ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

ปริญญา

วิทยาการคอมพิวเตอร์ วิทยาการคอมพิวเตอร์

สาขา **ภาควิชา**

เรื่อง การหลีกเลี่ยงข้อผิดพลาดโดยใช้การสกัดกฎจากเบย์เซียนเน็ตเวิร์คและตรวจสอบกฎโดยอัตโนมัติด้วยการโปรแกรมเชิงลักษณะ

Avoiding Fault using Rule Extraction from Bayesian Network and Automatic Checking with Aspect Oriented Programming

นามผู้วิจัย นางสาวกนกวรรณ แก้วทอง

ได้พิจารณาเห็นชอบโดย

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก .....

(อาจารย์อุษา สัมมาพันธ์, Ph.D.)

หัวหน้าภาควิชา .....

(ผู้ช่วยศาสตราจารย์ศิริกร จันทร์นวล, M.Sc.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ ..... **เดือน** ..... **พ.ศ.** .....

วิทยานิพนธ์

เรื่อง

การหลีกเลี่ยงข้อผิดพลาดโดยใช้การสกัดกฎจากเบย์เซียนเน็ตเวิร์คและตรวจสอบกฎโดยอัตโนมัติด้วยการโปรแกรมเชิงลักษณะ

Avoiding Fault using Rule Extraction from Bayesian Network and Automatic Checking with Aspect Oriented Programming

โดย

นางสาวกนกวรรณ แก้วทอง

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

เพื่อความสมบูรณ์แห่งปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

พ.ศ. 2555

กนกวรรณ แก้วทอง 2555: การหลีกเลี่ยงข้อผิดพลาดโดยใช้การสกัดกฎจากเบย์เซียนเน็ตเวิร์คและตรวจสอบกฎโดยอัตโนมัติด้วยการโปรแกรมเชิงลักษณะ ปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์) สาขาวิทยาการคอมพิวเตอร์ ภาควิชาวิทยาการคอมพิวเตอร์ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก: อาจารย์อุษา สัมมาพันธ์, Ph.D. 73 หน้า

ในการพัฒนาระบบเพื่อให้ระบบสามารถทำงานได้อย่างถูกต้องและมีประสิทธิภาพจำเป็นต้องมีการตรวจสอบข้อผิดพลาดต่างๆที่เกิดขึ้น โดยในการตรวจสอบจำเป็นต้องมีข้อมูลในการตรวจสอบซึ่งเรียกว่า กฎ และในการสกัดกฎนั้น นักพัฒนาระบบจำเป็นต้องศึกษาโครงสร้างของระบบอย่างละเอียดซึ่งต้องใช้เวลานาน ดังนั้นหากนำวิธีการเรียนรู้มาใช้ในการสกัดกฎจะช่วยลดระยะเวลาในการศึกษาระบบ นอกจากนั้นเมื่อตรวจพบข้อผิดพลาดแล้วจำเป็นต้องมีการจัดการกับข้อผิดพลาดเหล่านั้น ซึ่งหนึ่งในวิธีจัดการกับข้อผิดพลาด คือ การนำกฎมาใช้ในการหลีกเลี่ยงข้อผิดพลาดก่อนที่มันจะเกิดขึ้น ดังนั้นในงานวิจัยจึงได้สร้างระบบหลีกเลี่ยงข้อผิดพลาดโดยอาศัยข้อมูลที่ได้จากการทำงานจริงของระบบมาเรียนรู้เพื่อหาความสัมพันธ์ระหว่างเหตุการณ์ด้วยเบย์เซียนเน็ตเวิร์ค จากนั้นนำความสัมพันธ์ที่ได้มาใช้ในการสกัดกฎ เพื่อนำไปใช้ลดข้อผิดพลาดและป้องกันไม่ให้ระบบเข้าสู่สถานะไม่ปกติ โดยใช้การแทรกกฎตรวจสอบการทำงานและแก้ไขข้อผิดพลาดด้วย Aspect Oriented Programming (AOP)

ในงานวิจัยจะแบ่งการทดลองออกเป็น 2 ส่วน คือ (1) ส่วนของการสกัดกฎ เป็นการเก็บข้อมูลและนำข้อมูลมาเรียนรู้โดยใช้เบย์เซียนเน็ตเวิร์ค โดยใช้เครื่องมือ WEKA จากนั้นจะนำความสัมพันธ์ระหว่างข้อผิดพลาดและพฤติกรรมที่ทำให้เกิดข้อผิดพลาดที่ได้มาสกัดเป็นกฎ แล้วทำการตรวจสอบความถูกต้องและเวลาในการทำงานของกฎ และ (2) ส่วนของการหลีกเลี่ยงข้อผิดพลาด จะมีการนำกฎที่ได้จากข้อ (1) มาใช้ในการเปรียบเทียบกับเหตุการณ์ เพื่อตรวจสอบและปรับปรุงการทำงานเมื่อมีข้อผิดพลาดเกิดขึ้นโดยใช้ Aspect Oriented Programming

จากผลการทดลองพบว่าสามารถตรวจสอบเหตุการณ์ที่ตรงกับกฎและทำการหลีกเลี่ยงเหตุการณ์เหล่านั้นได้ ซึ่งช่วยให้โปรแกรมสามารถทำงานต่อไปได้

_____ _____ ___/____/____

ลายมือชื่อนิสิต ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

Kanokwan Kaewthong 2012: Avoiding Fault using Rule Extraction from Bayesian Network and Automatic Checking with Aspect Oriented Programming. Master of Science (Computer Science), Major Field: Computer Science, Department of Computer Science. Thesis Advisor: Miss Usa Sammapun, Ph.D. 73 pages.

In software development, we want to build software which can work correctly and efficiently. Therefore, we need a technique to detect errors in the software. To be able to detect errors, one can define rules specifying possible patterns of errors. However, in defining such rules, developers need to investigate and understand the software comprehensively, which can take a long time. Thus, using a learning technique to learn and extract rules can save developers' time. In addition, once errors are found, they should be managed by, for example, avoiding errors before occurring. Therefore, this research builds a failure-avoidance system to learn rules from application-level data derived from the running software. The learning technique finds the relationships between failures and behaviors that lead to failures and transform into rules. The rules are used to detect possible failures while the software is running. Then, Aspect Oriented Programming (AOP) is applied to avoid such failures.

Experiments in this work are divided into 2 parts which are (1) a rule extraction process and (2) a failure-avoidance process. A racing game case study is used in the experiments. The rule extraction process uses Bayesian Network in the WEKA tool to learn relationships between failures and behaviors that lead to failures. Then, the relationships are transformed into rules. The resulting rules are evaluated using performance measures and execution time. Finally, the extracted rules are used to monitor software runtime behaviors to detect possible failures, which are then avoided using Aspect Oriented Programming.

