได้ของพารามิเตอร์ที่ยังขาดไป จากนั้นจะทำการแทนที่ค่าที่เป็นไปได้ของพารามิเตอร์ที่ยังขาดไปด้วย “-” แล้วเก็บกรณีทดสอบที่ได้ใน T และพิจารณาคู่ลำดับคู่ถัดไปใน π ซึ่งหากค่าที่เป็นไปได้ของคู่ลำดับคู่ถัดไปใน π ไม่ขัดแย้งกับค่าที่เป็นไปได้ค่าใดค่าหนึ่งในกรณีทดสอบ T จะนำมารวมกันกับกรณีทดสอบนั้นๆ ใน T เพื่อให้กรณีทดสอบนั้นๆ ใน T เป็นกรณีทดสอบที่สมบูรณ์มากขึ้น และในกรณีที่ค่าที่เป็นไปได้ของคู่ลำดับคู่ถัดไปใน π ขัดแย้งกับค่าที่เป็นไปได้ค่าใดค่าหนึ่งในกรณีทดสอบ T จะนำคู่ลำดับนั้น มาสร้างเป็นกรณีทดสอบใหม่ โดยแทนค่า “-” ในค่าที่เป็นไปได้ของพารามิเตอร์ที่ยังขาดไป แล้วเก็บกรณีทดสอบที่ได้ใน T ทำซ้ำเช่นนี้ไปจนกว่ากรณีทดสอบใน T จะสามารถครอบคลุมทุกคู่ลำดับใน π

พิจารณาแต่ละคู่ลำดับที่เหลือใน π จากขั้นตอนที่ 1

โดยพิจารณา (A_2,C_2) แทนค่า “-” จะได้ $(A_2,-,C_2)$

(A_1,C_3) แทนค่า “-” จะได้ $(A_1,-,C_3)$

(B_1,C_2) แทนค่า “-” จะได้ $(-,B_1,C_2)$ จะเห็นได้ว่ากรณีทดสอบ $(-,B_1,C_2)$ ขาดค่าที่เป็นไปได้ของพารามิเตอร์ A และกรณีทดสอบ $(A_2,-,C_2)$ ขาดค่าที่เป็นไปได้ของพารามิเตอร์ B ซึ่งทั้งสองกรณีทดสอบมีค่าที่เป็นไปได้ที่ยังขาดไปของอีกกรณีทดสอบ และเนื่องจากทั้งสองกรณีทดสอบมีค่าที่เป็นไปได้ของพารามิเตอร์ C คือ C_2 เหมือนกัน จึงสามารถนำกรณีทดสอบ $(-,B_1,C_2)$ ไปรวมกับกรณีทดสอบ $(A_2,-,C_2)$ ได้ ดังนั้น เมื่อนำกรณีทดสอบทั้งสองมารวมกันแล้ว จะได้เป็นกรณีทดสอบ (A_2,B_1,C_2)

(B_2,C_3) แทนค่า “-” จะได้ $(-,B_2,C_3)$ จะเห็นได้ว่ากรณีทดสอบ $(-,B_2,C_3)$ ขาดค่าที่เป็นไปได้ของพารามิเตอร์ A และกรณีทดสอบ $(A_1,-,C_3)$ ขาดค่าที่เป็นไปได้ของพารามิเตอร์ B ซึ่งทั้งสองกรณีทดสอบมีค่าที่เป็นไปได้ที่ยังขาดไปของอีกกรณีทดสอบ และเนื่องจากทั้งสองกรณีทดสอบมีค่าที่เป็นไปได้ของพารามิเตอร์ C คือ C_3 เหมือนกัน จึงสามารถนำกรณีทดสอบ $(-,B_2,C_3)$ ไปรวมกับกรณีทดสอบ $(A_1,-,C_3)$ ได้ ดังนั้น เมื่อนำกรณีทดสอบทั้งสองมารวมกันแล้ว จะได้เป็นกรณีทดสอบ (A_1,B_2,C_3)

ในขั้นตอนที่ 2 จะได้กรณีทดสอบเพิ่มขึ้นอีก 2 กรณีทดสอบ

ดังนั้น อัลกอริทึมไอพีโอจะสร้างกรณีทดสอบขึ้นมาทั้งหมด 6 กรณีทดสอบ เพื่อให้ทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบ □

จากอัลกอริทึมไอพีโอ งานวิจัยนี้จะทำการปรับปรุงอัลกอริทึมเพื่อลดขั้นตอนในการสร้างกรณีทดสอบ ซึ่งมีแนวคิดในการจัดอันดับข้อมูลนำเข้าใหม่ (input rearrange) โดยจะเรียงลำดับ

จากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้มากที่สุด (maximum number of values) เข้าไปก่อน ซึ่งการเรียงลำดับเช่นนี้ ทำให้กรณีทดสอบที่ถูกสร้างขึ้นในขั้นตอนการเติบโตแนวนอนมีจำนวนกรณีทดสอบมากกว่าการเรียงลำดับจากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้น้อยก่อน ด้วยเหตุนี้ จะทำให้กรณีทดสอบที่ถูกสร้างขึ้นในขั้นตอนนี้ มีโอกาสครอบคลุมทุกคู่ลำดับใน π ได้

ตัวอย่างที่ 3.2

สร้างกรณีทดสอบสำหรับระบบที่มีพารามิเตอร์ A, B และ C ซึ่งมีความสัมพันธ์กันแบบแพร์ไวส์ ซึ่งจะเรียงลำดับข้อมูลนำเข้า โดยเริ่มจากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้มากที่สุดก่อน

ตารางที่ 3.2 ค่าที่เป็นไปได้ของพารามิเตอร์ *A*, *B* และ *C* ที่เรียงลำดับแล้ว

C	***A***	***B***
C_1	A_1	B_1
C_2	A_2	B_2
C_3		

จะสร้างกรณีทดสอบที่สร้างขึ้นมา โดยกรณีทดสอบที่สร้างขึ้นมา จะครอบคลุมทุกๆ คู่ลำดับในเซตที่ยังไม่ครอบคลุม ซึ่งในตัวอย่างนี้แทนด้วย π

จากตารางที่ 3.2 จะได้เซตที่ยังไม่ครอบคลุม ดังนี้

π = { (C_1,A_1), (C_1,A_2), (C_2,A_1), (C_2,A_2), (C_3,A_1), (C_3,A_2), (C_1,B_1), (C_1,B_2), (C_2,B_1), (C_2,B_2), (C_3,B_1), (C_3,B_2), (A_1,B_1), (A_1,B_2), (A_2,B_1), (A_2,B_2) }

ขั้นตอนที่ 1 การเติบโตแนวนอน

ทำการสร้างคู่ลำดับที่เกิดจากการรวมกันระหว่างสองค่าที่เป็นไปได้ของสองพารามิเตอร์แรก หลังจากนั้นจะใส่ค่าที่เป็นไปได้ของพารามิเตอร์ตัวถัดไป

จากตารางที่ 3.2 จะได้คู่ลำดับที่เกิดจากการรวมกันระหว่างสองค่าที่เป็นไปได้ของสองพารามิเตอร์แรก ดังนี้

(C_1,A_1), (C_1,A_2), (C_2,A_1), (C_2,A_2), (C_3,A_1) และ (C_3,A_2)

จากนั้นจะใส่ค่าที่เป็นไปของพารามิเตอร์ *B* ได้แก่ B_1 และ B_2

จะได้กรณีทดสอบ 2 กรณีทดสอบ ได้แก่ (C_1,A_1,B_1) และ (C_1,A_2,B_2) จากกรณีทดสอบทั้งสองกรณีทดสอบ จะเหลือคู่ลำดับใน π คือ { (C_2,A_1), (C_2,A_2), (C_3,A_1), (C_3,A_2), (C_2,B_1), (C_2,B_2), (C_3,B_1), (C_3,B_2), (A_1,B_2), (A_2,B_1) }

จากนั้นจึงพิจารณา (C_2,A_1), (C_2,A_2), (C_3,A_1) และ (C_3,A_2) ที่เหลืออยู่ ซึ่งจะทำการพิจารณาหาค่าที่เป็นไปได้ของพารามิเตอร์ B ที่เหมาะสม โดยดูจากจำนวนคู่ลำดับที่กรณีทดสอบหลังจากเติมพารามิเตอร์ B เข้าไปแล้วครอบคลุมใน π

จากการทุกๆการรวมกันของค่าที่เป็นไปได้ของพารามิเตอร์ในกรณีทดสอบ (C_2,A_1,B_1) และ (C_2,A_1,B_2) ซึ่งได้หลังจากเติมค่าที่เป็นไปได้ของพารามิเตอร์ B เข้าไป จะเห็นได้ว่า

จะเห็นได้ว่า (C_2,A_1,B_1) จะครอบคลุม 2 คู่ (C_2,A_1) และ (C_2,B_1)

(C_2,A_1,B_2) จะครอบคลุม 3 คู่ (C_2,A_1), (C_2,B_2) และ (A_1,B_2)

ดังนั้น จะเลือกกรณีทดสอบ (C_2,A_1,B_2) เนื่องจากมีจำนวนคู่ลำดับที่สามารถครอบคลุมใน π มากที่สุด และทำการพิจารณาหาค่าที่เป็นไปได้ของพารามิเตอร์ B สำหรับ (C_2,A_2), (C_3,A_1) และ (C_3,A_2) จนครบทุกคู่

จะได้กรณีทดสอบที่เหลือ 3 กรณีทดสอบ คือ (C_2,A_2,B_1), (C_3,A_1,B_2) และ (C_3,A_2,B_1)

ในขั้นตอนที่ 1 นี้ จะได้กรณีทดสอบทั้งหมด 6 กรณีทดสอบ คือ (C_1,A_1,B_1), (C_1,A_2,B_2), (C_2,A_1,B_2), (C_2,A_2,B_1), (C_3,A_1,B_2) และ (C_3,A_2,B_1) ซึ่งทั้ง 6 กรณีทดสอบนั้น ครอบคลุมทุกคู่ลำดับใน π ทำให้สามารถสร้างกรณีทดสอบครอบคลุมทุกกรณีที่ต้องการทดสอบ โดยใช้ขั้นตอนการเติบโตแนวนอนของอัลกอริทึมไอพีโอเพียงขั้นตอนเดียวเท่านั้น □

จากตัวอย่างที่ 3.1 และ 3.2 จะเห็นได้ว่าในบางกรณีการเรียงอันดับข้อมูลนำเข้า โดยเรียงลำดับจากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้มากๆ ก่อน มีผลทำให้ขั้นตอนที่ใช้สร้างกรณีทดสอบลดลงจากสองขั้นตอนเหลือเพียงขั้นตอนเดียวเท่านั้น ซึ่งกรณีทดสอบที่สร้างขึ้นนั้นสามารถใช้ในการทดสอบได้ครบทุกกรณีเช่นเดียวกัน

3.2 แนวคิดในการปรับปรุงอัลกอริทึมไอพีโอ

จากแนวคิดในการลดขั้นตอนการสร้างกรณีทดสอบของอัลกอริทึมไอพีโอ งานวิจัยนี้จะนำแนวคิดดังกล่าว ไปประยุกต์ใช้พร้อมทั้งปรับปรุงให้อัลกอริทึมไอพีโอนี้ ให้สามารถใช้สร้างกรณีทดสอบเพื่อทดสอบระบบที่ทุกพารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ โดยที่ผู้ทำการทดสอบจะต้องกำหนดความสัมพันธ์ (relations) ให้เหมาะสมก่อนที่จะนำไปสร้างกรณีทดสอบ

นิยามที่ 3

ความสัมพันธ์ (relations) *หรือ* กลุ่มของพารามิเตอร์ (set of parameters) *ซึ่งแต่ละพารามิเตอร์ในกลุ่มความสัมพันธ์เดียวกัน จะมีความสัมพันธ์กันแบบแพร์ไวส์ และพารามิเตอร์ที่อยู่ต่างกลุ่มความสัมพันธ์กัน ไม่จำเป็นต้องมีความสัมพันธ์กันแบบแพร์ไวส์*

นิยามที่ 4

ความสัมพันธ์แบบเอ็นเวย์ (n-way) *คือ ความสัมพันธ์ของพารามิเตอร์แต่ละพารามิเตอร์สามารถกำหนดเป็นกลุ่มความสัมพันธ์ได้ โดยที่แต่ละพารามิเตอร์จะกำหนดให้อยู่ในกลุ่มความสัมพันธ์ได้มากกว่า 1 กลุ่มความสัมพันธ์ และพารามิเตอร์ใดอยู่ต่างกลุ่มกันจะไม่มีความสัมพันธ์กัน*

*ในกรณีที่มีพารามิเตอร์ใดพารามิเตอร์หนึ่ง อยู่ในกลุ่มความสัมพันธ์มากกว่าหนึ่งกลุ่ม กลุ่มความสัมพันธ์จะมี*ความเหมาะสม (suitable) *ที่จะนำมาสร้างกรณีทดสอบ ก็ต่อเมื่อความสัมพันธ์แต่ละกลุ่มความสัมพันธ์นั้น จะต้องไม่เป็น*เซตย่อย (subset) *ซึ่งกันและกัน*

ตัวอย่างที่ 3.3

กำหนดให้ความสัมพันธ์สองความสัมพันธ์ เป็น G_1 และ G_2 ซึ่งในแต่ละความสัมพันธ์มีพารามิเตอร์เดียวกัน คือ { P_1, P_2, P_3, …, P_n } ให้อธิบายกรณีที่ความสัมพันธ์ทั้งสองมีความเหมาะสม

สมมติ C_1 เป็นเซตของคู่ลำดับที่เกิดจากการจับคู่ทั้งหมดของสองค่าที่เป็นไปได้ใน G_1 และให้ C_2 เป็นเซตของการจับคู่ทั้งหมดของสองค่าที่เป็นไปได้ใน G_2

ถ้า $Card(C_1) > Card(C_2)$ แล้ว

- $C_2 - C_1$ ไม่เป็นเซตว่างแล้ว G_2 เป็นความสัมพันธ์ที่มีความเหมาะสม สามารถนำมาใช้ในการสร้างกรณีทดสอบได้
- $C_2 - C_1$ เป็นเซตว่างแล้ว G_2, Ø, แสดงว่า $G_2 \subseteq G_1$ หรือเรียกว่า ความสัมพันธ์ G_2 เป็นความสัมพันธ์ที่ไม่เหมาะสมในการนำไปใช้สร้างกรณีทดสอบ □

ตัวอย่างที่ 3.4

กำหนดให้ความสัมพันธ์ G_1 มีพารามิเตอร์ { P_1, P_2, P_3, …, P_n } และความสัมพันธ์ G_2 มีพารามิเตอร์ { Q_1, Q_2, Q_3, …, Q_n } ซึ่งทุก ๆพารามิเตอร์ P_i แตกต่างกับทุกพารามิเตอร์ Q_i ให้อธิบายกรณีที่ความสัมพันธ์ทั้งสองมีความเหมาะสม

จะเห็นได้ว่าทุกพารามิเตอร์ใน G_1 ไม่มีความสัมพันธ์กับพารามิเตอร์ใด ๆใน ความสัมพันธ์ G_2 ดังนั้น ความสัมพันธ์ G_1 และ G_2 เป็นความสัมพันธ์ที่มีความเหมาะสมในการนำไปสร้างกรณีทดสอบ □

ตัวอย่างที่ 3.3 และ 3.4 แสดงให้เห็นความสัมพันธ์ที่มีความเหมาะสมที่จะนำไปสร้างกรณีทดสอบด้วยอัลกอริทึมใหม่ที่ทำการปรับปรุงมาจากอัลกอริทึมไอพีโอ

แนวคิดในการปรับปรุงอัลกอริทึมไอพีโอ เพื่อออกแบบอัลกอริทึม ที่สามารถสร้างกรณีทดสอบทั้งสำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ และมีความสัมพันธ์กันแบบเอ็นเวย์ ผู้วิจัยแบ่งแนวทางในการปรับปรุงอัลกอริทึมออกเป็น 3 ขั้นตอน ดังนี้

1. กำหนดเงื่อนไขบังคับ (constraints identification)
2. สร้างกรณีทดสอบร่วม (common test case generation)
3. ทำคำตอบให้สมบูรณ์ (solution completion)

3.2.1 กำหนดเงื่อนไขบังคับ

จากระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ จะมีบางกรณีที่ไม่ให้กรณีทดสอบที่สร้างขึ้นทำการทดสอบบางคู่ลำดับของค่าที่เป็นไปได้ของพารามิเตอร์ที่อยู่ต่างกลุ่มความสัมพันธ์กัน เรียกคู่ลำดับของค่าที่เป็นไปได้ของพารามิเตอร์นั้นว่า *เงื่อนไขบังคับ (constraints)*

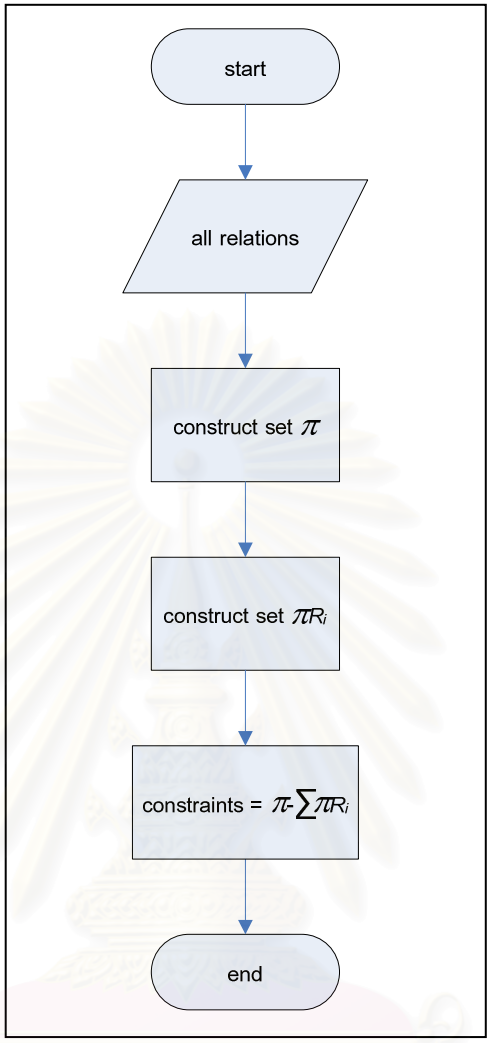
นิยามที่ 5

เงื่อนไขบังคับ (constraints) *จะสร้างขึ้นจากทุกความสัมพันธ์ที่ผู้ทดสอบกำหนดขึ้น ซึ่งอาจมีหลายกรณีที่เป็นไปไม่ได้ หรือไม่ให้กรณีทดสอบที่สร้างขึ้นทำการทดสอบ โดยที่เซตของกรณีที่เป็นไปไม่ได้ เรียกว่า* กรณีที่ไม่มีความจำเป็น (unnecessary cases) *กรณีทดสอบที่สร้างขึ้นสำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ จะไม่สามารถทดสอบกรณีดังกล่าวได้ โดยที่เซตของกรณีที่ไม่มีความจำเป็น เรียกว่า เงื่อนไขบังคับ*