From the experimental results, we can accurately catch behaviors that lead to failure and avoid them.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

ผู้วิจัยขอกราบขอบพระคุณ อ.ดร.อุษา สัมมาพันธ์ ประธานกรรมการที่ปรึกษาวิทยานิพนธ์ ที่ให้คำปรึกษาในการเรียน การค้นคว้าวิจัย ตลอดจนการตรวจแก้ไขวิทยานิพนธ์จนกระทั่งเสร็จสมบูรณ์

ขอกราบขอบพระคุณอาจารย์ประจำภาควิชาวิทยาการคอมพิวเตอร์ทุกท่าน ที่ได้อบรมสั่งสอนและให้ความรู้อันเป็นประโยชน์ในการนำไปใช้ประโยชน์ต่อไป และขอขอบคุณเจ้าหน้าที่ภาควิชาวิทยาการคอมพิวเตอร์ทุกท่าน ที่ได้ให้ความช่วยเหลือและให้คำแนะนำต่างๆ ในการทำวิจัย

ขอขอบคุณพี่ๆ และเพื่อนๆ ในภาควิชาทุกคนที่คอยให้กำลังใจและให้คำปรึกษาในเรื่องต่างๆ ที่เป็นประโยชน์ในด้านการทำวิจัย

ด้วยความดีหรือประโยชน์อันใดเนื่องจากวิทยานิพนธ์เล่มนี้ ขอมอบแด่คุณพ่อ คุณแม่ ที่ได้อบรมและให้กำลังใจผู้วิจัยตลอดมา

กนกวรรณ แก้วทอง

ตุลาคม 2555

สารบัญ

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

การหลีกเลี่ยงข้อผิดพลาดโดยใช้การสกัดกฎจากเบย์เซียนเน็ตเวิร์คและตรวจสอบกฎโดยอัตโนมัติด้วยการโปรแกรมเชิงลักษณะ

Avoiding Fault using Rule Extraction from Bayesian Network and Automatic Checking with Aspect Oriented Programming

คำนำ

ในปัจจุบัน นักพัฒนาระบบพยายามปรับปรุงระบบให้มีการทำงานที่มีความถูกต้องและน่าเชื่อถือเพิ่มขึ้นเรื่อย ๆ โดยสังเกตได้จากงานวิจัยที่เกี่ยวข้องกับการตรวจสอบและการทำนายประสิทธิภาพของระบบ (Catal, 2011) แต่แทบจะเป็นไปไม่ได้ที่จะตรวจสอบและกำจัดข้อผิดพลาดที่มีทั้งหมดออกไปได้ เนื่องจากนักพัฒนาระบบไม่สามารถทดสอบระบบได้ครอบคลุมในทุกกรณี และไม่สามารถควบคุมสภาพแวดล้อมในการทำงานของระบบได้ ทำให้ยังมีการตรวจพบข้อผิดพลาดอยู่ และเนื่องจากข้อผิดพลาดบางอย่างเกิดขึ้นในขณะที่ระบบกำลังทำงานอยู่ในสภาพแวดล้อมจริงซึ่งมีความแตกต่างจากสภาพแวดล้อมที่ใช้ในการทดสอบ และการทดสอบดังกล่าวก็ไม่สามารถตรวจสอบหาข้อผิดพลาดในช่วงที่ระบบกำลังทำงานอยู่ได้ (runtime error) ดังนั้นจึงได้มีการพัฒนาวิธีการตรวจสอบที่สามารถตรวจสอบการทำงานได้ในขณะที่ระบบกำลังทำงาน ซึ่งช่วยให้ระบบมีการตรวจสอบข้อผิดพลาดที่ครอบคลุมยิ่งขึ้น โดยการดักจับเหตุการณ์หรือ พฤติกรรมที่เกิดขึ้นและเปรียบเทียบกับข้อมูลที่มีอยู่ โดยในการตรวจสอบข้อผิดพลาดที่เกิดขึ้นนั้นมักจะตรวจสอบเพื่อหาfault ซึ่งเป็นสาเหตุของข้อผิดพลาด ที่อยู่ภายในระบบ หรือ failure ซึ่งเป็นพฤติกรรมภายที่อยู่นอกเหนือจากพฤติกรรมที่เป็นความต้องการของระบบ (Hamill และ Goseva-Popstojanova, 2009)

ในการตรวจสอบระบบนั้นจะมีการใช้ข้อมูลในการตรวจสอบอยู่ 2 ประเภท คือ ข้อมูลที่ได้จากการวิเคราะห์การเขียนโปรแกรมโดยที่ไม่ต้องอาศัยการทำงานของระบบ (static analysis) เช่น จำนวน interface จำนวน static method เป็นต้น และข้อมูลที่ได้จากการวิเคราะห์ขั้นตอนการทำงานของระบบ (dynamic analysis) เช่น system-level data application-level data เป็นต้น

ข้อผิดพลาดที่เกิดจากการทำงานของระบบส่วนใหญ่สามารถตรวจสอบและแก้ไขได้ในขั้นตอนของการทดสอบ แต่ยังมีข้อผิดพลาดบางอย่างที่ตรวจสอบและแก้ไขได้ยาก นั่นคือ ข้อผิดพลาดที่เกิดจากสภาพแวดล้อมในการทำงาน เนื่องจากอยู่นอกเหนือการควบคุมของนักพัฒนาระบบ เช่น การเชื่อมต่อเน็ตเวิร์ค การใช้งานของผู้ใช้ การทำงานร่วมกับระบบอื่น เป็นต้น ดังนั้นการแก้ไขข้อผิดพลาดเหล่านี้จึงทำได้เพียงปกป้องระบบจากเหตุการณ์เหล่านั้น โดยการสร้างความทนทานให้กับระบบ (fault-tolerance)

การสร้างความทนทานให้กับระบบจะช่วยให้ระบบมีความสามารถในการตอบสนองหรือแก้ไขข้อผิดพลาดตามที่ได้กำหนดไว้ ซึ่งจะช่วยให้ระบบสามารถทำงานต่อไปได้ในสภาพแวดล้อมที่ไม่ปกติหรือกลับเข้าสู่สถานะปกติ ซึ่งสามารถทำได้โดยการสร้างระบบที่สามารถกู้คืนตนเองได้ (self-healing) โดยระบบสามารถตรวจสอบและจัดการกับข้อผิดพลาดได้ด้วยตนเอง หรือการสร้างระบบที่สามารถหลีกเลี่ยงข้อผิดพลาดที่จะเกิดขึ้นภายในระบบได้ (fault-avoidance) ซึ่งจะช่วยลดความเสี่ยงในการเกิดข้อผิดพลาด เป็นต้น

เพื่อให้ระบบมีการทำงานที่น่าเชื่อถือ งานวิจัยชิ้นนี้จึงนำเสนอการสกัดกฎโดยใช้หลักการของเบย์เซียนเน็ตเวิร์ค ซึ่งเป็นตัวจำแนก (classifier) ที่มีความสามารถในการเรียนรู้และจำแนกความสัมพันธ์ของข้อมูล โดยในงานวิจัยจะใช้เบย์เซียนเน็ตเวิร์คในการหาความสัมพันธ์ของสถานะไม่ปกติ (failure) กับพฤติกรรมที่จะทำให้ระบบอยู่ในสถานะไม่ปกติ ซึ่งเกิดจากการทำงานของผู้ใช้ โดยนำข้อมูลเหตุการณ์ซึ่งแสดงถึงพฤติกรรมการทำงานในระบบมาใช้ในการเรียนรู้ เพื่อสกัดกฎให้อยู่ในรูป IF-THEN สำหรับตรวจสอบและหลีกเลี่ยงพฤติกรรมเหล่านั้นก่อนที่มันจะเกิดขึ้น ซึ่งการสกัดกฎแบบนี้จะช่วยลดเวลาที่จะใช้ในการศึกษาและทำความเข้าใจในตัวโปแกรม ก่อนที่นักพัฒนาระบบจะทำการปรับปรุงแก้ไข เนื่องจากหากโปรแกรมมีขนาดใหญ่มากๆ การที่จะศึกษาระบบให้มีความเข้าใจที่ชัดเจนนั้นต้องใช้เวลานาน ซึ่งอาจไม่ทันต่อความจำเป็นในการใช้งาน ดังนั้นหากนำการสกัดกฎแบบนี้มาใช้ก็จะทำให้นักพัฒนาสามารถศึกษาโปรแกรมแบบคร่าวๆเพื่อใช้ในการเลือกตัวแปรที่จะทำมาใช้และกฎที่ได้ยังครอบคลุมเหตุการณ์ที่นักพัฒนาคาดไม่ถึง ซึ่งเป็นการช่วยโปรแกรมมีการทำงานที่ถูกต้องมากยิ่งขึ้น

ในงานวิจัยจะแบ่งงานออกเป็น 2 ส่วน คือ (1) ส่วนของการสกัดกฎ ในส่วนนี้จะเก็บข้อมูลพฤติกรรมการทำงานของโปรแกรมที่จะส่งผลให้เกิดข้อผิดพลาดขึ้นภายในระบบ จากนั้นจะนำข้อมูลที่ได้มาเรียนรู้ด้วยเบย์เซียนเน็ตเวิร์คเพื่อหาความสัมพันธ์และค่าความน่าจะเป็นในการเกิด

เหตุการณ์ที่มีพฤติกรรมที่จะทำให้ระบบอยู่ในสถานะไม่ปกติ จากนั้นจะนำข้อมูลที่ได้มาเขียนให้อยู่ในรูปกฎอย่างง่าย และ (2) ส่วนของการหลีกเลี่ยงข้อผิดพลาดจะมีการนำกฎที่ได้จากข้อ (1) มาใช้ในการเปรียบเทียบกับเหตุการณ์ที่เกิดขึ้นในขณะที่โปรแกรมกำลังทำงานอยู่ เพื่อตรวจสอบว่าเหตุการณ์นั้นจะทำให้ระบบเกิดข้อผิดพลาดหรือไม่ โดยใช้ AspectJ ซึ่งเป็นเครื่องมือที่ใช้ในการเขียนโปรแกรมเชิงลักษณะ

ในงานวิจัยนี้ได้ตั้งสมมติฐานไว้ว่า เราสามารถนำเบย์เซียนเน็ตเวิร์คมาใช้ในการสกัดกฎเพื่อหลีกเลี่ยงข้อผิดพลาดที่จะเกิดขึ้นในระบบได้ และเพื่อพิสูจน์สมมติฐานนี้จึงนำเอาโปรแกรมประยุกต์ Racing Game 1.0 (J. Montgomery, 2011) ซึ่งเป็นโปรแกรมแข่งรถมาใช้เป็นกรณีศึกษา เนื่องจากการทำงานของโปรแกรมรถแข่งที่นำมาใช้ในการทดลองนั้นมีการทำงานที่ไม่ซับซ้อนและเข้าใจง่าย ซึ่งจะมีส่วนช่วยในการอธิบายวิธีการในการสกัดกฎและการนำไปใช้งานได้ชัดเจนยิ่งขึ้น และหากกฎที่ได้สามารถนำมาใช้งานกับโปรแกรมแข่งรถได้อย่างมีประสิทธิภาพ ดังนั้นก็น่าจะสามารถนำวิธีการสกัดกฎนี้มาใช้ในการตรวจสอบและหลีกเลี่ยงข้อผิดพลาดกับระบบอื่นๆ ได้เช่นกัน

วัตถุประสงค์

เพื่อนำเบย์เซียนเน็ตเวิร์คมาใช้ในการหาความสัมพันธ์ระหว่างสถานะไม่ปกติ (failure) กับเหตุการณ์ที่มีพฤติกรรม (behavior) ที่น่าจะทำให้โปรแกรมอยู่ในสถานะไม่ปกติ เพื่อนำไปใช้ในการสกัดกฎและหลีกเลี่ยงพฤติกรรมเหล่านั้นด้วยการเขียนโปรแกรมแบบ Aspect Oriented Programming โดยกฎที่ได้จะอยู่ในรูปแบบอย่างง่ายและสามารถตรวจสอบเหตุการณ์ที่มีพฤติกรรมที่ทำให้โปรแกรมอยู่ในสถานะไม่ปกติและหลีกเลี่ยงไม่ให้ระบบเข้าสู่สถานะไม่ปกติได้

ประโยชน์ที่ได้รับ

1. สามารถใช้กฎในการตรวจสอบและหลีกเลี่ยงข้อผิดพลาดที่เกิดขึ้นได้อย่างทันท่วงที

2. สามารถลดความผิดพลาดและผลเสียที่เกิดขึ้นจากการทำงานที่ผิดพลาดของระบบ

ขอบเขตและข้อจำกัด

1. นำเบย์เซียนเน็ตเวิร์คมาใช้การเรียนรู้เพื่อสกัดกฎสำหรับหลีกเลี่ยงข้อผิดพลาด

2. ในงานวิจัยจะใช้โปรแกรม RacingGame 1.0 ซึ่งเป็นโปรแกรมแข่งรถมาเป็นกรณีศึกษา เนื่องจากโปรแกรม RacingGame 1.0 มีโครงสร้างที่ไม่ซับซ้อน ทำให้มองเห็นการทำงานในส่วนต่างๆ ได้อย่างชัดเจน ซึ่งจะช่วยอธิบายกระบวนการสกัดกฎได้ดียิ่งขึ้น

3. ในการเรียนรู้จะใช้ข้อมูลทดสอบที่ได้จากการทำงาน (application-level data) โดยเก็บข้อมูลเหตุการณ์ที่มีพฤติกรรมที่ทำให้โปรแกรมเกิดข้อผิดพลาดซึ่งเป็นสถานะไม่ปกติ

4. ในงานวิจัยนี้ได้กำหนดให้รถชนขอบสนามเป็นข้อผิดพลาดในโปรแกรมซึ่งจะทำให้ระบบอยู่ในสถานะไม่ปกติ ดังนั้นในการเก็บข้อมูลเพื่อนำมาสกัดกฎจึงจะเก็บข้อมูลที่เกี่ยวข้องกับการชนขอบสนาม เนื่องจากสามารถตรวจสอบความครอบคลุมของกฎและทดสอบระบบได้ง่าย

5. เมื่อได้กฎแล้วจะใช้ Aspect Oriented Programming (AOP) ในการแทรกกฎ เพื่อใช้ในตรวจสอบและหลีกเลี่ยงข้อผิดพลาด

6. ในการตรวจสอบข้อผิดพลาดนั้นจะตรวจสอบข้อมูลเข้าที่ได้จากผู้ใช้และเปรียบเทียบกับกฎก่อนที่ระบบจะเริ่มต้นทำงานกับข้อมูลเข้านั้น หากข้อมูลเข้านั้นมีความน่าจะเป็นที่จะส่งผลให้ระบบเข้าสู่สถานะไม่ปกติ ส่วนหลีกเลี่ยงข้อผิดพลาดจะทำการปรับเปลี่ยนการทำงานบางอย่างเพื่อป้องกันไม่ให้ระบบเข้าสู่สถานะไม่ปกติ

การตรวจเอกสาร

ทฤษฎีที่เกี่ยวข้องกับงานวิจัย

การหลีกเลี่ยงข้อผิดพลาดเป็นวิธีการจัดการกับข้อผิดพลาดแบบหนึ่งที่จะช่วยเพิ่มความน่าเชื่อถือให้กับระบบ โดยการทำให้มั่นใจว่าได้แก้ไขข้อผิดพลาดก่อนที่มันจะเกิดขึ้น โดยการตรวจสอบในขณะที่ระบบกำลังทำงานอยู่ และเมื่อการตรวจสอบพบว่าระบบกำลังจะมีข้อผิดพลาดเกิดขึ้นก็จะแก้ไขโดยการหลีกเลี่ยงหรือปรับปรุงการทำงานบางอย่างที่อาจส่งผลให้มีข้อผิดพลาดเกิดขึ้นภายในระบบ ซึ่งจะช่วยลดความเสียหายที่เกิดขึ้นกับระบบ เนื่องจากว่าข้อผิดพลาดบางอย่างหากปล่อยให้เกิดขึ้นแล้ว ก็จะสร้างความเสียหายจนยากที่จะแก้ไขให้กลับมาเหมือนเดิม เช่น ไวรัสคอมพิวเตอร์ เป็นต้น