จากนิยามที่ 5 จะเสนออัลกอริทึมสำหรับกำหนดเงื่อนไขบังคับ ดังนี้

Algorithm: constraints identification
Input: *All relations defined on parameters*
Output: *All constraints*
Begin
Construct a set of all combinations of all parameters (π);
Construct a set of all combinations of each relation R_i (πR_i);
Constraints are all combinations that are in π but not in πR_i;
End

จากอัลกอริทึม สามารถเขียนเป็นผังงาน (flowchart) ได้ดังรูปที่ 3.2



รูปที่ 3.2 ผังงานขั้นตอนการกำหนดเงื่อนไขบังคับ

ตัวอย่างที่ 3.5

พิจารณาหาเงื่อนไขบังคับของทุกความสัมพันธ์ที่ผู้ทดสอบเป็นผู้กำหนด โดยกำหนดให้ความสัมพันธ์ เป็นดังตารางที่ 3.3

ตารางที่ 3.3 ตัวอย่างของสองความสัมพันธ์ที่ผู้ทดสอบเป็นผู้กำหนด

ความสัมพันธ์ที่ 1			
A	***B***	***C***	***D***
A_1	B_1	C_1	D_1
A_2	B_2	C_2	D_2
	B_3	C_3	D_3

ความสัมพันธ์ที่ 2			
A	***B***	***C***	***D***
A_3	B_1	C_1	D_1
	B_2	C_2	D_2
		C_3	D_3

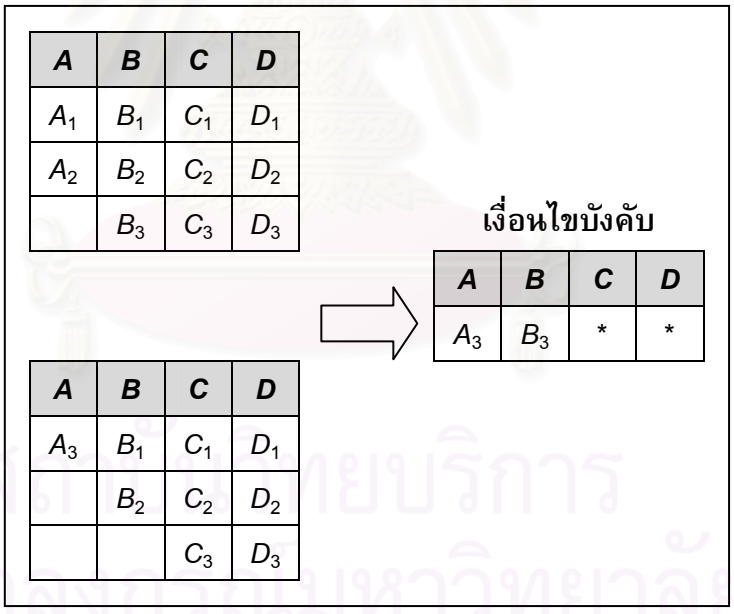
พิจารณาความสัมพันธ์ ในตารางที่ 3.3 จะเห็นได้ว่าทั้งสองความสัมพันธ์มีพารามิเตอร์ *A*, *B*, *C* และ *D* เหมือนกัน แต่มีค่าที่เป็นไปได้ของพารามิเตอร์บางค่าแตกต่างกัน

เมื่อพิจารณาแต่ละพารามิเตอร์ของทั้งสองความสัมพันธ์ พบว่า มีพารามิเตอร์ทั้งหมด 4 พารามิเตอร์ ได้แก่ พารามิเตอร์ *A*, *B*, *C* และ *D* ซึ่งแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า ทำให้เซตของการจับคู่ของค่าที่เป็นไปได้ของพารามิเตอร์ ซึ่งในตัวอย่างนี้แทนด้วย π มีสมาชิกทั้งหมด 54 คู่ เซตของการจับคู่ของค่าที่เป็นไปได้ของความสัมพันธ์ที่ 1 ซึ่งในตัวอย่างนี้แทนด้วย πR_1 มีสมาชิกทั้งหมด 45 คู่ และเซตของการจับคู่ของค่าที่เป็นไปได้ของความสัมพันธ์ที่ 2 ซึ่งในตัวอย่างนี้แทนด้วย πR_2 มีสมาชิกทั้งหมด 29 คู่ จะได้เงื่อนไขบังคับของความสัมพันธ์ที่ 1 และ 2 คือ

$$\pi - (\pi R_1 + \pi R_2) = \{ (A_3, B_3) \}$$

ดังนั้น เงื่อนไขบังคับของความสัมพันธ์ที่ 1 และ 2 มีสมาชิก 1 คู่ คือ (A_3, B_3) □

จากตัวอย่างที่ 3.5 จะสรุปขั้นตอนในการสร้างเงื่อนไขบังคับจากความสัมพันธ์ที่กำหนดให้ ได้ดังรูปที่ 3.3



รูปที่ 3.3 ขั้นตอนการกำหนดเงื่อนไขบังคับ

3.2.2 สร้างกรณีทดสอบร่วม

ขั้นตอนนี้จะสร้างปัญหาย่อย (sub-problem) ขึ้นจากความสัมพันธ์ทั้งหมดที่ต้องการนำมาสร้างกรณีทดสอบ ซึ่งปัญหาย่อยที่จะสร้างนั้น ต้องไม่มีพารามิเตอร์ของค่าที่เป็นไปได้ทั้งสองตัวในเงื่อนไขบังคับอยู่ในปัญหาย่อยเดียวกัน กล่าวคือ หากในเงื่อนไขบังคับมีคู่ของค่าที่เป็นไปได้ของพารามิเตอร์ *A* และ *B* ปัญหาย่อยจะต้องไม่มีพารามิเตอร์ *A* และ *B* อยู่

ด้วยกัน ซึ่งจะนำเสนออัลกอริทึมที่ใช้ในการสร้างปัญหาย่อยที่จะนำมาใช้ในการสร้างกรณีทดสอบ ดังนี้

Algorithm: Sub-problem construction

Input: ***All relations***

Constraints

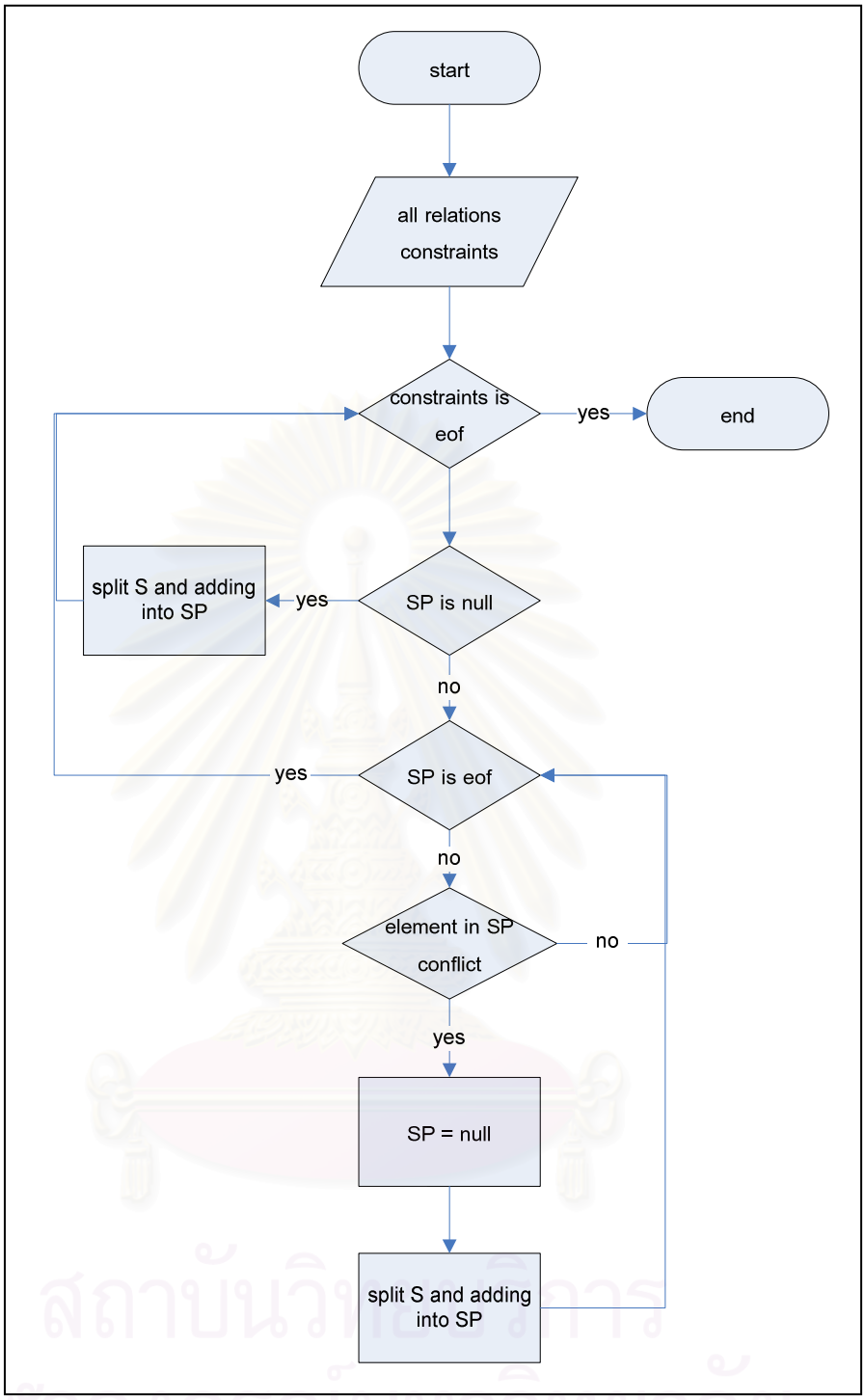
Output: *All set of parameters (sub-problem)*

Begin

let *S* is set of all parameters;
let *SP* is empty set;
let *STEMP* is empty set;
for (each pairs in constraints)
{ **if** (*SP* = null)
{ split set of parameter *S* into sets of parameters *SP* which each elements in *SP* must not contain both parameters in same pair of constraints;
}
else
{ **for** (each elements in *SP*)
{ **if** (element *i*th in *SP* conflict with constraints)
{ *STEMP* = *SP*;
SP = null;
split set of parameter *S* into sets of parameters *SP* which each elements in *SP* must not contain both parameters in same pair of constraints;
};
};
};
};

End

จากอัลกอริทึม สามารถเขียนเป็นผังงานขั้นตอนการสร้างปัญหาย่อย ได้ดังรูปที่ 3.4



รูปที่ 3.4 ผังงานขั้นตอนการสร้างปัญหาย่อย

ตัวอย่างที่ 3.6

พิจารณาสร้างปัญหาย่อยของความสัมพันธ์ในตารางที่ 3.3 ด้วยอัลกอริทึมการสร้างปัญหาย่อย

จากตัวอย่างที่ 3.5 จะได้เงื่อนไขบังคับ คือ (A_3,B_3) ซึ่ง A_3 เป็นค่าที่เป็นไปได้ของพารามิเตอร์ *A* และ B_3 เป็นค่าที่เป็นไปได้ของพารามิเตอร์ *B* ดังนั้น เซตของปัญหาย่อย จะต้องไม่มีพารามิเตอร์ *A* และ *B* อยู่ในปัญหาย่อยเดียวกัน

ตารางที่ 3.4 ปัญหาย่อยของความสัมพันธ์ที่ 1 และ 2

A	***C***	***D***
A_1	C_1	D_1
A_2	C_2	D_2
A_3	C_3	D_3

B	***C***	***D***
B_1	C_1	D_1
B_2	C_2	D_2
B_3	C_3	D_3

จากตารางที่ 3.4 ปัญหาย่อยที่สร้างได้ จะมี 2 ปัญหาย่อย ได้แก่ ปัญหาย่อยที่มีพารามิเตอร์ *A*, *C*, *D* และปัญหาย่อยที่มีพารามิเตอร์ *B*, *C*, *D* □

เมื่อสร้างปัญหาย่อยจากอัลกอริทึมการสร้างปัญหาย่อย แล้วในกรณีที่ปัญหาย่อยที่ถูกสร้างขึ้นมา มีมากกว่าหนึ่งปัญหาย่อย ดังในตัวอย่างที่ 3.6 มีปัญหาย่อยถึง 2 ปัญหาย่อย จะทำการเลือกปัญหาย่อยที่มีจำนวนพารามิเตอร์มากที่สุดเพื่อสร้างกรณีทดสอบในขั้นตอนถัดไป

หลังจากได้ปัญหาย่อยมาแล้ว จะนำไปสร้างกรณีทดสอบ เรียกกรณีทดสอบที่สร้างขึ้นมาจากปัญหาย่อยว่า *กรณีทดสอบร่วม (common test case)* โดยจะเริ่มจากจัดอันดับพารามิเตอร์ของปัญหาย่อยใหม่ โดยพิจารณาจากจำนวนค่าที่เป็นไปได้ของพารามิเตอร์ ซึ่งในแนวคิดนี้ จะเรียงลำดับพารามิเตอร์ จากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้มากที่สุดไปน้อยที่สุด ตามลำดับ จากนั้นจึงสร้างกรณีทดสอบของปัญหาย่อยด้วยอัลกอริทึมไอพีโอ ดังอัลกอริทึมต่อไปนี้

กำหนดให้ R_i มีพารามิเตอร์ P_1,P_2,…,P_m ซึ่ง P_i มีค่าที่เป็นไปได้ $v_{i,}1$, $v_{i,}2$, …, $v_{i,ni}$

Algorithm: Common test case generation

Input: Selected sub-problem

Output: All test cases

Begin

let *MIR* is sub-problem;

let πIR is uncover set of sub-problem;

rearrange parameters of sub-problem;

// *Horizontal growth*

let T be an empty set;

generate all combination of values of the first two parameters of *MIR*, and adding pairs to T;

if ($Card(T) \leq ni$)

{ **for** $1 < j \leq Card(T)$

extend the jth test in T by adding value v_j and remove from πIR pairs covered by the extended test;

}

else

for $1 < j \leq ni$

extend the jth test in T by adding value v_j and remove from πIR pairs covered by the extended test;

for $ni < j \leq Card(T)$

extend the jth test in T by adding one value of P_i such that the resulting test covers the most number of pairs in πIR, and remove from πIR pairs covered by the extended test;

// *Vertical growth*

let T' be an empty set;

for (each pair in πIR)

{ assume that the pairs contains value w of P_k,

$1 \leq k <$ i, and value u of P_i;

if (T' contains a test with "-" as the value of P_k and u as the value of P_i)

modify this test by replacing the "-" with w;

else

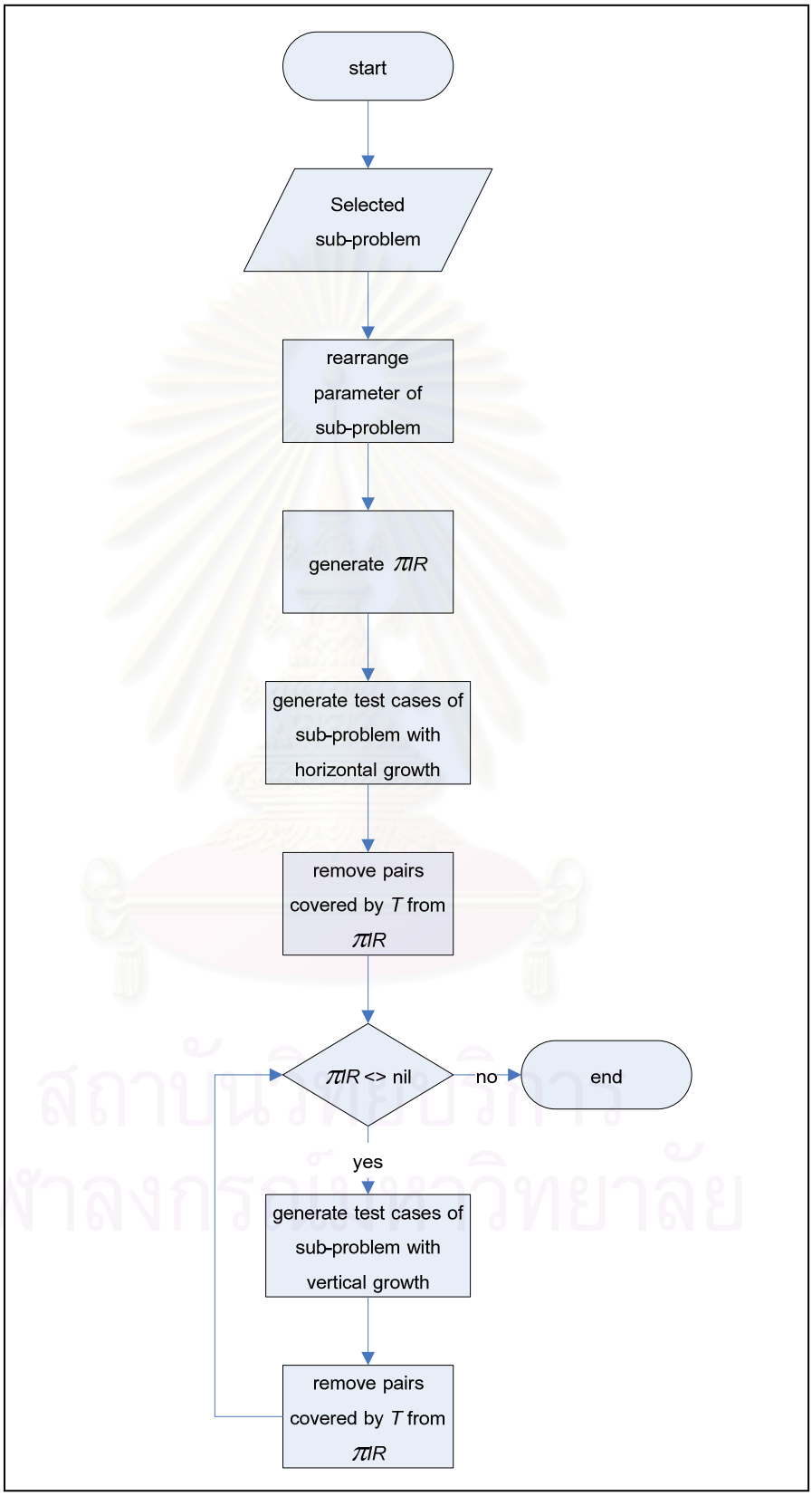
add a new test to T' that has w as the value of P_k, u as the value of P_i, and "-" as the value of every other parameter;

}

$T = T \cup T'$;

End

จากอัลกอริทึมการสร้างกรณีทดสอบของปัญหาย่อยที่นำเสนอ สามารถเขียนเป็นผังงานขั้นตอนการสร้างกรณีทดสอบของปัญหาย่อย ได้ดังรูปที่ 3.5



รูปที่ 3.5 ผังงานขั้นตอนการสร้างกรณีทดสอบของปัญหาย่อย

ต่อไปนี้จะนำเสนอตัวอย่างการสร้างกรณีทดสอบของปัญหาย่อย ในตัวอย่างที่ 3.7

ตัวอย่างที่ 3.7

สร้างกรณีทดสอบร่วมของปัญหาย่อย ที่ได้สร้างในตัวอย่างที่ 3.6 ด้วยอัลกอริทึมไอพีโอ ที่ทำการปรับปรุงลำดับของข้อมูลนำเข้าแล้ว

จากตัวอย่างที่ 3.6 จะเห็นได้ว่ามีปัญหาย่อยที่มีจำนวนพารามิเตอร์มากที่สุดมีมากกว่าหนึ่งปัญหาย่อย ดังนั้น จะทำการเลือกตัวใดตัวหนึ่ง เพื่อมาทำการสร้างกรณีทดสอบร่วม ดังตารางที่ 3.5

ตารางที่ 3.5 ปัญหาย่อยที่ต้องนำมาสร้างกรณีทดสอบ

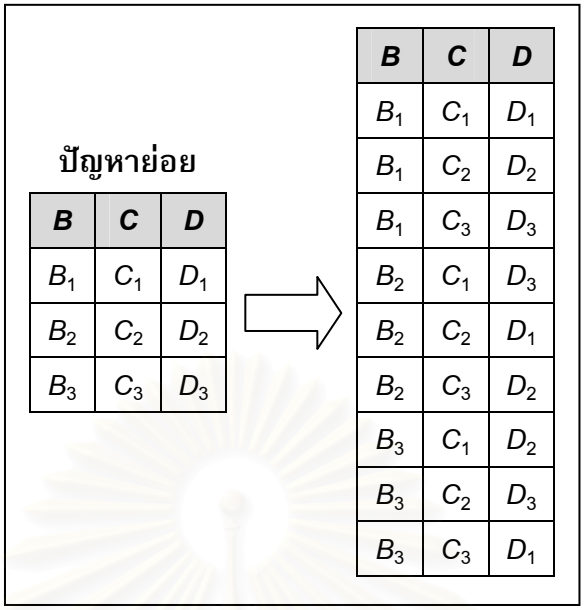
B	***C***	***D***
B_1	C_1	D_1
B_2	C_2	D_2
B_3	C_3	D_3

หลังจากสร้างปัญหาย่อยได้แล้ว จากแนวทางการลดขั้นตอนการสร้างกรณีทดสอบสำหรับอัลกอริทึมไอพีโอ จะทำการจัดอันดับใหม่ให้กับพารามิเตอร์ของปัญหาย่อยก่อน

จากนั้น จะทำการสร้างกรณีทดสอบร่วมของปัญหาย่อยด้วยอัลกอริทึมไอพีโอ โดยเริ่มจากการหาเซตที่ยังไม่ครอบคลุมของปัญหาย่อยก่อน จากตารางที่ 3.5 จะได้คู่ลำดับในเซตที่ยังไม่ครอบคลุมทั้งหมด 27 คู่ ได้แก่ (B_1,C_1), (B_1,C_2), (B_1,C_3), (B_2,C_1), (B_2,C_2), (B_2,C_3), (B_3,C_1), (B_3,C_2), (B_3,C_3), (B_1,D_1), (B_1,D_2), (B_1,D_3), (B_2,D_1), (B_2,D_2), (B_2,D_3), (B_3,D_1), (B_3,D_2), (B_3,D_3), (C_1,D_1), (C_1,D_2), (C_1,D_3), (C_2,D_1), (C_2,D_2), (C_2,D_3), (C_3,D_1), (C_3,D_2) และ (C_3,D_3)

จากนั้นจะสร้างกรณีทดสอบร่วมเพื่อให้ครอบคลุมทุกคู่ลำดับ ซึ่งจากอัลกอริทึมไอพีโอจะได้กรณีทดสอบ 9 กรณีทดสอบ ได้แก่ (B_1,C_1,D_1), (B_1,C_2,D_2), (B_1,C_3,D_3), (B_2,C_1,D_3), (B_2,C_2,D_1), (B_2,C_3,D_2), (B_3,C_1,D_2), (B_3,C_2,D_3) และ (B_3,C_3,D_1) □

จากตัวอย่างที่ 3.7 จะสรุปขั้นตอนในการสร้างกรณีทดสอบของปัญหาย่อยที่ได้กำหนดขึ้นในขั้นตอนนี้ แสดงดังรูปที่ 3.6



รูปที่ 3.6 ขั้นตอนการสร้างกรณีทดสอบร่วมของปัญหาย่อย

3.2.3 ทำคำตอบให้สมบูรณ์

จากกรณีทดสอบร่วมที่ได้ ยังขาดบางพารามิเตอร์ที่ขัดแย้งกับในเงื่อนไขบังคับ จะนำกรณีทดสอบนั้น มาทำให้เป็น *กรณีทดสอบที่สมบูรณ์ (complete test cases)* เพื่อให้สามารถนำไปทดสอบได้ครอบคลุมทุกกรณีของความสัมพันธ์ทุกความสัมพันธ์

เริ่มต้นต้องสร้างคู่ลำดับของค่าที่เป็นไปได้ที่ยังไม่ครอบคลุมทั้งหมดก่อน จากนั้นจะทำการเพิ่มพารามิเตอร์ที่ยังขาดไปเข้าไปในกรณีทดสอบร่วม ซึ่งจะใช้การเพิ่มค่าที่เป็นไปได้แต่ละตัวของพารามิเตอร์ที่ขาดไปเข้าไปในกรณีทดสอบร่วม เพื่อดูว่ากรณีทดสอบหลังจากเพิ่มค่าที่เป็นไปได้เข้าไปแล้วกรณีใดสามารถทำการทดสอบได้ครอบคลุมกรณีที่เหลืออยู่มากที่สุด จึงใช้ค่าที่เป็นไปได้ค่านั้น เพิ่มเข้าไปในกรณีทดสอบร่วม โดยค่าที่เป็นไปได้ที่เพิ่มเข้าไปในกรณีทดสอบนั้น จะต้องไม่ขัดแย้งกับเงื่อนไขบังคับ และทำซ้ำเช่นนี้ไปเรื่อยๆ จนกว่ากรณีทดสอบร่วม จะมีพารามิเตอร์ครบเป็นกรณีทดสอบที่สมบูรณ์

ในกรณีที่ได้ทำการเพิ่มค่าที่เป็นไปได้ของพารามิเตอร์ที่ขาดไปในทุกกรณีทดสอบร่วมจนเป็นกรณีทดสอบที่สมบูรณ์แล้ว แต่ยังไม่ครอบคลุมทุกกรณีที่ต้องการทดสอบ จะทำการสร้างกรณีทดสอบเพิ่มขึ้น เพื่อให้ครอบคลุมทุกกรณี โดยจะทำการพิจารณาคู่ลำดับที่ยังเหลืออยู่ในเซตที่ยังไม่ครอบคลุมทีละคู่ลำดับ ตั้งแต่คู่ลำดับคู่แรกซึ่งจะทำการสร้างกรณีทดสอบขึ้นโดยเติมสัญลักษณ์ “-” เข้าไปแทนที่ค่าที่เป็นไปได้ของพารามิเตอร์ที่ยังขาดไปในคู่ลำดับแต่ละคู่ที่เหลือ และคู่ลำดับคู่ถัดไปจะพิจารณาว่าสามารถรวมกันกับกรณีทดสอบที่ได้สร้างขึ้นก่อนหน้าหรือไม่ เนื่องจากต้องการสร้างกรณีทดสอบที่สามารถทำการทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบ จึงทำการสร้างกรณีทดสอบด้วยแต่ละกรณีที่ยังไม่ครอบคลุม จึงใช้เทคนิค

อัลกอริทึมเชิงละโมบ [4] มาช่วยในการสร้างกรณีทดสอบเพิ่มเติม กล่าวคือ คู่ลำดับหรือกรณีที่ยังไม่ครอบคลุมที่พบเป็นคู่แรกที่สามารถรวมกับกรณีทดสอบดังกล่าวได้โดยไม่ขัดแย้งกับค่าที่เป็นไปได้ของพารามิเตอร์ใดๆ จะนำคู่ลำดับดังกล่าวมารวมกับกรณีทดสอบก่อนหน้า และสร้างเป็นกรณีทดสอบใหม่ จากนั้นจะทำซ้ำไปเรื่อยจนได้กรณีทดสอบที่สามารถทดสอบได้ครอบคลุมทุกคู่ลำดับในเซตที่ยังไม่ครอบคลุม หลังจากพิจารณาครบทุกคู่ลำดับของกรณีที่ยังไม่ครอบคลุมแล้ว หากยังมีกรณีทดสอบใดที่มีสัญลักษณ์ “-” อยู่ ให้แทนที่สัญลักษณ์ “-” ด้วยค่าที่เป็นไปได้ใดๆ ของพารามิเตอร์ที่ยังขาดไป โดยค่าที่เป็นไปได้ที่แทนที่เข้าไป ต้องไม่ทำให้กรณีทดสอบนั้นเกิดความขัดแย้งกับเงื่อนไขบังคับ ในที่นี้ จะเสนอ อัลกอริทึมสำหรับสร้างกรณีทดสอบที่สมบูรณ์ ดังนี้

Algorithm: solution completion

Input: *incomplete test cases from common test case generation algorithm*

Output: *All test cases*

Begin

let πR be an empty set;

let πIR is uncover set of sub-problem;

let πR_i is uncover set of each parameter of R_i;

$\pi R = \pi IR - \pi R_i$;

for (each test case in *T*)

{ **for** (other parameters of test cases in *T*)

extend the test case in *T* by adding one value of other parameter. The test case is not covers in the constraints. The resulting test covers the most number of pairs in πR, and replacing the test case to *T*, and remove from πR pairs covered by the extended test case;

};

let *T''* be an empty set;

for (each pair in πR)

{ assume that the pairs contains value *w* of P_k,

$1 \leq k \leq i$, and value *u* of P_i;

if (*T''* contains a test with “-” as the value of P_k, $1 \leq k < i$, and value *u* of P_i)

modify test by replacing the “-” with *w*;

else

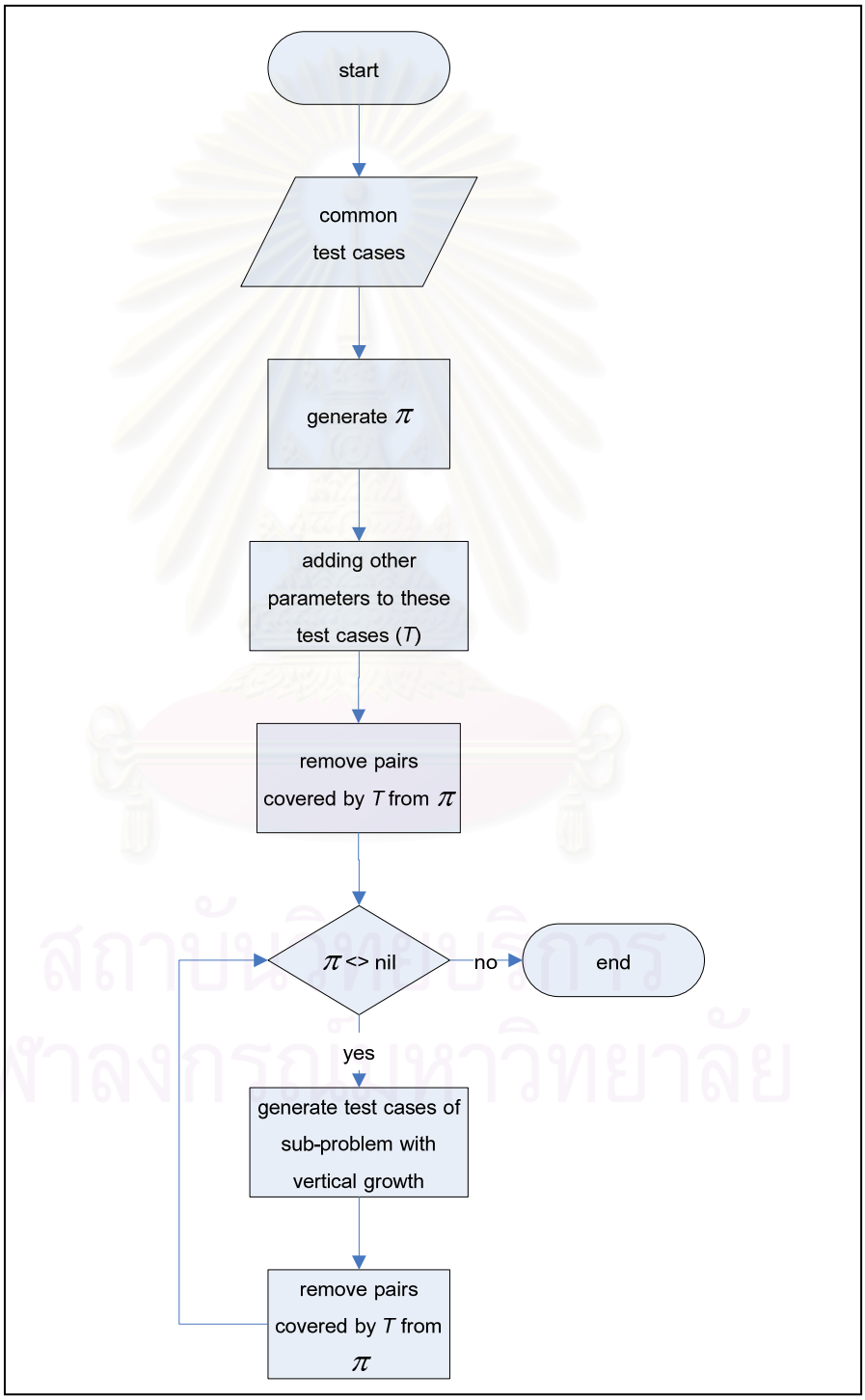
add a new test to *T''* that has *w* as the value of P_k, *u* as the value of P_i, and “-” as the value of every other parameter;

};

$T = T \cup T''$;

End

จากอัลกอริทึม สามารถเขียนเป็นผังงานขั้นตอนการทำให้คำตอบสมบูรณ์ ได้ดังรูปที่ 3.7



รูปที่ 3.7 ผังงานขั้นตอนการทำคำตอบให้สมบูรณ์

ตัวอย่างที่ 3.8

จากกรณีทดสอบที่ได้จากตัวอย่างที่ 3.7 จงทำให้เป็นกรณีทดสอบที่สมบูรณ์ โดยพารามิเตอร์ของทุกความสัมพันธ์เป็นตามตารางที่ 3.3

จากตารางที่ 3.3 พบว่า กรณีทดสอบที่สร้างขึ้นในตัวอย่างที่ 3.7 ยังเหลือคู่ลำดับที่ยังไม่ครอบคลุมอีก 26 คู่ที่ต้องทำการทดสอบ คือ (A_1,B_1), (A_1,B_2), (A_1,B_3), (A_2,B_1), (A_2,B_2), (A_2,B_3), (A_3,B_1), (A_3,B_2), (A_1,C_1), (A_1,C_2), (A_1,C_3), (A_2,C_1), (A_2,C_2), (A_2,C_3), (A_3,C_1), (A_3,C_2), (A_3,C_3), (A_1,D_1), (A_1,D_2), (A_1,D_3), (A_2,D_1), (A_2,D_2), (A_2,D_3), (A_3,D_1), (A_3,D_2) และ (A_3,D_3)

จะเห็นได้ว่า พารามิเตอร์ที่ยังขาดไปจากกรณีทดสอบทั้ง 9 กรณีทดสอบดังกล่าว คือ พารามิเตอร์ *A* จากนี้จะทำการเพิ่มค่าที่เป็นไปได้ของพารามิเตอร์ *A* เข้าไปในกรณีทดสอบแต่ละตัว เช่น กรณีทดสอบ (B_1,C_1,D_1) จะทำการเพิ่มค่าที่เป็นไปได้ของพารามิเตอร์ *A* คือ A_1 A_2 หรือ A_3 จะได้