การพัฒนาระบบให้มีความสามารถในการหลีกเลี่ยงข้อผิดพลาดได้นั้น สามารถทำได้โดยใช้วิธีการซึ่งคล้ายกับระบบกู้คืนตนเอง แต่แตกต่างกันตรงการตัดสินใจในการแก้ไขข้อผิดพลาด โดยระบบที่มีการหลีกเลี่ยงข้อผิดพลาดนั้นจะป้องกันไม่ให้ข้อผิดพลาดเกิดขึ้น คือ ปรับปรุงการทำงานก่อนที่ข้อผิดพลาดนั้นจะเกิดขึ้น แต่ระบบกู้คืนตนเองนั้นจะแก้ไขข้อผิดพลาดหลังจากข้อผิดพลาดนั้นได้เกิดขึ้นแล้ว และเนื่องจากการพัฒนาระบบทั้งสองมีความคล้ายคลึงกัน ดังนั้นจึงจะอธิบายการทำงานโดยใช้ระบบกู้คืนตนเอง

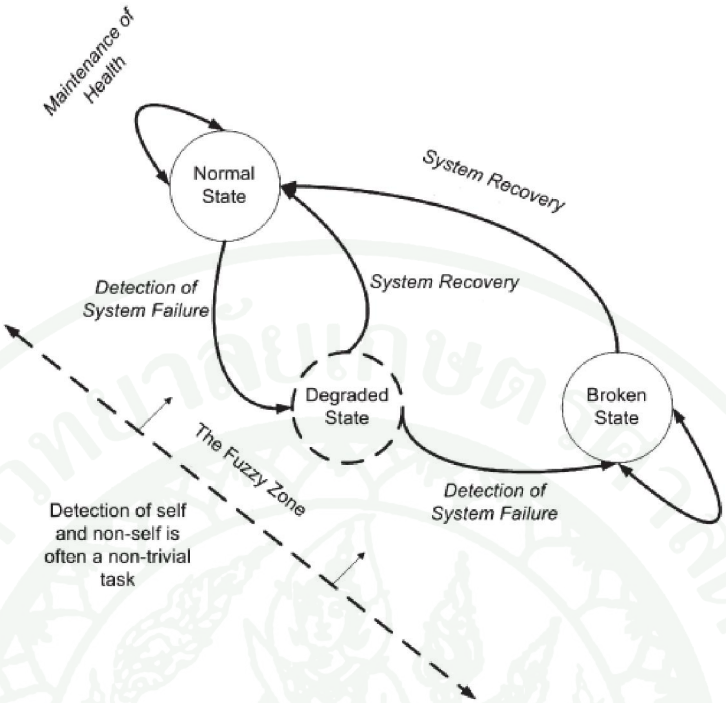
1. ระบบกู้คืนตนเอง (Self-healing)

การกู้คืนด้วยตนเองเป็นคุณสมบัติที่จะช่วยให้ระบบรู้ว่ามีการทำงานบางอย่างที่ไม่ถูกต้องเกิดขึ้นโดยไม่ต้องอาศัย หรืออาศัยการแทรกแซงของมนุษย์เพียงเล็กน้อยในการคืนตนเองกลับสู่สภาวะปกติ โดยระบบกู้คืนตนเองจะพยายาม “รักษา” ตัวเองโดยการฟื้นฟูตัวเองจากความผิดพลาดให้สามารถกลับมาทำงานได้ดังเดิม (Ghosh et al., 2007) โดยอาศัยแนวคิดที่มีลักษณะเหมือนระบบชีวภาพที่สามารถสมานแผลได้ โดยระบบจะสามารถกู้คืนตนเองจากสถานะที่ผิดปกติ (unhealthy) กลับไปที่สถานะที่ปกติ (healthy) ของการทำงานก่อนที่จะเกิดข้อผิดพลาดได้ โดยระบบกู้คืนตนเองจะสามารถจัดการกับข้อผิดพลาดและรักษาตัวเองจากความผิดพลาดหรือป้องกันตนเองจากการโจมตีที่เป็นอันตรายอย่างรุนแรงได้ ซึ่งจะช่วยเพิ่มความน่าเชื่อถือให้กับระบบ

1.1 สถานะปกติและสถานะไม่ปกติของระบบ

ก่อนที่ระบบจะสามารถฟื้นฟูตัวเองจากความผิดพลาดกลับคืนสู่สภาวะปกติของได้นั้น จำเป็นจะต้องมีการรับรู้ก่อนว่าอะไรคือสิ่งที่เรียกว่า “ปกติ” และ “ผิดปกติ” สำหรับระบบ โดยปกติแล้วนักพัฒนาระบบจะเป็นผู้กำหนดกฎเกณฑ์และคุณสมบัติที่สำคัญของระบบ โดยคุณสมบัติเหล่านี้มักจะถูกกำหนดไว้ในขั้นตอนการออกแบบระบบ โดยระบบที่มีการกำหนดเกณฑ์สำหรับระบบที่ “ปกติ” ได้อย่างชัดเจนและถูกต้อง จะทำให้การทำงานของระบบมีประสิทธิภาพเพิ่มมากขึ้น แต่เนื่องจากนักพัฒนาระบบมักจะได้รับข้อมูลของระบบและความต้องการของผู้ใช้ที่ไม่ชัดเจน จึงทำให้ระบบมักจะมีความเสี่ยงและเกิดความผิดพลาดได้ง่าย

โดยทั่วไปความแตกต่างระหว่าง “ปกติ” และ “ไม่ปกติ” มักจะไม่สามารถกำหนดได้อย่างชัดเจนเนื่องจากมีการเปลี่ยนแปลงไปมาระหว่างทั้งสองสถานะอย่างช้าๆ ดังนั้นจึงได้มีการกำหนดสถานะคลุมเครือขึ้นสำหรับเก็บพฤติกรรมที่ยังไม่สามารถแบ่งแยกได้เพื่อนำไปสู่การแบ่งแยกสถานะที่ชัดเจนต่อไป การกำหนดสถานะคลุมเครือจะขึ้นอยู่กับความต้องการของผู้ใช้และการเปลี่ยนแปลงของสิ่งแวดล้อมของระบบ ดังแสดงในภาพที่ 1



ภาพที่ 1 การเปลี่ยนแปลงสถานะระหว่างสถานะปกติ (Normal state) สถานะคลุมเครือ (Degraded state) และสถานะไม่ปกติ (Broken state)

ที่มา: Ghosh et al. (2007)