(A_1,B_1,C_1,D_1) ซึ่งครอบคลุม 3 คู่ คือ (A_1,B_1), (A_1,C_1) และ (A_1,D_1)

(A_2,B_1,C_1,D_1) ซึ่งครอบคลุม 3 คู่ คือ (A_2,B_1), (A_2,C_1) และ (A_2,D_1)

(A_3,B_1,C_1,D_1) ซึ่งครอบคลุม 3 คู่ คือ (A_3,B_1), (A_3,C_1) และ (A_3,D_1)

เลือกกรณีทดสอบที่สามารถทดสอบได้ครอบคลุมมากที่สุด ในที่นี้เลือกกรณีทดสอบ (A_1,B_1,C_1,D_1) และทำซ้ำเช่นนี้จนครบทั้ง 9 กรณีทดสอบ จะได้ กรณีทดสอบ คือ (A_1,B_1,C_1,D_1), (A_3,B_1,C_2,D_2), (A_2,B_1,C_3,D_3), (A_3,B_2,C_1,D_3), (A_2,B_2,C_2,D_1), (A_1,B_2,C_3,D_2), (A_2,B_3,C_1,D_2), (A_1,B_3,C_2,D_3) และ (A_1,B_3,C_3,D_1)

จากทั้ง 9 กรณีทดสอบที่มีครบทุกพารามิเตอร์ที่ต้องการนำมาทดสอบแล้ว พบว่ายังไม่ครอบคลุม กรณี (A_3,C_3) และ (A_3,D_1) ซึ่งจะทำการสร้างกรณีทดสอบเพื่อให้ครอบคลุมคู่ลำดับอีกสองคู่ที่เหลืออยู่ โดยจะเพิ่ม “-” เข้าไปแทนที่ค่าที่เป็นไปได้ของพารามิเตอร์ที่ยังขาดไปในแต่ละคู่ลำดับที่เหลือ พร้อมพิจารณาว่า หากคู่ลำดับใดก็ตามสามารถนำมารวมกันได้ จะนำมารวมกันเป็นกรณีทดสอบเดียวกัน คือ

(A_3,C_3) จะได้ (A_3,-,C_3,-)

(A_3,D_1) จะได้ (A_3,-,-,D_1)

ทั้งสองกรณี สามารถนำมารวมกันได้ จะได้เป็น (A_3,-,C_3,D_1) และพบว่า กรณีทดสอบที่ได้นั้น ครอบคลุมคู่ลำดับทั้ง 2 คู่ที่เหลืออยู่แล้ว แต่กรณีทดสอบที่ได้นี้ยังขาดค่าที่เป็นไปได้ของพารามิเตอร์ *B* อยู่ ดังนั้น เพื่อให้เป็นกรณีทดสอบที่สมบูรณ์ จะทำการเพิ่มค่าที่เป็นไปได้ใดๆ ของพารามิเตอร์ *B* ซึ่งไม่ขัดแย้งกับเงื่อนไขบังคับเข้าไป

จากกรณีทดสอบ (A_3,-,C_3,D_1) พบว่าสามารถนำค่าที่เป็นไปได้ B_1 และ B_2 ของพารามิเตอร์ B แทนที่ได้ แต่ไม่สามารถนำค่าที่เป็นไปได้ B_3 เนื่องจากขัดแย้งกับเงื่อนไขบังคับ (A_3,B_3) ดังนั้น จะได้กรณีทดสอบทั้งหมด 10 กรณีทดสอบ ดังตารางที่ 3.6

ตารางที่ 3.6 กรณีทดสอบสำหรับความสัมพันธ์ที่ 1 และ 2

A	***B***	***C***	***D***
A_1	B_1	C_1	D_1
A_3	B_1	C_2	D_2
A_2	B_1	C_3	D_3
A_3	B_2	C_1	D_3
A_2	B_2	C_2	D_1
A_1	B_2	C_3	D_2
A_2	B_3	C_1	D_2
A_1	B_3	C_2	D_3
A_1	B_3	C_3	D_1
A_3	B_1	C_3	D_1

□

จากตัวอย่างที่ 3.8 จะสรุปขั้นตอนในการทำให้กรณีทดสอบของปัญหาย่อยที่สร้างขึ้นในขั้นตอนที่ 2 เป็นกรณีทดสอบที่สมบูรณ์ แสดงดังรูปที่ 3.8

A	***B***	***C***	***D***
A_1	B_1	C_1	D_1
A_3	B_1	C_2	D_2
A_2	B_1	C_3	D_3
A_3	B_2	C_1	D_3
A_2	B_2	C_2	D_1
A_1	B_2	C_3	D_2
A_2	B_3	C_1	D_2
A_1	B_3	C_2	D_3
A_1	B_3	C_3	D_1

⇒

A	***B***	***C***	***D***
A_1	B_1	C_1	D_1
A_3	B_1	C_2	D_2
A_2	B_1	C_3	D_3
A_3	B_2	C_1	D_3
A_2	B_2	C_2	D_1
A_1	B_2	C_3	D_2
A_2	B_3	C_1	D_2
A_1	B_3	C_2	D_3
A_1	B_3	C_3	D_1
A_3	B_1	C_3	D_1

รูปที่ 3.8 ขั้นตอนการทำให้เป็นกรณีทดสอบที่สมบูรณ์

3.3 สรุปอัลกอริทึมสำหรับสร้างกรณีทดสอบ

อัลกอริทึมที่จะนำเสนอต่อไป เป็นอัลกอริทึมที่ใช้สร้างกรณีทดสอบเมื่อที่พารามิเตอร์ในระบบที่ต้องการนำมาสร้างกรณีทดสอบมีความสัมพันธ์กันแบบแพร์ไวส์ หรือแบบเอ็นเวย์

กำหนดให้ R_i มีพารามิเตอร์ $P_1,P_2,\ldots,P_m$ ซึ่ง P_i มีค่าที่เป็นไปได้ $v_{i,1}, v_{i,2}, \ldots, v_{i,ni}$

Algorithm: n-way test case generation

Input: *All relations defined on parameters*

Output: *All test cases*

Begin

*// **Constraints identification***

generates π;

generates πR_i;

construct the constraints;

*// **Common test cases generation***

construct *MIR*, and generates πIR;

// Horizontal growth

let *T* be an empty set;

generate all combination of values of the first two parameters of *MIR*, and adding pairs to *T*;

if ($Card(T) \leq ni$)

{ **for** $1 < j \leq Card(T)$

extend the *j*th test in *T* by adding value v_j and remove from πIR pairs covered by the extended test;

}

else

{ **for** $1 < j \leq ni$

extend the *j*th test in *T* by adding value v_j and remove from πIR pairs covered by the extended test;

for $ni < j \leq Card(T)$

extend the *j*th test in *T* by adding one value of P_i such that the resulting test covers the most number of pairs in πIR, and remove from πIR pairs covered by the extended test;

}

// Vertical growth

let *T'* be an empty set;

for (each pair in πIR)

{ assume that the pairs contains value *w* of P_k, $1 \leq k <$ i, and value *u* of P_i;

if (*T'* contains a test with "-" as the value of P_k and *u* as the value of P_i)

modify this test by replacing the "-" with *w*;

else

add a new test to *T'* that has *w* as the value of P_k, *u* as the value of P_i, and "-" as the value of every other parameter;

}

$T = T \cup T'$;

// Solution completion

let πR be an empty set;

remove πIR pairs from πR_i, and add all pairs in πR_i to πR;

for (each test case in *T*)

{ **for** (other parameters of test cases in *T*)

extend the test case in *T* by adding one value of other parameter. The test case is not covers in the constraints. The resulting test covers the most number of pairs in πR, and replacing the test case to *T*, and remove from πR pairs covered by the extended test case;

};

let *T''* be an empty set;

for (each pair in πR)

{ assume that the pairs contains value *w* of P_k, $1 \leq k < i$, and value *u* of P_i;

if (*T''* contains a test with "-" as the value of P_k, and *u* as the value of P_i)

modify test by replacing the "-" with *w*;

else

add a new test to *T''* that has *w* as the value of P_k, *u* as the value of P_i, and "-" as the value of every other parameter;

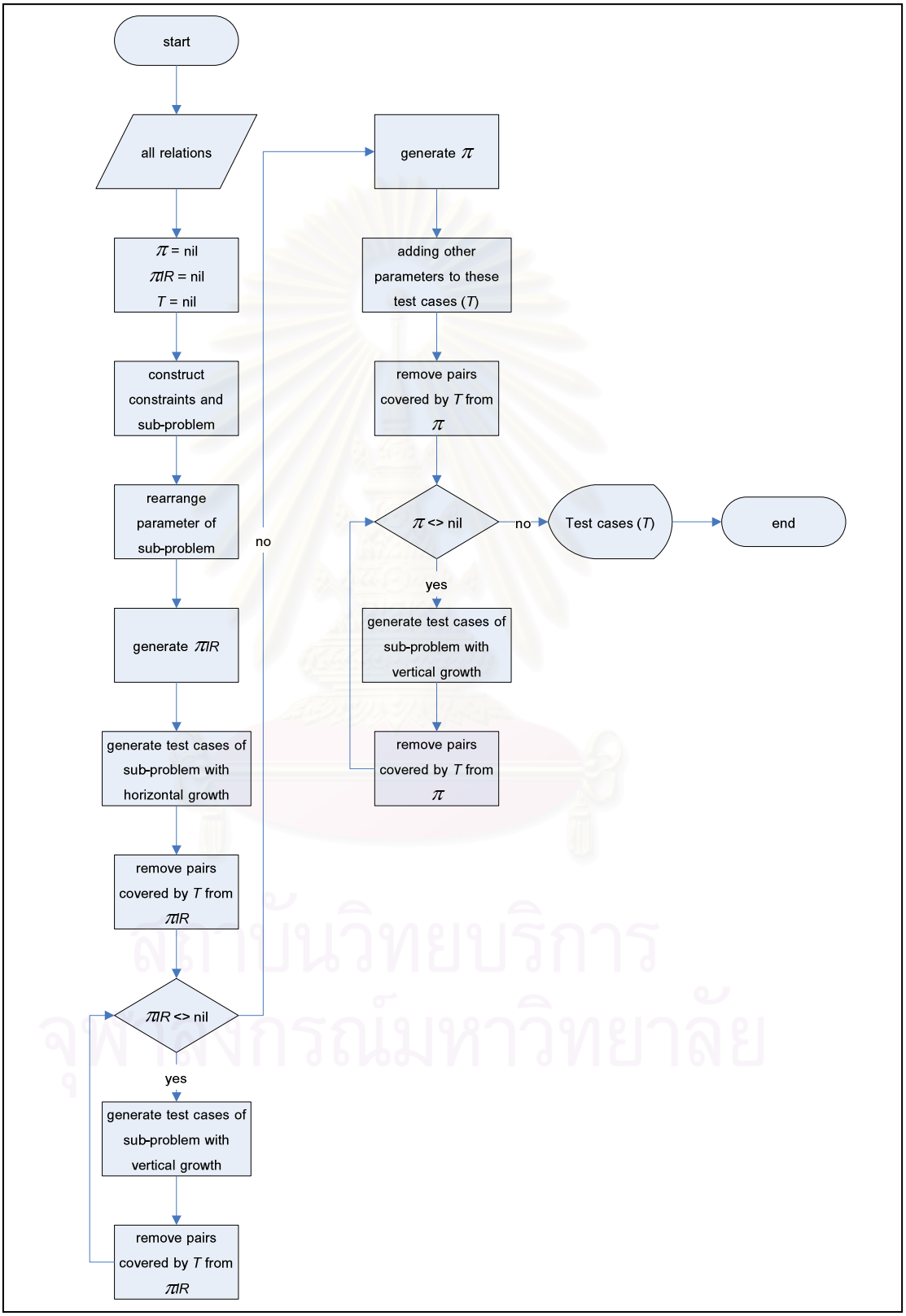
};

$T = T \cup T''$;

End

3.4 ผังงานขั้นตอนในการสร้างกรณีทดสอบ

ผังงาน (Flowchart) ดังรูปที่ 3.9 เป็นผังงานขั้นตอนที่ใช้ในการสร้างกรณีทดสอบ



รูปที่ 3.9 ผังงานขั้นตอนที่ใช้ในการสร้างกรณีทดสอบ

บทที่ 4

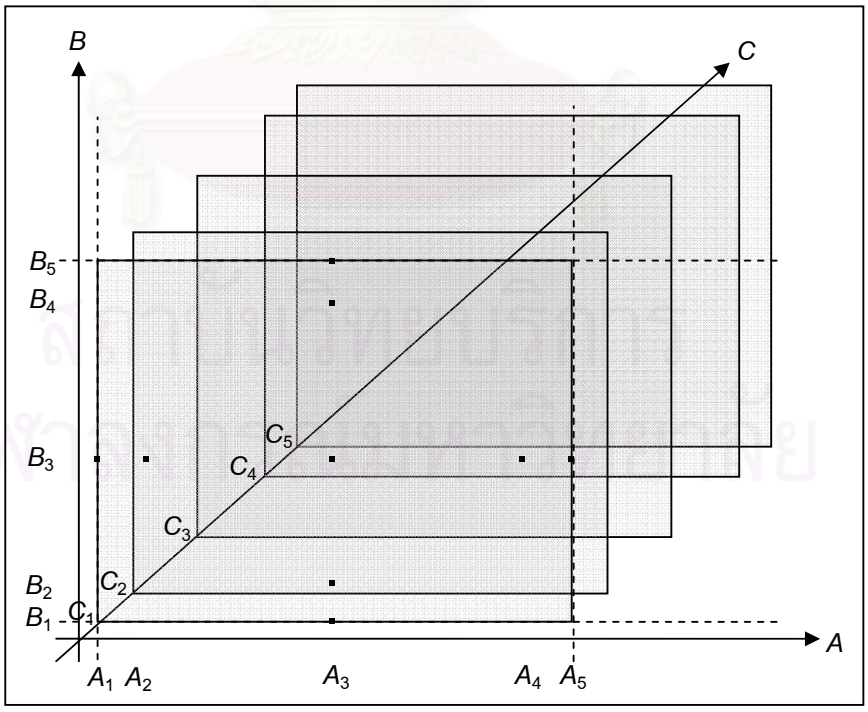
การเปรียบเทียบผลการทดลอง

4.1 การเปรียบเทียบกรณีที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์

เนื่องจากงานวิจัยนี้ ได้ทำการปรับปรุงอัลกอริทึมจากอัลกอริทึมไอพีโอ ซึ่งสามารถสร้างกรณีทดสอบได้เฉพาะในกรณีพารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์เท่านั้น ดังนั้น กรณีพารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ จะทำการเปรียบเทียบจำนวนกรณีทดสอบที่ได้จากอัลกอริทึมที่ทำการพัฒนาขึ้น กับวิธีการวิเคราะห์ค่าขอบเขต วิธีการทดสอบสภาพทนทาน วิธีการทดสอบกรณีเลวร้ายที่สุด และจะเปรียบเทียบผลที่ได้จากอัลกอริทึมที่นำเสนอกับผลที่ได้จากอัลกอริทึมไอพีโอในสองประเด็น ได้แก่ จำนวนกรณีทดสอบที่สร้างขึ้น และจำนวนขั้นตอนที่ใช้ในการสร้างกรณีทดสอบ

4.1.1 เปรียบเทียบจำนวนกรณีทดสอบกับวิธีการวิเคราะห์ค่าขอบเขต

การเปรียบเทียบจำนวนกรณีทดสอบระหว่างวิธีการวิเคราะห์ค่าขอบเขตกับอัลกอริทึมที่นำเสนอนั้น ในที่นี้จะทำการทดสอบในกรณีที่ระบบมีพารามิเตอร์ 3 พารามิเตอร์ คือ พารามิเตอร์ *A B* และ *C* ซึ่งแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 5 ค่า



รูปที่ 4.1 พารามิเตอร์และค่าที่เป็นไปได้สำหรับเปรียบเทียบกับวิธีการวิเคราะห์ค่าขอบเขต

ภาพที่ 4.1 แสดงจุดที่เลือกมาเพื่อสร้างเป็นกรณีทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต แต่ละแกนแทนพารามิเตอร์ ซึ่งมี 5 ค่าที่เป็นไปได้ จะได้กรณีทดสอบทั้งหมด 45 กรณีทดสอบ ดังนี้

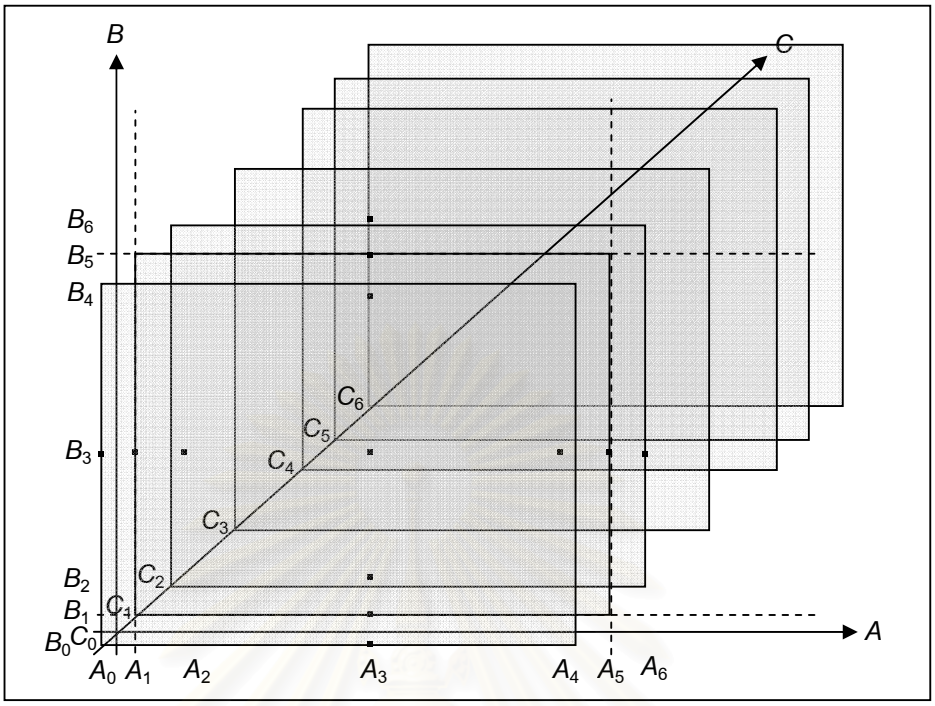
{ <A_3,B_1,C_1>, <A_3,B_2,C_1>, <A_3,B_3,C_1>, <A_3,B_4,C_1>, <A_3,B_5,C_1>, <A_1,B_3,C_1>, <A_2,B_3,C_1>, <A_4,B_3,C_1>, <A_5,B_3,C_1>, <A_3,B_1,C_2>, <A_3,B_2,C_2>, <A_3,B_3,C_2>, <A_3,B_4,C_2>, <A_3,B_5,C_2>, <A_1,B_3,C_2>, <A_2,B_3,C_2>, <A_4,B_3,C_2>, <A_5,B_3,C_2>, <A_3,B_1,C_3>, <A_3,B_2,C_3>, <A_3,B_3,C_3>, <A_3,B_4,C_3>, <A_3,B_5,C_3>, <A_1,B_3,C_3>, <A_2,B_3,C_3>, <A_4,B_3,C_3>, <A_5,B_3,C_3>, <A_3,B_1,C_4>, <A_3,B_2,C_4>, <A_3,B_3,C_4>, <A_3,B_4,C_4>, <A_3,B_5,C_4>, <A_1,B_3,C_4>, <A_2,B_3,C_4>, <A_4,B_3,C_4>, <A_5,B_3,C_4>, <A_3,B_1,C_5>, <A_3,B_2,C_5>, <A_3,B_3,C_5>, <A_3,B_4,C_5>, <A_3,B_5,C_5>, <A_1,B_3,C_5>, <A_2,B_3,C_5>, <A_4,B_3,C_5>, <A_5,B_3,C_5> }

และจำนวนกรณีทดสอบที่สร้างจากอัลกอริทึมที่นำเสนอ มีจำนวน 26 กรณีทดสอบ ดังนี้

{ (A_1,B_1,C_1), (A_1,B_2,C_2), (A_1,B_3,C_3), (A_1,B_4,C_4), (A_1,B_5,C_5), (A_2,B_1,C_5), (A_2,B_2,C_4), (A_2,B_3,C_2), (A_2,B_4,C_3), (A_2,B_5,C_1), (A_3,B_1,C_4), (A_3,B_2,C_5), (A_3,B_3,C_1), (A_3,B_4,C_2), (A_3,B_5,C_3), (A_4,B_1,C_3), (A_4,B_2,C_1), (A_4,B_3,C_5), (A_4,B_4,C_5), (A_4,B_5,C_4), (A_5,B_1,C_2), (A_5,B_2,C_3), (A_5,B_3,C_4), (A_5,B_4,C_1), (A_5,B_5,C_5), (A_4,B_5,C_2) }

4.1.2 เปรียบเทียบจำนวนกรณีทดสอบกับวิธีการทดสอบสภาพทนทาน

การเปรียบเทียบจำนวนกรณีทดสอบระหว่างวิธีการทดสอบสภาพทนทานกับอัลกอริทึมที่นำเสนอนั้น ในที่นี้จะทำการทดสอบในกรณีที่ระบบมีพารามิเตอร์ 3 พารามิเตอร์ คือ พารามิเตอร์ A B และ C เช่นเดียวกันกับวิธีการวิเคราะห์ค่าขอบเขต แต่ทุกพารามิเตอร์มีค่าที่เป็นไปได้ 7 ค่า



รูปที่ 4.2 พารามิเตอร์และค่าที่เป็นไปได้สำหรับเปรียบเทียบกับวิธีการทดสอบสภาพทนทาน

ภาพที่ 4.2 แสดงจุดที่เลือกมาเพื่อสร้างเป็นกรณีทดสอบด้วยวิธีการทดสอบสภาพทนทาน แต่ละแกนแทนพารามิเตอร์ ซึ่งมี 5 ค่าที่เป็นไปได้ จะได้กรณีทดสอบทั้งหมด 91 กรณีทดสอบ ดังนี้

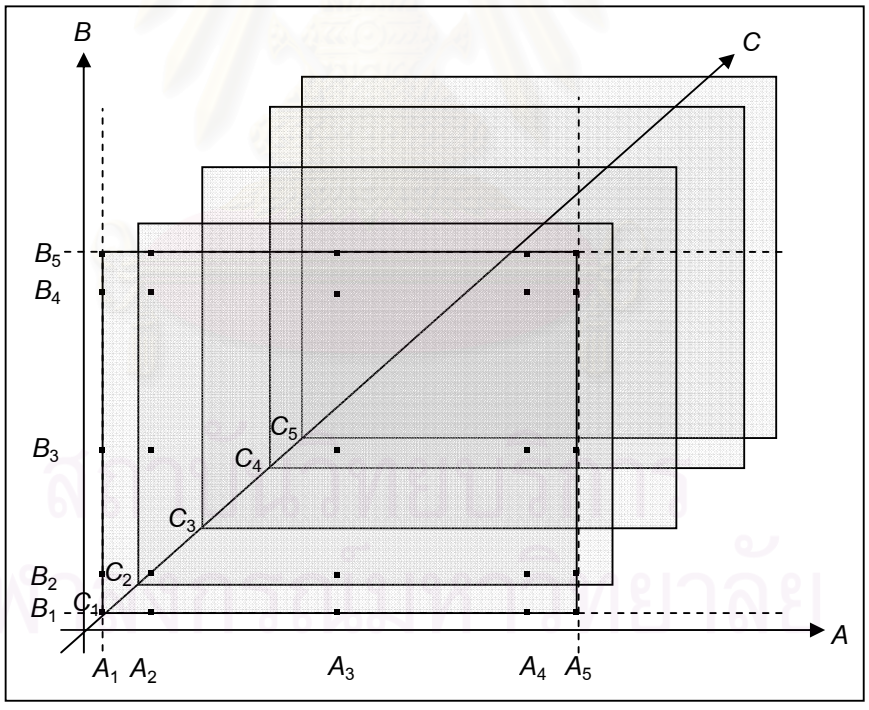
{ <A_3,B_0,C_0>, <A_3,B_1,C_0>, <A_3,B_2,C_0>, <A_3,B_3,C_0>, <A_3,B_4,C_0>, <A_3,B_5,C_0>, <A_3,B_6,C_0>, <A_0,B_3,C_0>, <A_1,B_3,C_0>, <A_2,B_3,C_0>, <A_4,B_3,C_0>, <A_5,B_3,C_0>, <A_6,B_3,C_0>, <A_3,B_0,C_1>, <A_3,B_1,C_1>, <A_3,B_2,C_1>, <A_3,B_3,C_1>, <A_3,B_4,C_1>, <A_3,B_5,C_1>, <A_3,B_6,C_1>, <A_0,B_3,C_1>, <A_1,B_3,C_1>, <A_2,B_3,C_1>, <A_4,B_3,C_1>, <A_5,B_3,C_1>, <A_6,B_3,C_1>, <A_3,B_0,C_2>, <A_3,B_1,C_2>, <A_3,B_2,C_2>, <A_3,B_3,C_2>, <A_3,B_4,C_2>, <A_3,B_5,C_2>, <A_3,B_6,C_2>, <A_0,B_3,C_2>, <A_1,B_3,C_2>, <A_2,B_3,C_2>, <A_4,B_3,C_2>, <A_5,B_3,C_2>, <A_6,B_3,C_2>, <A_3,B_0,C_3>, <A_3,B_1,C_3>, <A_3,B_2,C_3>, <A_3,B_3,C_3>, <A_3,B_4,C_3>, <A_3,B_5,C_3>, <A_3,B_6,C_3>, <A_0,B_3,C_3>, <A_1,B_3,C_3>, <A_2,B_3,C_3>, <A_4,B_3,C_3>, <A_5,B_3,C_3>, <A_6,B_3,C_3>,<A_3,B_0,C_4>, <A_3,B_1,C_4>, <A_3,B_2,C_4>, <A_3,B_3,C_4>, <A_3,B_4,C_4>, <A_3,B_5,C_4>, <A_3,B_6,C_4>, <A_0,B_3,C_4>, <A_1,B_3,C_4>, <A_2,B_3,C_4>, <A_4,B_3,C_4>, <A_5,B_3,C_4>, <A_6,B_3,C_4>, <A_3,B_0,C_5>, <A_3,B_1,C_5>, <A_3,B_2,C_5>, <A_3,B_3,C_5>, <A_3,B_4,C_5>, <A_3,B_5,C_5>, <A_3,B_6,C_5>, <A_0,B_3,C_5>, <A_1,B_3,C_5>, <A_2,B_3,C_5>, <A_4,B_3,C_5>, <A_5,B_3,C_5>, <A_6,B_3,C_5>, <A_3,B_0,C_6>, <A_3,B_1,C_6>, <A_3,B_2,C_6>, <A_3,B_3,C_6>, <A_3,B_4,C_6>, <A_3,B_5,C_6>, <A_3,B_6,C_6>, <A_0,B_3,C_6>, <A_1,B_3,C_6>, <A_2,B_3,C_6>, <A_4,B_3,C_6>, <A_5,B_3,C_6>, <A_6,B_3,C_6> }

และจำนวนกรณีทดสอบที่สร้างจากอัลกอริทึมที่นำเสนอ มีจำนวน 52 กรณีทดสอบ ดังนี้

{ (A_1,B_1,C_1), (A_1,B_2,C_2), (A_1,B_3,C_3), (A_1,B_4,C_4), (A_1,B_5,C_5), (A_1,B_6,C_6), (A_1,B_7,C_7), (A_2,B_1,C_7), (A_2,B_2,C_6), (A_2,B_3,C_5), (A_2,B_4,C_3), (A_2,B_5,C_4), (A_2,B_6,C_2), (A_2,B_7,C_1), (A_3,B_1,C_6), (A_3,B_2,C_7), (A_3,B_3,C_4), (A_3,B_4,C_5), (A_3,B_5,C_3), (A_3,B_6,C_1), (A_3,B_7,C_2), (A_4,B_1,C_5), (A_4,B_2,C_4), (A_4,B_3,C_7), (A_4,B_4,C_6), (A_4,B_5,C_2), (A_4,B_6,C_3), (A_4,B_7,C_6), (A_5,B_1,C_4), (A_5,B_2,C_5), (A_5,B_3,C_6), (A_5,B_4,C_7), (A_5,B_5,C_1), (A_5,B_6,C_7), (A_5,B_7,C_3), (A_6,B_1,C_3), (A_6,B_2,C_1), (A_6,B_3,C_2), (A_6,B_4,C_7), (A_6,B_5,C_6), (A_6,B_6,C_5), (A_6,B_7,C_4), (A_7,B_1,C_2), (A_7,B_2,C_3), (A_7,B_3,C_1), (A_7,B_4,C_7), (A_7,B_5,C_7), (A_7,B_6,C_4), (A_7,B_7,C_5), (A_4,B_4,C_1), (A_5,B_4,C_2), (A_7,B_1,C_6) }

4.1.3 เปรียบเทียบจำนวนกรณีทดสอบกับวิธีทดสอบกรณีเลวร้ายที่สุด

การเปรียบเทียบจำนวนกรณีทดสอบระหว่างวิธีทดสอบกรณีเลวร้ายที่สุดกับอัลกอริทึมที่ได้นำเสนอนั้น ในที่นี้จะทำการทดสอบในกรณีที่ระบบมีพารามิเตอร์ 3 พารามิเตอร์ คือ พารามิเตอร์ *A* *B* และ *C* ซึ่งแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 5 ค่า เช่นเดียวกันกับวิธีการวิเคราะห์ค่าขอบเขต แต่วิธีการนี้จะเพิ่มจุดที่เลือกมาเพื่อสร้างเป็นกรณีทดสอบ



รูปที่ 4.3 พารามิเตอร์และค่าที่เป็นไปได้สำหรับเปรียบเทียบกับวิธีทดสอบกรณีเลวร้ายที่สุด

ภาพที่ 4.3 แสดงจุดที่เลือกมาเพื่อสร้างเป็นกรณีทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต แต่ละแกนแทนพารามิเตอร์ ซึ่งมี 5 ค่าที่เป็นไปได้ จะได้กรณีทดสอบทั้งหมด 125 กรณีทดสอบ ดังนี้

{ $<A_3,B_1,C_1>$, $<A_3,B_2,C_1>$, $<A_3,B_3,C_1>$, $<A_3,B_4,C_1>$, $<A_3,B_5,C_1>$, $<A_1,B_3,C_1>$, $<A_2,B_3,C_1>$, $<A_4,B_3,C_1>$, $<A_5,B_3,C_1>$, $<A_1,B_1,C_1>$, $<A_1,B_2,C_1>$, $<A_1,B_4,C_1>$, $<A_1,B_5,C_1>$, $<A_2,B_1,C_1>$, $<A_2,B_2,C_1>$, $<A_2,B_4,C_1>$, $<A_2,B_5,C_1>$, $<A_4,B_1,C_1>$, $<A_4,B_2,C_1>$, $<A_4,B_4,C_1>$, $<A_4,B_5,C_1>$, $<A_5,B_1,C_1>$, $<A_5,B_2,C_1>$, $<A_5,B_4,C_1>$, $<A_5,B_5,C_1>$, $<A_3,B_1,C_2>$, $<A_3,B_2,C_2>$, $<A_3,B_3,C_2>$, $<A_3,B_4,C_2>$, $<A_3,B_5,C_2>$, $<A_1,B_3,C_2>$, $<A_2,B_3,C_2>$, $<A_4,B_3,C_2>$, $<A_5,B_3,C_2>$, $<A_1,B_1,C_2>$, $<A_1,B_2,C_2>$, $<A_1,B_4,C_2>$, $<A_1,B_5,C_2>$, $<A_2,B_1,C_2>$, $<A_2,B_2,C_2>$, $<A_2,B_4,C_2>$, $<A_2,B_5,C_2>$, $<A_4,B_1,C_2>$, $<A_4,B_2,C_2>$, $<A_4,B_4,C_2>$, $<A_4,B_5,C_2>$, $<A_5,B_1,C_2>$, $<A_5,B_2,C_2>$, $<A_5,B_4,C_2>$, $<A_5,B_5,C_2>$, $<A_3,B_1,C_3>$, $<A_3,B_2,C_3>$, $<A_3,B_3,C_3>$, $<A_3,B_4,C_3>$, $<A_3,B_5,C_3>$, $<A_1,B_3,C_3>$, $<A_2,B_3,C_3>$, $<A_4,B_3,C_3>$, $<A_5,B_3,C_3>$, $<A_1,B_1,C_3>$, $<A_1,B_2,C_3>$, $<A_1,B_4,C_3>$, $<A_1,B_5,C_3>$, $<A_2,B_1,C_3>$, $<A_2,B_2,C_3>$, $<A_2,B_4,C_3>$, $<A_2,B_5,C_3>$, $<A_4,B_1,C_3>$, $<A_4,B_2,C_3>$, $<A_4,B_4,C_3>$, $<A_4,B_5,C_3>$, $<A_5,B_1,C_3>$, $<A_5,B_2,C_3>$, $<A_5,B_4,C_3>$, $<A_5,B_5,C_3>$, $<A_3,B_1,C_4>$, $<A_3,B_2,C_4>$, $<A_3,B_3,C_4>$, $<A_3,B_4,C_4>$, $<A_3,B_5,C_4>$, $<A_1,B_3,C_4>$, $<A_2,B_3,C_4>$, $<A_4,B_3,C_4>$, $<A_5,B_3,C_4>$, $<A_1,B_1,C_4>$, $<A_1,B_2,C_4>$, $<A_1,B_4,C_4>$, $<A_1,B_5,C_4>$, $<A_2,B_1,C_4>$, $<A_2,B_2,C_4>$, $<A_2,B_4,C_4>$, $<A_2,B_5,C_4>$, $<A_4,B_1,C_4>$, $<A_4,B_2,C_4>$, $<A_4,B_4,C_4>$, $<A_4,B_5,C_4>$, $<A_5,B_1,C_4>$, $<A_5,B_2,C_4>$, $<A_5,B_4,C_4>$, $<A_5,B_5,C_4>$, $<A_3,B_1,C_5>$, $<A_3,B_2,C_5>$, $<A_3,B_3,C_5>$, $<A_3,B_4,C_5>$, $<A_3,B_5,C_5>$, $<A_1,B_3,C_5>$, $<A_2,B_3,C_5>$, $<A_4,B_3,C_5>$, $<A_5,B_3,C_5>$, $<A_1,B_1,C_5>$, $<A_1,B_2,C_5>$, $<A_1,B_4,C_5>$, $<A_1,B_5,C_5>$, $<A_2,B_1,C_5>$, $<A_2,B_2,C_5>$, $<A_2,B_4,C_5>$, $<A_2,B_5,C_5>$, $<A_4,B_1,C_5>$, $<A_4,B_2,C_5>$, $<A_4,B_4,C_5>$, $<A_4,B_5,C_5>$, $<A_5,B_1,C_5>$, $<A_5,B_2,C_5>$, $<A_5,B_4,C_5>$, $<A_5,B_5,C_5>$ }