1.2 การดูแลระบบให้มีสถานะปกติ

ในการตรวจสอบหรือรักษาฟังก์ชันการทำงานให้เป็นปกติอยู่ตลอดนั้น จำเป็นต้องมีการตรวจสอบข้อผิดพลาดในการทำงานเป็นระยะๆ หรืออย่างต่อเนื่อง โดยวิธีการที่นำมาใช้จะต้องมีความสามารถที่จะเข้าใจการแยกกลุ่มระหว่าง “สถานะปกติ” และ “สถานะไม่ปกติ” และสามารถป้องกันและแก้ไขความผิดปกติที่อาจเกิดขึ้นได้ ตัวอย่างเช่น การกำหนดให้มีส่วนประกอบ (component) บางอย่างมีการทำงานที่ซ้ำซ้อนกัน เพื่อให้แน่ใจว่าส่วนประกอบที่ซ้ำซ้อนเหล่านี้จะมีเพียงพอต่อความต้องการใช้งาน ซึ่งวิธีนี้จะช่วยให้ส่วนประกอบอื่นที่ต้องการใช้การทำงานแบบเดียวกันที่เวลาเดียวกันสามารถใช้งานได้พร้อมกัน ในกรณีที่มีส่วนหนึ่งล้มเหลว มีการทำงานผิดปกติหรือถูกโจมตีนั้น ระบบจะสามารถนำส่วนประกอบอื่นที่มีการทำงานแบบเดียวกันมาใช้แทนได้ ซึ่งจะทำให้ระบบสามารถอยู่ในสถานะปกติต่อไปได้

1.3 การตรวจสอบและกู้คืนระบบจากสถานะไม่ปกติและกลับไปสู่สถานะปกติ

ในการตรวจสอบความล้มเหลวของระบบ ระบบจะต้องมีความรู้ที่เกี่ยวกับตัวเองและสภาพแวดล้อม เพื่อใช้ในการกู้คืนตนเอง ซึ่งความรู้เหล่านี้จะถูกเก็บอยู่ในที่เก็บข้อมูล (data repository) โดยระบบจะต้องตรวจสอบความล้มเหลวหรือค้นหาสิ่งที่เป็นอันตรายได้ในเวลาที่เหมาะสมและจะต้องมีความสามารถที่จะวัดระดับของความผิดปกติในระบบและประเมินว่าระบบควรได้รับการแทรกแซงจากโปรแกรมกู้คืนเมื่อไรและอย่างไร รวมทั้งให้ความสำคัญกับสถานะหลังจากเกิดความผิดพลาดหรือสถานะหลังจากเกิดการโจมตี

ในการแก้ไขข้อผิดพลาดเพื่อกลับเข้าสู่สถานะปกติอีกครั้งนั้น ส่วนประกอบอื่นๆ ที่ไม่เกี่ยวข้องกับข้อผิดพลาดไม่ควรมีการหยุดทำงานและส่วนประกอบทุกส่วนที่เกี่ยวข้องกับข้อผิดพลาดควรได้รับการกู้คืนการทำงาน ถึงแม้ว่าความผิดพลาดนั้นจะไม่ได้ส่งผลกระทบต่อส่วนประกอบเหล่านั้นก็ตาม

1.4 สถาปัตยกรรมของระบบกู้คืนตนเอง

โดยทั่วไประบบกู้คืนตนเองต้องการความเป็นอิสระในการทำงาน ความทนทานและความสามารถในการปรับตัวค่อนข้างมาก ดังนั้นโครงสร้างของระบบมักจะมีรูปแบบองค์ประกอบ (component-based model) หรือรูปแบบตัวแทน (agent-based model) โดยรูปแบบจะขึ้นอยู่กับสถาปัตยกรรมของระบบที่นำมาใช้ เพื่อให้สามารถทำหน้าที่ในการตรวจสอบการทำงานของส่วนประกอบภายในระบบหลักได้อย่างมีประสิทธิภาพ

ระบบกู้คืนตนเองมีการทำงานโดยทั่วไป ดังนี้ (1) สังเกตการทำงานที่มีความผิดปกติ (2) วิเคราะห์และตรวจสอบส่วนที่ผิดปกติ (3) หาวิธีแก้ไขความผิดปกติ (4) แก้ไขส่วนที่ผิดปกติ ดังแสดงในภาพที่ 2 และสามารถอธิบายการทำงานในส่วนต่างๆ ได้ดังในหัวข้อ 1.4.1 - 1.4.4 ตามลำดับ

1.4.1 การสังเกตระบบ (monitoring)

ส่วนนี้จะเกี่ยวข้องกับการเก็บข้อมูลในขณะที่ระบบหลักกำลังทำงานอยู่และสภาพแวดล้อมในการทำงาน โดยข้อมูลที่เก็บนั้นจะเป็นข้อมูลที่เกี่ยวข้องกับการตรวจสอบความผิดปกติ เพื่อส่งไปยังส่วนต่อไป

1.4.2 การวิเคราะห์และตัดสินใจเกี่ยวกับความผิดปกติ (analysis and decision)

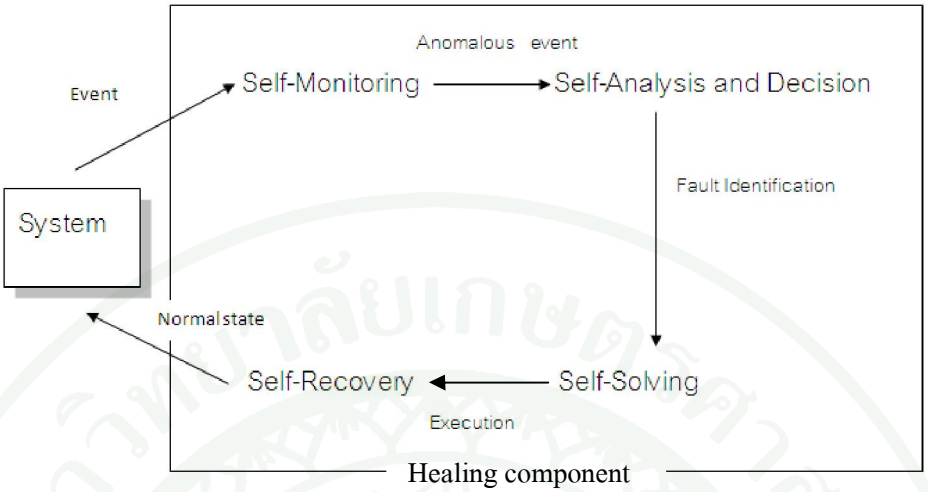
ส่วนนี้จะนำข้อมูลที่ได้จากการสังเกตมาวิเคราะห์และตรวจสอบกับข้อมูลที่อยู่ในที่เก็บข้อมูล (data repository) ซึ่งใช้ในการตรวจสอบข้อผิดพลาด ว่าข้อมูลที่เก็บมานั้นจะทำให้เกิดความผิดปกติขึ้นหรือไม่ หากว่าใช่ก็จะทำการส่งไปยังส่วนต่อไป

1.4.3 การแก้ปัญหา (solving)

เมื่อระบบตรวจพบความผิดปกติระบบจะเลือกกลไกการตอบสนองที่เป็นไปได้และเหมาะกับความผิดพลาดที่เกิดขึ้นสำหรับความผิดปกติแต่ละแบบโดยเฉพาะ เช่น ย้อนกลับไปยังจุดที่กำหนด (rollback to a checkpoint) ทำงานต่อไปโดยชดเชยสิ่งที่สูญเสียไป (roll forward with compensation) พยายามทำงานต่อไปโดยใช้ทรัพยากรเดิมหรือใช้ทรัพยากรใหม่ จบการทำงานที่มีความสำคัญน้อย ขอความช่วยเหลือจากภายนอก เป็นต้น

1.4.4 การกู้คืนตนเอง (recovery)

ในส่วนนี้จะทำการแก้ไขเพื่อให้ระบบกลับเข้าสู่สถานะปกติโดยใช้กลไกตามที่ได้เลือกมา โดยการตอบสนองเหล่านี้จะช่วยให้ระบบสามารถรักษาคุณสมบัติที่สำคัญของระบบหลักเอาไว้ได้ เช่น ความถูกต้องในการทำงาน คุณภาพของงาน ความปลอดภัยในการทำงาน ความสมบูรณ์ของการทำธุรกรรม เป็นต้น