และจำนวนกรณีทดสอบที่สร้างจากอัลกอริทึมที่นำเสนอ มีจำนวน 26 กรณีทดสอบ ดังนี้

{ (A_1,B_1,C_1), (A_1,B_2,C_2), (A_1,B_3,C_3), (A_1,B_4,C_4), (A_1,B_5,C_5), (A_2,B_1,C_5), (A_2,B_2,C_4), (A_2,B_3,C_2), (A_2,B_4,C_3), (A_2,B_5,C_1), (A_3,B_1,C_4), (A_3,B_2,C_5), (A_3,B_3,C_1), (A_3,B_4,C_2), (A_3,B_5,C_3), (A_4,B_1,C_3), (A_4,B_2,C_1), (A_4,B_3,C_5), (A_4,B_4,C_5), (A_4,B_5,C_4), (A_5,B_1,C_2), (A_5,B_2,C_3), (A_5,B_3,C_4), (A_5,B_4,C_1), (A_5,B_5,C_5), (A_4,B_5,C_2) }

จากการสร้างกรณีทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต และวิธีทดสอบกรณีเลวร้ายที่สุด เมื่อมีพารามิเตอร์ 3 พารามิเตอร์ และมีค่าที่เป็นไปได้ของแต่ละพารามิเตอร์ 5 ค่าที่เป็นไปได้ แต่วิธีการวิเคราะห์ค่าขอบเขตมีกรณีทดสอบทั้งหมด 45 กรณีทดสอบ และวิธีทดสอบกรณีเลวร้ายที่สุดมีกรณีทดสอบทั้งหมด 125 กรณีทดสอบ สาเหตุที่จำนวนกรณีทดสอบที่สร้างขึ้นจากทั้งสองวิธีมีจำนวนไม่เท่ากัน เพราะว่าทั้งสองวิธีเลือกจุดที่สนใจนำมาสร้างกรณีทดสอบแตกต่างกัน แต่เมื่อนำมาสร้างกรณีทดสอบด้วยอัลกอริทึมที่นำเสนอนี้ จะได้จำนวนกรณี

ทดสอบ 26 กรณีทดสอบเท่ากัน เพราะอัลกอริทึมที่นำเสนอให้ความสนใจที่พารามิเตอร์ และค่าที่เป็นไปได้ของพารามิเตอร์เท่านั้น

4.1.4 เปรียบเทียบกับอัลกอริทึมไอพีโอ

1) ผลการเปรียบเทียบด้านจำนวนกรณีทดสอบที่สร้างขึ้น

ตารางที่ 4.1 เปรียบเทียบจำนวนกรณีทดสอบที่สร้างขึ้น

ระบบ	กรณีศึกษาที่ 1	กรณีศึกษาที่ 2	กรณีศึกษาที่ 3	กรณีศึกษาที่ 4	กรณีศึกษาที่ 5
อัลกอริทึมไอพีโอ	10	9	23	52	59
อัลกอริทึมของโซลเหมิน ไมตี้	-	9	15	24	24
อัลกอริทึมที่นำเสนอ	9	9	23	50	55

โดยที่ กรณีศึกษาที่ 1: 4 พารามิเตอร์ (2 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 2 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่ 2: 4 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า

กรณีศึกษาที่ 3: 13 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า

กรณีศึกษาที่ 4: 61 พารามิเตอร์ (15 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 17 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 29 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่ 5: 75 พารามิเตอร์ (1 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 39 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 35 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่นำมาใช้ในการเปรียบเทียบนำมาจากงานวิจัยของหยูเล่ย [9, 10] ซึ่งกรณีศึกษานี้ได้นำไปใช้ในการเปรียบเทียบระหว่างจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมของโซลเหมิน ไมตี้ [6] และจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี [7] เมื่อพารามิเตอร์ในระบบมีความสัมพันธ์กันแบบแพร์ไวส์

จากผลการเปรียบเทียบระหว่างจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอกับอัลกอริทึมไอพีโอ เห็นได้ว่า มีหลายกรณีศึกษาที่จำนวนกรณีทดสอบซึ่งสร้างจากอัลกอริทึมที่นำเสนอมีจำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างจากอัลกอริทึมไอพีโอ

แสดงว่า การจัดอันดับของพารามิเตอร์ที่จะนำมาสร้างกรณีทดสอบ มีผลต่อจำนวนกรณีทดสอบที่สร้างขึ้นด้วย

เมื่อเปรียบเทียบจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอกับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมของโซลเหมิน ไมตี้จะเห็นได้ว่าส่วนมากจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมของโซลเหมิน ไมตี้ มีจำนวนน้อยกว่า แต่เนื่องจากอัลกอริทึมของโซลเหมือน ไมตี้ แบ่งกลุ่มของค่าที่เป็นไปได้ออกเป็น 3 กลุ่ม คือ กลุ่มที่แต่ละพารามิเตอร์มีค่าที่เป็นไปได้จำนวน 2 ค่าที่เป็นไปได้เท่ากันทุกพารามิเตอร์ กลุ่มถัดมาคือ กลุ่มที่แต่ละพารามิเตอร์มีจำนวนค่าที่เป็นไปได้มากกว่า 2 ค่าที่เป็นไปได้เท่ากันทุกพารามิเตอร์ และกลุ่มสุดท้าย คือ กลุ่มที่ค่าที่เป็นไปได้ของแต่ละพารามิเตอร์มีจำนวนไม่เท่ากัน หลังจากแบ่งกลุ่มแล้ว แต่ละกลุ่มจะใช้อัลกอริทึมในการสร้างกรณีทดสอบแตกต่างกัน [6] ซึ่งอัลกอริทึมที่โซลเหมิน ไมตี้ใช้ในการสร้างกรณีทดสอบ ไม่เป็นรูปแบบทั่วไป (generic form) จึงเลือกอัลกอริทึมไอพีโอที่เป็นรูปแบบทั่วไปมากกว่า มาทำการปรับปรุงเพื่อให้สามารถสร้างกรณีทดสอบแบบเอ็นเวย์ได้

2) ผลการเปรียบเทียบด้านขั้นตอนที่ใช้ในการสร้างกรณีทดสอบ

ตารางที่ 4.2 เปรียบเทียบขั้นตอนที่ใช้ในการสร้างกรณีทดสอบ

ระบบ	อัลกอริทึมไอพีโอ		อัลกอริทึมที่นำเสนอ	
	แนวนอน	แนวตั้ง	แนวนอน	แนวตั้ง
กรณีศึกษาที่ 1	✓	✓	✓	✗
กรณีศึกษาที่ 2	✓	✓	✓	✓
กรณีศึกษาที่ 3	✓	✓	✓	✓
กรณีศึกษาที่ 4	✓	✓	✓	✓
กรณีศึกษาที่ 5	✓	✓	✓	✓

โดยที่ กรณีศึกษาที่ 1: 4 พารามิเตอร์ (2 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 2 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่ 2: 4 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า

กรณีศึกษาที่ 3: 13 พารามิเตอร์ โดยแต่ละพารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า

กรณีศึกษาที่ 4: 61 พารามิเตอร์ (15 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 17 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 29 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่ 5: 75 พารามิเตอร์ (1 พารามิเตอร์มีค่าที่เป็นไปได้ 4 ค่า 39 พารามิเตอร์มีค่าที่เป็นไปได้ 3 ค่า 35 พารามิเตอร์มีค่าที่เป็นไปได้ 2 ค่า)

กรณีศึกษาที่นำมาใช้ในการเปรียบเทียบนำมาจากงานวิจัยของหยูเล่ย [9, 10] โดยในการทดลองนี้ ได้เพิ่มกรณีศึกษาที่ 1 เข้าไป เพื่อให้เห็นความแตกต่างของขั้นตอนที่ใช้ในการสร้างกรณีทดสอบของอัลกอริทึมไอพีโอ และอัลกอริทึมที่นำเสนอ เมื่อพารามิเตอร์ในระบบมีความสัมพันธ์กันแบบแพร์ไวส์

จากการเปรียบเทียบขั้นตอนที่ใช้ในการสร้างกรณีทดสอบในกรณีที่พารามิเตอร์ในระบบมีความสัมพันธ์กันแบบแพร์ไวส์ พบว่าในกรณีที่พารามิเตอร์ที่นำมาสร้างกรณีทดสอบมีจำนวนน้อยๆ และค่าที่เป็นไปได้ของแต่ละพารามิเตอร์มีจำนวนแตกต่างกัน จะทำให้สามารถลดขั้นตอนในการสร้างกรณีทดสอบได้ แต่หากทุกพารามิเตอร์มีค่าที่เป็นได้จำนวนเท่ากันจะไม่ทำให้ขั้นตอนที่ใช้ในการสร้างกรณีทดสอบลดลงไปจากอัลกอริทึมเดิม และในกรณีที่พารามิเตอร์ที่นำมาสร้างกรณีทดสอบมีจำนวนมาก การจัดอันดับของพารามิเตอร์จะไม่ทำให้ขั้นตอนในการสร้างกรณีทดสอบแตกต่างกัน เพราะว่าในขั้นตอนการเติบโตแนวนอนเป็นการสร้างกรณีทดสอบขึ้นจากค่าที่เป็นไปได้ของสองพารามิเตอร์แรก ซึ่งถ้าพารามิเตอร์มีจำนวนมาก ก็ต้องใช้กรณีทดสอบจำนวนมากในการทดสอบ เพื่อให้ครอบคลุมทุกกรณี ทำให้จำนวนกรณีทดสอบที่สร้างขึ้นในขั้นตอนการเติบโตแนวนอนนั้น ไม่เพียงพอที่จะใช้ทดสอบให้ครอบคลุมทุกกรณีที่ต้องการ ทำให้ต้องใช้ขั้นตอนการเติบโตแนวตั้งในการสร้างกรณีทดสอบด้วย

4.2 การเปรียบเทียบกรณีที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์

ในงานวิจัยนี้ จะเปรียบเทียบจำนวนกรณีทดสอบในกรณีที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ ระหว่างจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีการเออีทีจี และจำนวนกรณีทดสอบที่ได้จากอัลกอริทึมที่นำเสนอ

ตารางที่ 4.3 เปรียบเทียบจำนวนกรณีทดสอบที่สร้างกรณีสร้างขึ้น

ระบบ	เออีทีจี	อัลกอริทึมที่นำเสนอ
กรณีศึกษาที่ 1	8	8
กรณีศึกษาที่ 2	10	10
กรณีศึกษาที่ 3	8	9
กรณีศึกษาที่ 4	15	12
กรณีศึกษาที่ 5	10	9
กรณีศึกษาที่ 6	16	16
กรณีศึกษาที่ 7	9	9
กรณีศึกษาที่ 8	19	17

โดยที่ กรณีศึกษาที่ 1: 4 พารามิเตอร์ (2 พารามิเตอร์มี 3 ค่าที่เป็นไปได้ 2 พารามิเตอร์มี 2 ค่าที่เป็นไปได้)

กรณีศึกษาที่ 2: 4 พารามิเตอร์ โดยแต่ละพารามิเตอร์มี 3 ค่าที่เป็นไปได้

กรณีศึกษาที่ 3: 5 พารามิเตอร์ (2 พารามิเตอร์มี 3 ค่าที่เป็นไปได้ 3 พารามิเตอร์มี 2 ค่าที่เป็นไปได้)

กรณีศึกษาที่ 4: 5 พารามิเตอร์ โดยแต่ละพารามิเตอร์มี 3 ค่าที่เป็นไปได้

กรณีศึกษาที่ 5: 6 พารามิเตอร์ (2 พารามิเตอร์มี 3 ค่าที่เป็นไปได้ 4 พารามิเตอร์มี 2 ค่าที่เป็นไปได้)

กรณีศึกษาที่ 6: 6 พารามิเตอร์ โดยแต่ละพารามิเตอร์มี 3 ค่าที่เป็นไปได้

กรณีศึกษาที่ 7: 7 พารามิเตอร์ (2 พารามิเตอร์มี 3 ค่าที่เป็นไปได้ 5 พารามิเตอร์มี 2 ค่าที่เป็นไปได้)

กรณีศึกษาที่ 8: 7 พารามิเตอร์ โดยแต่ละพารามิเตอร์มี 3 ค่าที่เป็นไปได้

กรณีศึกษาที่นำมาใช้ในการทดลอง กรณีศึกษาที่ 1 และกรณีศึกษาที่ 2 นำมาจากงานวิจัยของ ดาวิด เอ็ม โคเฮน [7] และเพื่อให้เห็นความแตกต่างของจำนวนกรณีทดสอบที่สร้างขึ้นจากอัลกอริทึมที่นำเสนอกับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี จึงเพิ่มจำนวนพารามิเตอร์เข้าไปในกรณีศึกษาที่ 3 ถึงกรณีศึกษาที่ 8

เนื่องจากวิธีเออีทีจีเป็นอัลกอริทึมเชิงไม่กำหนด (nondeterministic) เพราะในแต่ละขั้นตอนของการสร้างกรณีทดสอบจะใช้วิธีการสุ่มค่าที่เป็นไปได้ของพารามิเตอร์มาสร้างเป็นกรณีทดสอบ และส่วนหนึ่งของอัลกอริทึมที่นำเสนอนำมาจากอัลกอริทึมไอพีโอซึ่งเป็นอัลกอริทึมเชิงไม่กำหนด ดังนั้น อัลกอริทึมที่นำเสนอจึงเป็นอัลกอริทึมเชิงไม่กำหนดด้วย ในการทดลองสร้างกรณีทดสอบจึงได้ทดลองสร้างกรณีทดสอบที่มีจำนวนน้อยที่สุดที่สร้างได้ แต่ไม่สามารถบอกได้ว่ากรณีทดสอบที่สร้างขึ้นมานั้น เป็นกรณีทดสอบที่มีจำนวนน้อยที่สุดในกรณีศึกษานั้นๆ แล้ว ด้วยสาเหตุดังกล่าว จึงทำให้ในบางกรณีจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอจะมีจำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี แต่ในบางกรณีจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอจะมีจำนวนมากกว่า

จะเห็นได้ว่า ในบางกรณีจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอจะมีจำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี แต่ในบางกรณีจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอจะมีจำนวนมากกว่า ดังนั้น จึงต้องทำการเปรียบเทียบด้วยวิธีทางสถิติเพื่อพิจารณาว่าอัลกอริทึมที่นำเสนอมีจำนวนกรณีทดสอบน้อยกว่าวิธีเออีทีจีอย่างมีนัยสำคัญ (significant) หรือไม่

ในการทดลองนี้ ต้องการทราบว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอน้อยกว่าหรือเท่ากับจำนวนกรณีทดสอบของอัลกอริทึมเออีทีจีหรือไม่ ดังนั้นจะต้องทดสอบโดยใช้วิธีทดสอบแบบจับคู่ โดยขั้นแรกจะทดสอบว่าจำนวนกรณีทดสอบมีการแจกแจงแบบปกติหรือไม่ เนื่องจากจำนวนกรณีทดสอบมีไม่เกิน 50 ตัว จึงใช้การทดสอบแชพิโล วิลค์

(shapiro-wilk) โดยกำหนดให้ *A* คือ จำนวนกรณีทดสอบที่สร้างด้วยอัลกอริทึมเออีทีจีและ *B* คือ กรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอ จะได้สมมติฐานสำหรับการทดสอบ *A* คือ

H_0: *A* มีการแจกแจงแบบปกติ

H_1: *A* ไม่ได้มีการแจกแจงแบบปกติ

กำหนดระดับนัยสำคัญที่ 0.01 จากการทดสอบได้ค่า p=0.102 ซึ่งมากกว่า 0.01 จึงไม่สามารถปฏิเสธ H_0 ได้ ดังนั้น *A* มีการแจกแจงแบบปกติที่ความเชื่อมั่น 99% และทดสอบ *B* โดยวิธีเดียวกับข้างต้นได้ p=0.04 ซึ่งมากกว่า 0.01 เช่นกัน ดังนั้น *B* มีการแจกแจงแบบปกติที่ความเชื่อมั่น 99%

เนื่องจากข้อมูลทั้งสองมีการแจกแจงแบบปกติจึงใช้การทดสอบที่ใช้พารามิเตอร์ (parametric) คือ การทดสอบผลต่างระหว่างค่าเฉลี่ย 2 ประชากรแบบจับคู่ โดยมีสมมติฐาน คือ

H_0: $\mu_A \geq \mu_B$

H_1: $\mu_A < \mu_B$

กำหนดระดับนัยสำคัญที่ 0.01 จากการทดสอบได้ค่า p=0.217 และ t=1.357 ซึ่งจะเห็นได้ว่า t มีค่าเป็นบวกและ $p/2$ ซึ่งเท่ากับ 0.1085 มีค่ามากกว่า 0.01 จึงไม่สามารถปฏิเสธ H_0 ได้ ดังนั้น จำนวนกรณีทดสอบเฉลี่ยที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอมีจำนวนน้อยกว่าหรือเท่ากับจำนวนกรณีทดสอบเฉลี่ยที่สร้างขึ้นด้วยอัลกอริทึมเออีทีจี ที่ความเชื่อมั่น 99%

บทที่ 5

สรุปผลการวิจัย และข้อเสนอแนะ

5.1 สรุปผลการวิจัย

การสร้างกรณีทดสอบแบบเอ็นเวย์ เป็นการสร้างกรณีทดสอบสำหรับระบบที่พารามิเตอร์ถูกแบ่งออกเป็นกลุ่ม ซึ่งแต่ละกลุ่มมีความสัมพันธ์กันแบบแพร์ไวส์ และพารามิเตอร์ที่อยู่ต่างกลุ่มกันนั้น จะมีความสัมพันธ์กันหรือไม่ก็ได้ กรณีทดสอบที่สร้างขึ้นมาจะมีประสิทธิภาพในการทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบ โดยไม่ขัดต่อเงื่อนไขบังคับ