ภาพที่ 2 ตัวอย่างการทำงานของระบบกู้คืนตนเอง

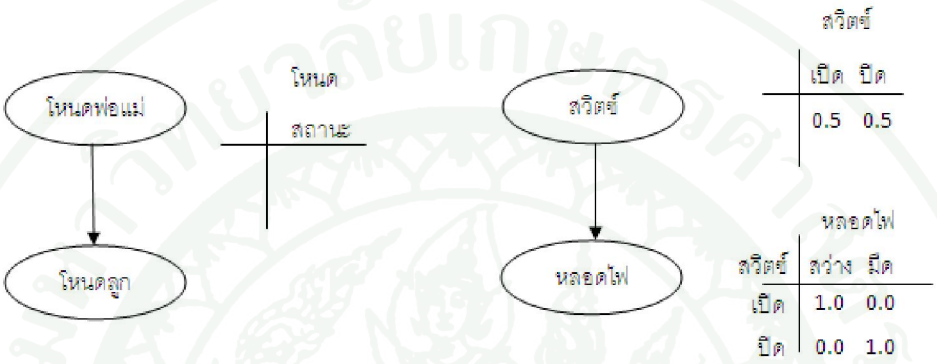
2. เบย์เซียนเน็ตเวิร์ค (Bayesian Network)

เบย์เซียนเน็ตเวิร์คเป็นแบบจำลองเชิงสาเหตุ (causal modeling) ที่ใช้อธิบายความสัมพันธ์ระหว่างเหตุการณ์ที่เราสนใจ สามารถแบ่งขั้นตอนการทำงานได้เป็น 2 ขั้นตอน คือ (1) การกำหนดรูปแบบ เป็นการสร้างกลุ่มของตัวแปรและการกำหนดสมมติฐานคร่าวๆ เพื่อใช้อธิบายความสัมพันธ์ของตัวแปร โดยการแสดงสาเหตุและผลกระทบของความสัมพันธ์ (2) การประมาณค่าเพื่อหาค่าความน่าจะเป็นแบบมีเงื่อนไขระหว่างตัวแปรโดยใช้ข้อเท็จจริงของเหตุการณ์ที่ได้จากแบบจำลองและสมมติฐานความไม่ขึ้นต่อกันระหว่างตัวแปร

2.1 โครงสร้างของเบย์เซียนเน็ตเวิร์ค

เบย์เซียนเน็ตเวิร์คประกอบด้วยกลุ่มตัวแปรสุ่มและค่าความน่าจะเป็นที่ขึ้นต่อกันอย่างมีเงื่อนไข โดยนำเสนอความรู้ที่มีความสัมพันธ์หรือเกี่ยวข้องกันในรูปของโครงข่ายกราฟไม่เป็นวงกลมที่มีทิศทาง (Directed Acyclic Graph) เพื่อแสดงความขึ้นต่อกันระหว่างตัวแปร และมีการแสดงค่าความน่าจะเป็นแบบมีเงื่อนไขระหว่างตัวแปรในตารางความน่าจะเป็นแบบมีเงื่อนไข โดยในกราฟจะมีโหนด (node) ใช้แทนตัวแปรสุ่มแต่ละตัวและเส้นเชื่อม (edge) ใช้แทนความสัมพันธ์ซึ่งอธิบายผลกระทบระหว่างตัวแปรที่เชื่อมต่อกัน ความสัมพันธ์ของแต่ละโหนดจะไม่วนกลับมา

หาโหนดเดิม ตัวอย่างเช่น ถ้าหัวลูกศรจากโหนด X ชี้ไปหาโหนด Y จะเรียกว่า โหนด X ว่าเป็นโหนดพ่อแม่ (parent) ของโหนด Y และแต่ละโหนดจะมีตารางแสดงสถานะและค่าความน่าจะเป็นของตัวแปรนั้นๆ อยู่ ซึ่งก็คือ ตารางความน่าจะเป็นแบบมีเงื่อนไข โดยโหนดลูกจะได้รับอิทธิพลโดยตรงจากโหนดพ่อแม่ เนื่องจากความน่าจะเป็นของโหนดลูกได้รับผลกระทบโดยตรงมาจากโหนดพ่อแม่ ดังแสดงในภาพที่ 3 (a)



(a) แสดงตัวอย่างกราฟของเบย์เซียน (b) แสดงความรู้เชิงคุณภาพ c) แสดงความรู้เชิงปริมาณ

ภาพที่ 3 แสดงการเปรียบเทียบโหนด เส้นเชื่อมและสถานะของตัวแปร

จากภาพที่ 3 หลอดไฟมีสถานะ คือ เปิดและปิด และมีโหนดพ่อแม่ คือ โหนดสวิตช์ โดยมีการแสดงความรู้อยู่ 2 แบบ คือ (1) ในเชิงคุณภาพ แสดงสถานะและความขึ้นต่อกันระหว่างตัวแปร ดังแสดงในภาพที่ 3 (b) และ (2) ในเชิงปริมาณ แสดงค่าความน่าจะเป็นระหว่างความสัมพันธ์ของตัวแปรที่ขึ้นต่อกัน (Luis M. de Campos, 2000) ดังแสดงในภาพที่ 3 (c)

2.2 ตารางความน่าจะเป็นแบบมีเงื่อนไข (Conditional Probability Table : CPT)

ตารางความน่าจะเป็นแบบมีเงื่อนไข เป็นตารางลักษณะ 2 มิติ ที่ใช้ในการเก็บค่าความน่าจะเป็น ตารางนี้ถูกสร้างขึ้นเพื่อใช้ในการแสดงค่าความน่าจะเป็นในแต่ละโหนด โดยอาศัยความน่าจะเป็นจากโหนดพ่อแม่ (prior probability) มาใช้ในการหาความน่าจะเป็นของโหนดปัจจุบัน ซึ่งเรียกว่า การกระจายของความน่าจะเป็นร่วม (joint probability distribution) ดังแสดงในภาพที่ 3 (c)

2.3 การเรียนรู้และทฤษฎีที่เกี่ยวข้องกับเบย์เซียนเน็ตเวิร์ค

การเรียนรู้เบย์เซียนเน็ตเวิร์คจะมีอยู่ 2 แบบ คือ เรียนรู้แบบไม่รู้โครงสร้าง (structure unknown) และการเรียนรู้แบบรู้โครงสร้าง (structure known) ดังนี้

1. การเรียนรู้แบบไม่รู้โครงสร้าง จะเป็นการเรียนรู้โดยที่ไม่รู้มาก่อนว่ากราฟจะมีโครงสร้างและการเชื่อมต่อโหนดอย่างไร โดยเบย์เซียนเน็ตเวิร์คจะพยายามเรียนรู้และคำนวณหาโครงสร้างกราฟและตารางความน่าจะเป็นที่ตรงกับข้อมูลฝึกสอนให้มากที่สุด โดยนำอัลกอริธึมในการค้นหา (search algorithm) มาใช้ในการสร้างกราฟ

2. การเรียนรู้แบบรู้โครงสร้างจะเป็นการเรียนรู้ที่รู้โครงสร้างและความสัมพันธ์ระหว่างโหนดอยู่แล้วแต่อาจมีข้อมูลฝึกสอนที่มีตัวแปรในบางเรคคอร์ดครบหรือไม่ก็ได้

เบย์เซียนเน็ตเวิร์คจะอาศัยทฤษฎีความน่าจะเป็นมาช่วยในการเรียนรู้ทั้งแบบรู้โครงสร้างและไม่รู้โครงสร้าง โดยจะสร้างแบบจำลองความน่าจะเป็นที่สอดคล้องกับข้อมูลที่มีอยู่ให้ได้มากที่สุด ซึ่งจะช่วยในการอธิบายความสัมพันธ์ในรูปความน่าจะเป็นของเหตุการณ์ที่อาจเกิดขึ้นอย่างมีเงื่อนไขโดยค้นหาผ่านการท่องไปในกราฟ และจะแสดงออกมาในรูปของตารางค่าความน่าจะเป็นแบบมีเงื่อนไข (Conditional Probability Table) ซึ่งมีทฤษฎีที่เกี่ยวข้องดังนี้

2.3.1 ทฤษฎีของเบย์ (Bayes' Rule) เป็นทฤษฎีทางด้านสถิติ โดยนำความน่าจะเป็นมาใช้ประเมินความไม่แน่นอนของเหตุการณ์ออกมาเป็นตัวเลข โดยจะแสดงในรูป P (A|B) ซึ่งก็คือความน่าจะเป็นของเหตุการณ์ที่จะเกิด (A) เมื่อมีอีกเหตุการณ์หนึ่ง (B) เกิดขึ้น สามารถคำนวณได้สมการต่อไปนี้

$$
\begin{aligned}
P(A) &= n(A)/n(s) \\
P(A \cap B) &= n(A \cap B)/n(s) = P(A,B) \\
P(A,B) &= P(A|B)*P(B) = P(B|A)*P(A) \\
P(A|B) &= P(B|A)*P(A)/P(B) \qquad (1)
\end{aligned}
$$

P (A|B) = ความน่าจะเป็นที่เหตุการณ์ A จะเกิดถ้าเหตุการณ์ B เกิดขึ้น

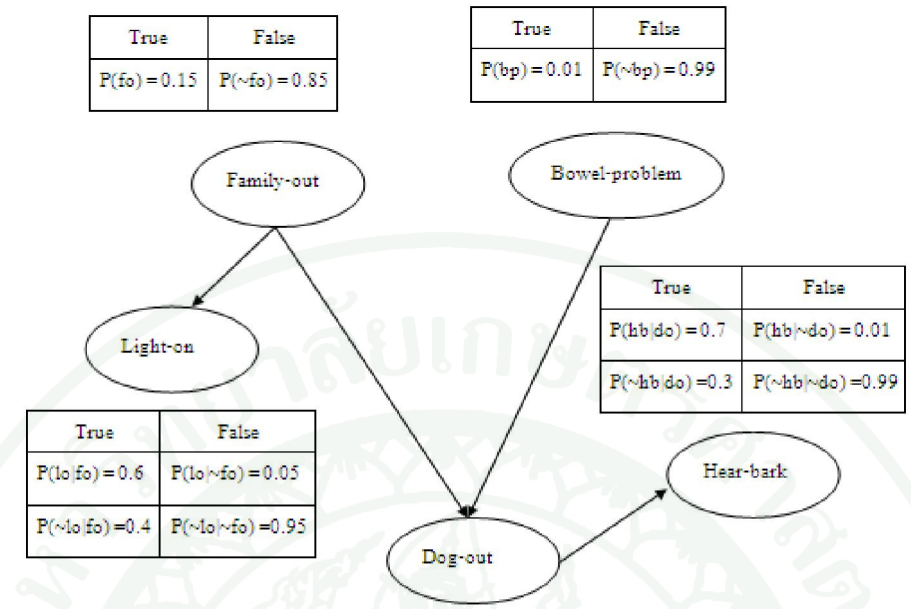
P (B|A) = ความน่าจะเป็นที่เหตุการณ์ B จะเกิดถ้าเหตุการณ์ A เกิดขึ้น

P (A) = ความน่าจะเป็นที่จะเกิดเหตุการณ์ A

P (B) = ความน่าจะเป็นที่จะเกิดเหตุการณ์ B

ตัวอย่างเช่น ในครอบครัวหนึ่ง โดยปกติ ภรรยาจะเปิดไฟรั้วบ้านก่อนที่จะออกจากบ้าน หรือบางครั้งภรรยาก็จะเปิดไฟรั้วบ้านทิ้งไว้เช่นกันเมื่อมีเพื่อนมาหา และที่บ้านมีสุนัขอยู่ตัวหนึ่ง เมื่อไม่มีคนอยู่ในบ้าน สุนัขก็จะถูกปล่อยไว้ในสนามหญ้า หรือบางครั้งเมื่อสุนัขมีปัญหาลำไส้มันก็จะถูกปล่อยไว้ในสนามหญ้าเช่นกัน และถ้าสุนัขอยู่ในสนามหญ้า สามีก็จะได้ยินเสียงมันเห่า จงหาค่าความน่าจะเป็นที่ภรรยาอยู่บ้านก่อนที่ประตูจะเปิดเมื่อเห็นไฟรั้วบ้านเปิดทิ้งไว้

กำหนดให้ เหตุการณ์ที่คนไม่อยู่บ้าน คือ Family out สุนัขมีปัญหาลำไส้ คือ Bowel-problem ไฟที่ประตูรั้วเปิด คือ Light-on สุนัขอยู่ที่สนามหญ้า คือ Dog-out และสามีได้ยินเสียงสุนัขเห่า คือ Hear-bark จากตัวอย่างสามารถสร้างกราฟได้ดังแสดงในภาพที่ 4



Family-out	True		False	
Bowel-problem	True	False	True	False
True	P(do|fo,bp) = 0.99	P(do|fo,~bp) = 0.90	P(do|~fo,bp) = 0.97	P(do|~fo,~bp) = 0.3
False	P(~do|fo,~bp) = 0.01	P(~do|~fo,~bp) = 0.1	P(~do|fo,~bp) =0.03	P(~do|~fo,~bp) =0.7

ภาพที่ 4 แสดงตัวอย่างเบย์เซียนเน็ตเวิร์ค

ที่มา: บุญเจริญ (2551)

วิธีแก้ปัญหา

P (fo) = ความน่าจะเป็นที่คนไม่อยู่บ้าน

P (~fo|lo) = ความน่าจะเป็นที่เห็นไฟรั้วบ้านเปิดทิ้งไว้แล้วภรรยาอยู่บ้าน

P (lo) = ความน่าจะเป็นที่เปิดไฟรั้วบ้านทิ้งไว้

P (lo|~fo) = ความน่าจะเป็นที่รู้ว่าภรรยาอยู่บ้านก่อนเห็นไฟรั้วบ้านเปิดทิ้งไว้

เนื่องจากเรายังไม่รู้ค่า P (lo) ดังนั้นเราจึงต้องหาค่าP (lo) จะได้ว่า

P (lo) = P (lo|fo)* P (fo)+ P (lo|~fo)* P (~fo)