วิทยานิพนธ์นี้ได้เสนอแนวทางในการออกแบบอัลกอริทึมซึ่งสามารถสร้างกรณีทดสอบสำหรับทดสอบระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ได้ โดยทำการปรับปรุงมาจากอัลกอริทึมไอพีโอ ที่สามารถสร้างกรณีทดสอบได้เฉพาะระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์เท่านั้น ซึ่งขั้นตอนการสร้างกรณีทดสอบ เริ่มจากการกำหนดกลุ่มของพารามิเตอร์ขึ้นมาเป็นความสัมพันธ์ก่อน แล้วนำความสัมพันธ์ที่ได้กำหนดขึ้นมาสร้างเงื่อนไขบังคับ แล้วจึงสร้างปัญหาย่อยโดยพิจารณาจากเงื่อนไขบังคับ จากนั้นจึงหากรณีทดสอบร่วมของปัญหาย่อยและนำกรณีทดสอบร่วมที่ได้มาทำให้เป็นกรณีทดสอบที่สมบูรณ์ในขั้นตอนสุดท้าย

ข้อดีของการสร้างกรณีทดสอบเพื่อใช้กับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ด้วยอัลกอริทึมที่ปรับปรุงมานั้น คือ การสร้างกรณีทดสอบที่สามารถทดสอบได้ครอบคลุมทุกกรณีที่ต้องการทดสอบโดยไม่ขัดต่อเงื่อนไขบังคับ และจากกรณีศึกษาจะพบว่าจำนวนกรณีทดสอบที่สร้างขึ้น จะน้อยกว่าการสร้างกรณีทดสอบด้วยวิธีการวิเคราะห์ค่าขอบเขต วิธีการทดสอบสภาพทนทาน และวิธีการทดสอบกรณีเลวร้ายที่สุดจากทฤษฎีการสร้างกรณีทดสอบในบทที่ 2 และจำนวนกรณีทดสอบที่สร้างขึ้น ส่วนมากจะมีจำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีการเออีทีจี ตามผลการเปรียบเทียบในบทที่ 4 ซึ่งจำนวนกรณีทดสอบที่น้อยนั้นมีผลทำให้ลดค่าใช้จ่าย และเวลาที่ใช้ในการทดสอบลงไปด้วย

จากการทดลองพบว่าการเปรียบเทียบจำนวนกรณีทดสอบในกรณีที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ซึ่งสร้างขึ้นด้วยอัลกอริทึมที่ได้ปรับปรุงมากกับจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีไอพีโอ พบว่าเมื่อพารามิเตอร์ที่ใช้ในระบบมีจำนวนน้อย สามารถลดขั้นตอนการสร้างกรณีทดสอบได้ด้วยการจัดเรียงพารามิเตอร์ที่มีค่าที่เป็นไปได้มากที่สุดก่อน แต่หากเป็นระบบที่มีพารามิเตอร์จำนวนมาก ขั้นตอนที่ใช้ในการสร้างกรณีทดสอบจะไม่แตกต่างกัน

ปัจจัยที่ทำให้ไม่สามารถลดขั้นตอนในการสร้างกรณีทดสอบเมื่อระบบที่นำมาใช้นั้นมีพารามิเตอร์มาก เพราะว่าต้องใช้กรณีทดสอบเพื่อทำการทดสอบให้ครบทุกกรณีจำนวนมากจากอัลกอริทึมไอพีโอ ในขั้นตอนการเติบโตแนวนอน จะได้จำนวนกรณีทดสอบที่ทั้งหมด คือ

จำนวนค่าที่เป็นไปได้ของพารามิเตอร์แรกคูณจำนวนค่าที่เป็นไปได้ของพารามิเตอร์ที่สอง ซึ่งหากในระบบมีพารามิเตอร์จำนวนมาก กรณีทดสอบที่สร้างขึ้นในขั้นตอนนี้จึงไม่สามารถที่ทำการทดสอบได้ครอบคลุมทุกกรณี ดังนั้น การเรียงลำดับของพารามิเตอร์ที่นำเข้าไปสร้างกรณีทดสอบใหม่นั้น จะลดขั้นตอนการสร้างกรณีทดสอบได้ก็ต่อเมื่อ พารามิเตอร์ในระบบมีจำนวนน้อย

พิจารณาทางด้านจำนวนกรณีทดสอบ การเรียงลำดับของพารามิเตอร์ที่จะนำมาสร้างกรณีทดสอบ เมื่อระบบที่นำมาสร้างกรณีทดสอบใช้พารามิเตอร์จำนวนมาก อาจมีผลทำให้จำนวนกรณีทดสอบที่สร้างขึ้นลดลง เพราะว่าเมื่อทำการเรียงลำดับของพารามิเตอร์แล้ว จะทำให้ในขั้นตอนการเติบโตแนวนอน นั้นสามารถสร้างกรณีทดสอบออกมาได้มาก ทำให้สามารถทดสอบครอบคลุมมากขึ้น ซึ่งมีผลทำให้กรณีทดสอบที่สร้างขึ้นในขั้นตอนการเติบโตแนวตั้ง มีจำนวนน้อยลง ทำให้จำนวนของกรณีทดสอบรวมทั้งหมด มีจำนวนกว่าการเรียงลำดับของพารามิเตอร์จากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้น้อยก่อน

จากผลการทดลองการเรียงอันดับพารามิเตอร์ตามจำนวนค่าที่เป็นไปได้ โดยเรียงจากพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้มากไปพารามิเตอร์ที่มีจำนวนค่าที่เป็นไปได้น้อย แม้ว่าจะต้องเสียเวลาในการเรียงลำดับของพารามิเตอร์ แต่สามารถทำให้จำนวนกรณีทดสอบที่สร้างขึ้นมีจำนวนลดลง ซึ่งจะทำให้จำนวนกรณีทดสอบสุดท้ายที่สร้างขึ้นในกรณีที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์มีจำนวนลดลงด้วย

จากผลการทดลองของ คัว ชุง ไถ และหยูเล่ย [9, 10] พบว่าหลายกรณีศึกษาที่ได้ทำการทดลอง จำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีไอพีโอ น้อยกว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี ดังนั้น ในงานวิจัยนี้ ได้นำเอาอัลกอริทึมไอพีโอ มาปรับปรุงให้ใช้สร้างกรณีทดสอบเมื่อพารามิเตอร์ในระบบมีความสัมพันธ์กันแบบเอ็นเวย์ จึงทำให้มีหลายกรณีที่จำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่ปรับปรุงนั้นมีจำนวนน้อยกว่าจำนวนกรณีทดสอบที่สร้างขึ้นด้วยวิธีเออีทีจี ดังใน บทที่ 4

5.2 ข้อจำกัดของงานวิจัย

5.2.1 กรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมนี้ ใช้กับระบบที่มีการกำหนดค่าต่างๆ ของระบบเท่านั้น

5.2.2 ระบบที่นำมาสร้างกรณีทดสอบต้องมีพารามิเตอร์อย่างน้อย 2 พารามิเตอร์ เนื่องจากอัลกอริทึมที่นำมาสร้างกรณีทดสอบ เริ่มต้นจากการจับคู่ค่าที่เป็นไปได้ของ 2 พารามิเตอร์แรก

5.3 ปัญหาและอุปสรรค

การสร้างกรณีทดสอบสำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์นั้น ในอัลกอริทึมที่ได้นำเสนอไปนั้น ผู้ทดสอบจะต้องกำหนดกลุ่มความสัมพันธ์ให้มีความเหมาะสมเพื่อใช้ในการสร้างกรณีทดสอบ

การกำหนดเงื่อนไขบังคับ จะขึ้นอยู่กับกลุ่มความสัมพันธ์ที่ผู้ทดสอบเป็นผู้กำหนด เพื่อจะนำไปสร้างเป็นปัญหาย่อย ในขั้นตอนนี้จะเกิดความซับซ้อนในการเลือกปัญหาย่อย หากปัญหาย่อยที่สร้างขึ้นนั้นมีมากกว่าหนึ่งปัญหาย่อย โดยการแก้ไขในงานวิจัยนี้ จะเลือกปัญหาย่อยชุดใดก็ได้ที่มีพารามิเตอร์มากที่สุด

ในการสร้างกรณีทดสอบของปัญหาย่อยด้วยอัลกอริทึมที่นำเสนอ นำมาจากอัลกอริทึมไอพีโอ ซึ่งอัลกอริทึมไอพีโอที่ใช้เป็นอัลกอริทึมเชิงไม่กำหนด ทำให้ไม่สามารถบอกได้ว่ากรณีทดสอบที่สร้างขึ้นมาเป็นกรณีทดสอบที่ดีที่สุด ด้วยสาเหตุดังกล่าว ทำให้จำนวนกรณีทดสอบที่สร้างขึ้นด้วยอัลกอริทึมที่นำเสนอในแต่ละครั้งอาจมีจำนวนไม่เท่ากัน

5.4 ข้อเสนอแนะในการพัฒนาเพิ่มเติม

แนวทางในการปรับปรุงอัลกอริทึมสำหรับสร้างกรณีทดสอบเพิ่มเติม มี 3 แนวทาง

5.3.1 ในงานวิจัยนี้ ได้นำอัลกอริทึมไอพีโอ มาช่วยในการสร้างกรณีทดสอบ โดยได้มาจากการทดลองเพื่อลดขั้นตอนในการสร้างกรณีทดสอบจากสองขั้นตอนเหลือเพียงขั้นตอนเดียว แต่อย่างไรก็ตาม เมื่อพารามิเตอร์มีจำนวนมาก ก็ไม่สามารถลดขั้นตอนลงได้ ดังนั้น จึงเป็นแนวทางให้ศึกษาเพื่อลดขั้นตอนการสร้างกรณีทดสอบของอัลกอริทึมไอพีโอต่อไป

5.3.2 เนื่องจากงานวิจัยนี้ ได้นำอัลกอริทึมไอพีโอ มาปรับปรุงเพื่อให้ใช้กับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบเอ็นเวย์ได้ ดังนั้น จึงเป็นแนวทางให้นำอัลกอริทึมอื่นที่สามารถสร้างกรณีทดสอบได้เฉพาะระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ มาปรับปรุงให้สามารถสร้างกรณีทดสอบให้กับระบบที่มีความสัมพันธ์แบบเอ็นเวย์ด้วย เช่น นำอัลกอริทึมในงานวิจัยของ โซลเหมิน ไมตี้ ที่ได้นำเสนอในปี 2005 [6] ซึ่งในงานวิจัยได้เสนอวิธีการสร้างกรณีทดสอบสำหรับระบบที่พารามิเตอร์มีความสัมพันธ์กันแบบแพร์ไวส์ ซึ่งสามารถสร้างกรณีทดสอบได้จำนวนน้อยกว่าทั้งวิธีไอพีโอ และวิธีเออีทีจี มาทำการปรับใช้แทนอัลกอริทึมไอพีโอ เป็นต้น

5.3.3 จากขั้นตอนการทำคำตอบให้สมบูรณ์ ในบทที่ 3 มีแนวทางในการพัฒนา คือ การปรับปรุงวิธีในการค้นหาคำตอบของอัลกอริทึม เพื่อให้ได้จำนวนกรณีทดสอบที่เหมาะสมมากยิ่งขึ้น ซึ่งมีผลทำให้จำนวนกรณีทดสอบที่สร้างขึ้นน้อยลงกว่าเดิมได้

รายการอ้างอิง

[1] S. R. Dalal, and Colin L. Mallows. Factor-Covering Designs for Testing Software. In **Technometrics 40(03)**, pp. 234-243. August, 1998.

[2] Vijay. N. Little Joe Model of Software Testing. In **3rd Annual International Software Testing Conference**, 2001.

[3] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. The Combinatorial Design Approach Test Generation. In **IEEE International on Software Reliability Engineering**, Vol. 13, No. 5, pp. 83-89. September, 1996.

[4] Paul C. Jorgensen. **Software Testing A Craftsman's Approach**. 2nd ed. pp. 79-95. Boca Raton London New York Washington, D.C.: CRC Press, 2002.

[5] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. In **IEEE Transactions on software engineering**, Vol. 23, No. 7, pp. 437-443. July, 1997.

[6] Soumen Maity, Amiya Nayak, Marzia Zaman, Nita Bansal, and Alka Srivastava. An Improved Test Generation Algorithm for Pair-Wise Testing. In **Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)**, 2005.

[7] Susan Khur, and Peter Grogono. Using Genetic Algorithm and Formal Concept Analysis to Generate Branch Coverage Test Data Automatically. In **Proceedings of the 19th International Conference on Automated Software Engineering (ASE'04)**, 2004.

[8] Alan W. Williams, and Robert L. Probert. A Practical Strategy for Testing Pair-wise Coverage of Network Interfaces. In **Proceeding of The Seventh International Symposium on Software Reliability Engineering (ISSRE '96)**, pp. 246-254. 1996.

[9] Yu Lei, and K.C. Tai. In-Parameter-Order: A Test Generation Strategy for Pair-wise testing. In **IEEE Transactions on software engineering**, 1998.

[10] Yu Lei, and K.C. Tai. Short Papers: A Test Generation Strategy for Pair-wise Testing. In **IEEE Transactions on software engineering**, Vol. 28, No. 1, pp. 109-111. January, 2002.

[11] Pressman, R.S. **Software Engineering: A practitioner's approach**. Singapore: McGraw-Hill, 1992.

[12] Maaret Pyhajarvi, Kristian Rautiainen, and Juha Itkonen. Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects. In **Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)**, 2002.

[13] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B.M. Horowitz. Model-Based Testing in Practice. In **Proceedings of ICSE'99**. ACM Press, May, 1999.

[14] สมชาย ประสิทธิ์จูตระกูล. **การออกแบบและวิเคราะห์อัลกอริทึม**. ประเทศไทย: NECTEC, 2545.

[15] D.M. Cohen, S. R. Dalal, A. Kajla, and G.C. Patton. The Automatic Efficient Test Generator (AETG) System. In **Internal Bellcore Technical Memorandum**, 1993.

ภาคผนวก

ภาคผนวก

ผลงานตีพิมพ์ในงาน NCSEC 2006

An improvement of n-way test case generation algorithm

Kitipoom Chaisuwan and Athasit Surarerks
ELITE (Engineering Laboratory in Theoretical Enumerable System)
Faculty of Engineering, Chulalongkorn University
Pathumwan, Bangkok 10330, Thailand
Email: g47kch@cp.eng.chula.ac.th, athasit@cp.eng.chula.ac.th

Abstract

Pair-wise testing is a software testing that combines all pairs of input values of each parameter while n-way testing generates only the necessary test cases. In this paper, we introduce an improvement version of In-Parameter-Order algorithm (IPO) proposed by Kuo-Chung Tai and Yu Lei for n-way test case generation. The concept of improvement points to the number of test cases. In our experiment, their algorithms such as AETG, IPO and our algorithm are focused for n-way testing. The results show that the number of test cases generated by our algorithm is not higher than the number of test cases obtained from AETG.

Key Words: Software testing, Pair-wise testing, N-way testing, Combination, IPO algorithm, AETG.

1. Introduction

Software testing is an important part of software develop life cycle (SDLC). Nevertheless, software testing is very expensive and consumes more time than other phases. Software testing can take as much as 20-30% of the total development budget of software project [3]. Moreover, software testing takes part all SDLC. Many researchers have aimed at reducing its cost and time.

There are many parts of testing level in software testing such as function testing, structure testing, integration testing, system testing and object-oriented testing may be tested by pair-wise testing. Pair-wise testing is known for its effectiveness in different types of software testing [1] and can test for any part of SDLC. The problem of generating the minimum test set for pair-wise testing is NP-complete [7].

This paper proposes a novel approach for software testing by improving IPO algorithm for the n-way test case generation. The IPO algorithm has the benefit of reusing old test cases when new parameters are added [4].

In this new approach, testers identify parameters those are possible for n-way definition which each relation is not a subset of other relations. The testers use the improved IPO algorithm to create a test plan that covers all pair-wise or n-way combinations of the test parameters.

Our research motivations are as follows: Since a parameter can be occurred in several relations, in order to reduce the number of test cases, the intersection of relations must be first considered. This can also reduce the time needed for generating all test cases.

Some researches concerning generation of test cases are similar to our research such as AETG (Automatic Efficient Test Generator) proposed by David M. Cohen, Siddhartha R. Dalal. Michael L. Fredman and Gardner C. Patton in 1997 [1].

Our experiments focus on comparison between AETG and our algorithm in the case of N-way testing. The experimental results demonstrate that the averaged number of test cases obtained from our algorithm is less than those from AETG.

Some background knowledge of pair-wise testing, n-way testing and constraints are recalled in Section 2. Section 3 describes an improved algorithm for n-way test case generation. Section 4 shows our proposed algorithm for n-way test case generation which improved from IPO algorithm. Some experiments are introduced for comparing the number of test cases obtained from AETG and from our algorithm in Section 5. Finally summary and conclusion are stated in Section 6.

2. Backgrounds

2.1 Pair-wise definition

Pair-wise testing is an alternative way for testing all possible combinations of parameters including all possible values. Set of test cases is generated that covers all combinations of all selected test data for all possible values for each pair of parameters. Pair-wise testing is also referred to as all-pair testing and 2-way testing.

2.2 N-way definition

In some situations, it is not in the case that all pair of parameters must be tested. The condition may be described by some relations. This kind of testing is so-called *N-way* testing. Parameters are separated into several groups. All possible combinations in each group must be included into the test set.

It is also in the case that some parameters can be in many groups (relations). One must be valid; a relation should not be a subset of others. These relations are said to be suitable for test case generation. This can be explained by two following examples.

Example1

Suppose that two relations G_1 and G_2 are expressed by the same parameters $\{P_1, P_2, P_3, \ldots, P_n\}$. Describe the situation that both relations are suitable.

Let C_1 is a set of all combination of members in G_1 and let C_2 is a set of all combination of members in G_2. if $Card(C_1) > Card(C_2)$ then

- $C_2 - C_1$ is not an empty set then G_2 is suitable for test cases generating.
- $C_2 - C_1$ is an empty set, $\emptyset$, this implies that $G_2 \subseteq G_1$ or the relation G_2 is not suitable for test cases generating. □

Example2

Suppose that G_1 is expressed by parameters in $\{P_1, P_2, P_3, \ldots, P_n\}$ and G_2 is described by $\{Q_1, Q_2, Q_3, \ldots, Q_m\}$ where all P_i are difference from Q_j. Describe the situation that both relations are suitable.