= 0.6*0.15+0.05*0.85

= 0.09+0.04

= 0.13

$$P\ (\sim fo|lo) = \frac{P(lo|\sim fo)P(\sim fo)}{P(lo)}$$

$$P\ (\sim fo|lo) = \frac{0.05*0.85}{0.13} = 0.32$$

เพราะฉะนั้นความน่าจะเป็นที่เห็นไฟรั้วบ้านเปิดทิ้งไว้แล้วภรรยาอยู่บ้าน คือ 0.32

2.3.2 ความเป็นอิสระต่อกันอย่างมีเงื่อนไข (conditional independent) เป็นความน่าจะเป็นที่แสดงให้เห็นความเป็นอิสระกันระหว่างเหตุการณ์อย่างมีเงื่อนไข ซึ่งใช้ในการหาความน่าจะเป็นของตัวแปรในตารางความน่าจะเป็นแบบมีเงื่อนไข โดยเหตุการณ์ X และเหตุการณ์ Y จะถูกเรียกว่าเป็นอิสระต่อกันอย่างมีเงื่อนไข เมื่อรู้ว่าเหตุการณ์ Z ได้เกิดขึ้นแล้ว นั่นคือ เมื่อมีเหตุการณ์ Z เกิดขึ้น ความน่าจะเป็นแบบมีเงื่อนไขในการเกิดเหตุการณ์ X จะไม่เปลี่ยนแปลงไม่ว่าเหตุการณ์ Y จะเกิดขึ้นหรือไม่ ซึ่งสามารถเขียนเป็นสมการ ได้ดังนี้

$$P(X|Y, Z) = P(X|Z) \quad (2)$$

จากกราฟในภาพที่ 4 พิจารณาตัวแปรสุ่ม Family-out และ Hear-bark ว่าตัวแปรทั้งสองนี้เป็นอิสระกันหรือไม่ ถ้าดูจากเส้นเชื่อมในกราฟจะเห็นว่า โหนดทั้งสองขึ้นต่อกัน เนื่องจากว่า ถ้าไม่มีคนอยู่ในบ้าน สุนัขก็จะถูกปล่อยออกมา และเราจะได้ยินเสียงเห่าของมัน แต่ถ้าสมมติว่า เรารู้แน่ว่า สุนัขอยู่ในสนามหญ้าหลังบ้าน แล้วเราได้ยินเสียงเห่าของมัน การที่คนจะอยู่บ้านหรือไม่ก็ไม่สำคัญ ดังนั้นแสดงว่า Family-out และ Hear-bark เป็นอิสระต่อกันอย่างมีเงื่อนไขเมื่อรู้การเกิดของ Dog out เพราะเราจะได้ยินเสียงเห่าหรือไม่ขึ้นอยู่กับว่าสุนัขอยู่ในสนามหญ้าหรือไม่

2.3.3 ความน่าจะเป็นร่วม (Joint Probability) ในเบย์เซียนเน็ตเวิร์คตัวแปรแต่ละตัวจะมีความน่าจะเป็นเฉพาะที่อาจเป็นความน่าจะเป็นที่ได้จากความสัมพันธ์มากกว่าหนึ่งโหนด โดยความน่าจะเป็นที่มาจากตัวแปรมากกว่าหนึ่งตัวเรียกว่า “ความน่าจะเป็นร่วม” ซึ่งสามารถเขียนสมการของความน่าจะเป็นร่วมได้ ดังนี้

$$P(x_1,x_2,\ldots,x_n) = \prod_{i=1}^{n} P(X_i|Parents(X_i)) \quad (3)$$

เมื่อ Parents (X_i) คือ โหนดพ่อแม่โดยตรงของ X_i ถ้าต้องการหาความน่าจะเป็นร่วมที่ $x_1, x_2,\ldots, x_n$ เกิดขึ้นพร้อมกัน สามารถหาได้โดยนำความน่าจะเป็นของ x_1 คูณความน่าจะเป็นของ x_2 ไปเรื่อยๆจนถึง x_n จากภาพที่ 4 สามารถหาความน่าจะเป็นร่วมของกราฟได้ดังนี้

P(fo, bp, lo, do, hb) = P(fo) P(bp) P(lo|fo) P(do| fo, bp) P(hb|do)

3. การโปรแกรมเชิงลักษณะ (aspect-oriented programming)

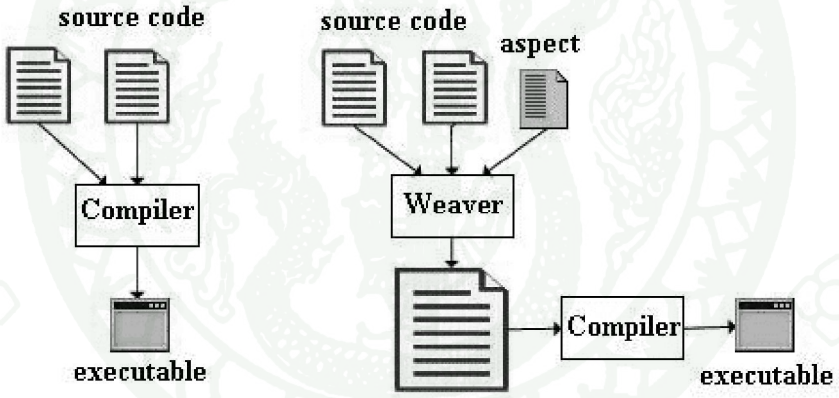
การโปรแกรมเชิงลักษณะ (aspect-oriented programming) เป็นรูปแบบหนึ่งของการเขียนโปรแกรมที่จะช่วยเพิ่มความเป็นโมดูลให้กับโปรแกรม โดยมีจุดประสงค์หลักเพื่อแก้ปัญหาที่ยากหรือแก้ไม่ได้ด้วยการโปรแกรมเชิงวัตถุ (object oriented programming) เนื่องจากคลาสส่วนใหญ่ในโปรแกรมเชิงวัตถุจะมีการทำงานแบบฟังก์ชันเฉพาะหรือมีความเป็นโมดูล คือ แต่ละคลาสมีฟังก์ชันการทำงานหลักที่แตกต่างกัน แต่บ่อยครั้งที่คลาสเหล่านี้สามารถใช้งานในส่วนของ common หรือ secondary requirements ร่วมกันกับคลาสอื่นได้ เนื่องจากมีโค้ดที่จำเป็นสำหรับฟังก์ชันการทำงานรองที่เหมือนกัน ดังนั้นเมื่อสร้างโค้ดเหล่านี้ไว้เพื่อใช้งานร่วมกันก็จะทำให้การเขียนโปรแกรมมีประสิทธิภาพเพิ่มมากขึ้น

3.1 การทำงานของโปรแกรมเชิงลักษณะ

แนวคิดของการโปรแกรมเชิงลักษณะ คือ การตัดหรือแยกโปรแกรมออกเป็นส่วน ๆ เรียกว่า คอนเซิร์นชนิดตัดขวาง (crosscutting concerns) จากนั้นจะทำการดึงโค้ดที่กระจายตัวในลักษณะคอนเซิร์นชนิดตัดขวาง (crosscutting concerns) มารวบไว้ด้วยกันเป็นหน่วยเรียกว่า aspects (ลักษณะ) โดย aspects จะห่อหุ้มพฤติกรรมการทำงานที่คล้ายกันในหลายๆ คลาสให้อยู่ในรูปของ

โมดูลที่สามารถนำกลับมาใช้ใหม่ (reusable modules) การทำงานของการโปรแกรมเชิงลักษณะ จะเริ่มต้นด้วยการทำงานของภาษาเชิงวัตถุ เช่น Java จากนั้น aspects จะจัดการแยกคอนเซิร์นชนิดตัดขวางในโค้ดออกจากกัน และเมื่อมีการทำงานถึงจุดตัดที่กำหนด โปรแกรมจะกระโดดไปทำคำสั่งของ aspects ที่กำหนดไว้ในโปรแกรม และด้วยคำสั่ง aspects นี้โปรแกรมจะสามารถเพิ่มเมท็อดหรือตัวแปรในคลาสได้ ซึ่งจะเห็นว่า aspects เดียวสามารถนำไปใช้งานได้ในหลายฟังก์