It is obvious that each element in relation G_1 has not a relationship to any elements in G_2. These relations are then suitable for test cases generation. □

2.3 Constraint for n-way IPO algorithm

A constraint can be expressed, using multiple relations. It may be more efficient to express them explicitly by using impossible cases.

A set of impossible case scenarios is also referred as unnecessary cases. Test cases are generated with n-way testing, not covering the unnecessary case. The set of unnecessary cases is called *constraint*.

3. Concept of improvement

In this section, we give a skeleton of pair-wise IPO algorithm [6] described by Yu Lei, and K.C. Tai in 2002. Our aim is to extend an IPO algorithm for covering n-way test case generation. We also remark that IPO algorithm can be simple improved to speed up in some cases. In fact, the number of generating steps will be decreased if the input parameters are reorganized.

We start by given a draft description of IPO algorithm using an example. The algorithm is composed of two steps: horizontal growth and vertical growth processes. For instance, let an application contains three parameters: *A*, *B*, and *C* where each has many possible values as illustrated by Table 1. The horizontal growth process generates test cases using all difference pairs of value of parameters *A* and *B*. The test cases must be completed by introducing the rest of parameters. The vertical growth process is performed whenever the test cases from the horizontal growth cannot cover all pair-wise cases.

Table 1. The values of parameters *A,B* and *C*

A	*B*	*C*
A_1	B_1	C_1
A_2	B_2	C_2
		C_3

The horizontal growth generates only four test cases, such as (A_1,B_1), (A_1,B_2), (A_2,B_1), and (A_2,B_2). Parameter *C* must be introduced into these test cases, then it is obtained (A_1,B_1,C_1), (A_1,B_2,C_2), (A_2,B_1,C_3), and (A_2,B_2,C_1). Since these cannot cover the pair of A_2 and C_2 for example, the vertical growth must be processed. This step is to generate a minimum number of test cases which contain all uncovered pair-wise cases. For instance, two test cases are generated, (A_1,B_2,C_3) and (A_2,B_1,C_2). It is seen that these six test cases cover all pair-wise cases.

Let us rearrange all input parameters as shown in Table 2. The two parameters with the most number of values are selected to be processed in the horizontal growth step. For instance, the parameters *C* and *A* are chosen. Then six pairs are generated: (C_1,A_1), (C_1,A_2), (C_2,A_1), (C_2,A_2), (C_3,A_1) and (C_3,A_2). Now the parameter *B* must be introduced into these pairs. Without performing the vertical growth step, it is obtained six test cases which are (C_1,A_1,B_1), (C_1,A_2,B_2), (C_2,A_1,B_2), (C_2,A_2,B_1), (C_3,A_1,B_2) and (C_3,A_2,B_1).

Table 2. Rearrange parameters *A,B* and *C* by the number of values

C	*A*	*B*
C_1	A_1	B_1
C_2	A_2	B_2
C_3		

By rearranging all parameters using the number of values, the number of processes can probably be decreased, although the number of test cases may not be decreased. All solutions in this case are generated only in the horizontal growth step.

Now, we propose three steps of extending IPO algorithm to support an n-way testing criteria. Using divide and conquer technique, the problem is divided into several sub-problems depended on all given relations. All sub-problems must be solved first. The final solution is obtained by combining all sub-solutions together. The algorithm is described by three following steps:

1. Constraints identification
   This step is to prepare all input parameters and to construct some constraints of the application.
2. Common test case generation
   All cases which include in many relations are generated in this step.
3. Solution completion
   The solution (set of all test cases) must be completed in to process.

3.1 Constraints identification

Constraint means pair of parameters that must be ignored for testing. In order to reduce the number of test cases, all unnecessary test cases can be removed from the testing criteria by introducing the concept of constraints. Constraints can be determined using all given relations of parameters. An algorithm for identifying constraints is described as follows:

Algorithm: constraints identification
Input: *All relations defined on parameters*
Output: *All constraints*
Begin
Construct a set of all combinations of all parameters (π)
Construct a set of all combinations of each relation R_i (πR_i)
Constraints are all combinations that are in π but not in πR_i.
End

Let us see an application containing two relations as shown by Table 3. Both relations contain the same parameters *A*, *B*, *C*, and *D* but some parameters have different values.

Since there are four parameters and each contains three values, the set of all combinations of all parameters must contain 54 pairs. The set πR_1 contains 45 pairs and πR_2 contains 29 pairs. Then the set of constraints contains only one element:

$$\pi - \pi R_1 - \pi R_2 = \{ (A_3, B_3) \}.$$

Table 3. Example of two relations identified by the tester

Relation 1			
A	*B*	*C*	*D*
A_1	B_1	C_1	D_1
A_2	B_2	C_2	D_2
	B_3	C_3	D_3

Relation 2			
A	*B*	*C*	*D*
A_3	B_1	C_1	D_1
	B_2	C_2	D_2
		C_3	D_3

3.2 Common test case generation

In this section, we construct the sub-problem (*MIR*: a set of parameters) that does not exist any two parameters appeared in the same constraint. For instance, the constraint is (A_3,B_3). Thus, there are two possible *MIR*'s as follows: the set of *A*, *C*, and *D* and the set of *B*, *C*, and *D*, then we select the largest *MIR* (*i.e.*, the maximum number of elements.) In this case, the second is selected to be *MIR* as illustrated by Table 4.

Table 4. Relation with respect to *MIR*

B	*C*	*D*
B_1	C_1	D_1
B_2	C_2	D_2
B_3	C_3	D_3

We generate all test cases for all possible pairs of parameters in *MIR* using the IPO algorithm. It is obtained the set πIR that contains 27 pairs as follows: (B_1,C_1), (B_1,C_2), (B_1,C_3), (B_2,C_1), (B_2,C_2), (B_2,C_3), (B_3,C_1), (B_3,C_2), (B_3,C_3), (B_1,D_1), (B_1,D_2), (B_1,D_3), (B_2,D_1), (B_2,D_2), (B_2,D_3), (B_3,D_1), (B_3,D_2), (B_3,D_3), (C_1,D_1), (C_1,D_2), (C_1,D_3), (C_2,D_1), (C_2,D_2), (C_2,D_3), (C_3,D_1), (C_3,D_2) and (C_3,D_3).

After that, test cases are generated for covering pairs in πIR. The results are (B_1,C_1,D_1), (B_1,C_2,D_2), (B_1,C_3,D_3), (B_2,C_1,D_3), (B_2,C_2,D_1), (B_2,C_3,D_2), (B_3,C_1,D_2), (B_3,C_2,D_3) and (B_3,C_3,D_1). These test cases are called *common test cases*.

3.3 Solution completion

Since common test cases are generated from elements in *MIR*, all pairs in πIR are covered by these test cases. The rest is to create test cases for covering all pairs in all πR_i which are not in πIR. The result πR contains only 26 pairs as follows: (A_1,B_1), (A_1,B_2), (A_1,B_3), (A_2,B_1), (A_2,B_2), (A_2,B_3), (A_3,B_1), (A_3,B_2), (A_1,C_1), (A_1,C_2), (A_1,C_3), (A_2,C_1), (A_2,C_2), (A_2,C_3),

(A_3,C_1), (A_3,C_2), (A_3,C_3), (A_1,D_1), (A_1,D_2), (A_1,D_3), (A_2,D_1), (A_2,D_2), (A_2,D_3), (A_3,D_1), (A_3,D_2) and (A_3,D_3).

To complete the set of all test cases, each parameter which is not in the *MIR* must be considered. A new test case is obtained by combining each common test case together with such parameter whenever this covers the maximum number of pairs in πR. For example, (B_1,C_1,D_1) must be combined with A_1 because this test cases can cover 3 pairs (*i.e.*, (A_1,B_1), (A_1,C_1), and (A_1,D_1).) Using this technique, all result test cases are obtained: (A_1,B_1,C_1,D_1), (A_3,B_1,C_2,D_2), (A_2,B_1,C_3,D_3), (A_3,B_2,C_1,D_3), (A_2,B_2,C_2,D_1), (A_1,B_2,C_3,D_2), (A_2,B_3,C_1,D_2), (A_1,B_3,C_2,D_3) and (A_1,B_3,C_3,D_1).

Two pairs, (A_3,C_3) and (A_3,D_1), are not covered by the result. The objective is to generate a minimum set of test cases that covers these two pairs. First, merge all possible pairs which do not contradict with all constraints. If the result contains some incomplete test case, any values of parameter can be defined if this does not appear in the constraint.

Consider two pairs: (A_3,C_3) and (A_3,D_1), two incomplete test cases are generated: $(A_3,\text{-},C_3,\text{-})$ and $(A_3,\text{-},\text{-},D_1)$. Since both of them can be merged together, it is obtained $(A_3,\text{-},C_3,D_1)$. Now, this test case covers all pairs in πR, but it is still an incomplete test case. We can see that B_1 or B_2 can be merged to this test case, but it is not in the case for B_3 (because (A_3,B_3) is a constraint.) The last test case is then (A_3,B_1,C_3,D_1).

The algorithm can be described in the next section.

4. N-way test case generation algorithm

Assume R_i contains parameter $P_1,P_2,\ldots,P_m$ where P_i contains value $v_{i,1}, v_{i,2}, \ldots, v_{i,ni}$;

Algorithm: n-way test case generation
Input: *All relations defined on parameters*
Output: *All test cases*
Begin
// Constraints identification
generates π;
generates πR_i;
construct the constraints;
// Common test cases generation
construct *MIR*, and generates πIR;
// Horizontal growth
let T be an empty set;
generate all combination of values of the first two parameters of *MIR*, and adding pairs to T;
if $(Card(T) \leq ni)$
{ **for** $1 < j \leq Card(T)$
extend the jth test in T by adding value v_j and remove from πIR pairs covered by the extended test;
}
else
{ **for** $1 < j \leq ni$
extend the jth test in T by adding value v_j and remove from πIR pairs covered by the extended test;
for $ni < j \leq Card(T)$
extend the jth test in T by adding one value of P_i such that the resulting test covers the most number of pairs in πIR, and remove from πIR pairs covered by the extended test;
}
// Vertical growth
let T' be an empty set;
for (each pair in πIR)
{ assume that the pairs contains value w of P_k, $1 \leq k \leq$ i, and value u of P_i;
if (T' contains a test with "-" as the value of P_k and u as the value of P_i)
modify this test by replacing the "-" with w;
else
add a new test to T' that has w as the value of P_k, u as the value of P_i, and "-" as the value of every other parameter;
}
$T = T \cup T'$;
// Solution completion
let πR be an empty set;
remove πIR pairs from πR_i, and add all pairs in πR_i to πR;
for (each test case in T)
{ **for** (other parameters of test cases in T)
extend the test case in T by adding one value of other parameter. The test case is not covers in the constraints. The resulting test covers the most number of pairs in πR, and replacing the test case to T, and remove from πR pairs covered by the extended test case;
};
let T'' be an empty set;
for (each pair in πR)
{ assume that the pairs contains value w of P_k, $1 \leq k \leq i$, and value u of P_i;
if (T'' contains a test with "-" as the value of P_k, $1 \leq k < i$, and value u of P_i)
modify test by replacing the "-" with w;
else
add a new test to T'' that has w as the value of P_k, u as the value of P_i, and "-" as the value of every other parameter;
};
$T = T \cup T''$;
End

The n-way test case generation algorithm can be described using a flowchart as Figure 1.

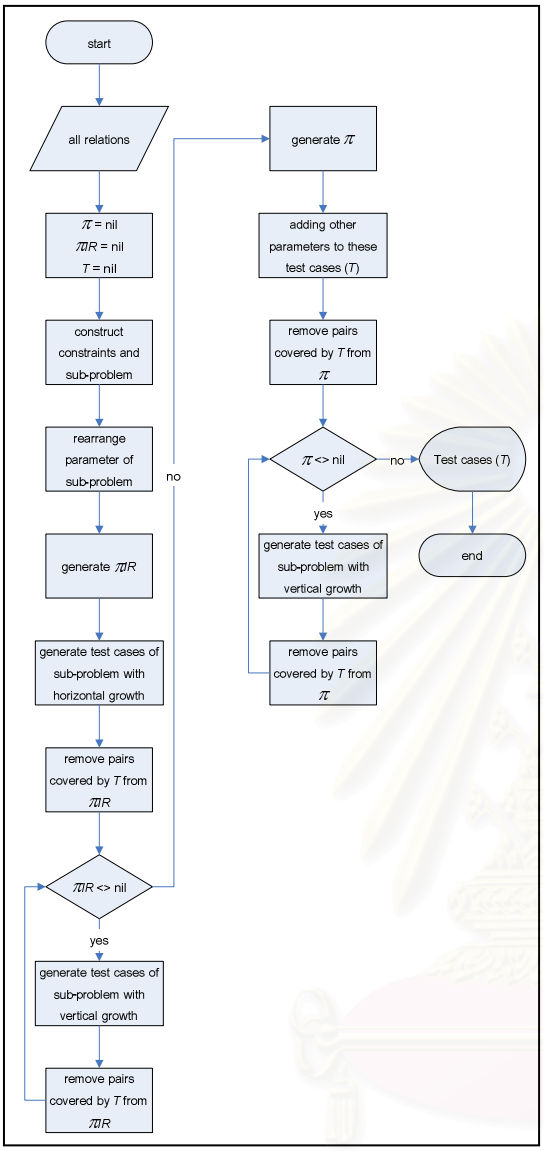


Figure 1. Flowchart of n-way test case generation.

5. Comparison with AETG

In this section, experiments on comparison between AETG and our algorithm are introduced. Section 3 explains three steps of test cases generation. The ten test cases of our solution are shown in Table 5.

The solution can be compared with the solution using AETG [5] which is shown in Table 6. In this case, both algorithms generate the same number of test cases.

Table 7 shows the number of test cases generated from our algorithm as well as AETG method. Our algorithm mostly generates less number of test cases comparing with AETG method.

Table 5. The test cases generation using our algorithm

A	B	C	D
A_1	B_1	C_1	D_1
A_3	B_1	C_2	D_2
A_2	B_1	C_3	D_3
A_3	B_2	C_1	D_3
A_2	B_2	C_2	D_1
A_1	B_2	C_3	D_2
A_2	B_3	C_1	D_2
A_1	B_3	C_2	D_3
A_1	B_3	C_3	D_1
A_3	B_1	C_3	D_1

Table 6. The test cases generation using AETG

A	B	C	D
A_1	B_2	C_3	D_2
A_2	B_3	C_1	D_2
A_3	B_1	C_2	D_2
A_1	B_3	C_2	D_3
A_2	B_1	C_3	D_3
A_3	B_2	C_1	D_3
A_1	B_1	C_1	D_1
A_2	B_2	C_2	D_2
A_3	B_1	C_3	D_1
A_1	B_3	C_3	D_1

Table 7. Comparison of number of test cases.

System	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
Our	8	10	9	12	9	16	9	17
AETG	8	10	8	15	10	16	9	19

Eight set of relations used in our experiments are as follows:

S1: 4 parameters (2 3-value parameter, 2 2-values parameters)
S2: 4 3-value parameters
S3: 5 parameters (2 3-value parameters, 3 2-value parameters)
S4: 5 3-value parameters
S5: 6 parameters (2 3-value parameters, 4 2-value parameters)
S6: 6 3-value parameters
S7: 7 parameters (2 3-value parameters, 5 2-value parameters)
S8: 7 3-values parameters

In addition, the test cases generated by both algorithms can use to test all cases in uncovered set when the parameter relationships are pair-wise or n-way. The number of cases covered by test cases from both algorithms is equal.

6. Conclusion

In this paper, we propose an n-way IPO algorithm which covers both pair-wise and n-way combinations of test cases for the application. From case study in section 5 the number of test cases generated by our algorithm is less than the number of test cases obtained from AETG in average. The n-way IPO algorithm can test in a variety of applications in term of function, structure, unit, system and interoperability testing. It generates both high-level test plans and detailed test cases. The experience with this new approach indicates that it can be used in widely applications.

7. References

[1] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transactions on software engineering, July 1997, Vol. 23, No. 7, pp. 437-443.

[2] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton, "The Combinatorial Design Approach Test Generation," IEEE International on Software Reliability Engineering, Sept 1996, Vol. 13, No. 5, pp. 83-89.

[3] D.M. Cohen, S. R. Dalal, A. Kajla, and G.C. Patton, "The Automatic Efficient Test Generator (AETG) System," Internal Bellcore Technical Memorandum, 1993.

[4] Myra B. Cohen, Peter B. Gibbons, Warwick B. Mugridge, and Charles J.Colbourn, "Constructing Test Suites for Interaction Testing," Proceedings of the 25th International Conference on Software Engineering, 2003, pp.38-48.

[5] Toshiaki Shina, Thtsuhiro Tsuchiya, and Tohru Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004.

[6] Yu Lei, and K.C. Tai, "Short Papers: A Test Generation Strategy for Pair-wise Testing," IEEE Transactions on software engineering, January 2002, Vol. 28, No. 1, pp. 109-111.

[7] Yu Lei, and K.C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pair-wise testing," IEEE Transactions on software engineering, 1998.

ประวัติผู้เขียนวิทยานิพนธ์

นายกิติภูมิ ชัยสุวรรณ เกิดเมื่อวันที่ 4 เมษายน พ.ศ. 2523 จบการศึกษาระดับมัธยมศึกษาตอนต้น และมัธยมศึกษาตอนปลายจากโรงเรียนวัดนวลนรดิศ เข้าศึกษาต่อในระดับปริญญาบัณฑิต สาขาวิชาคณิตศาสตร์ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี จนสำเร็จการศึกษาในปี พ.ศ. 2545 และในปี พ.ศ. 2547 เข้าศึกษาต่อในระดับปริญญามหาบัณฑิต สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย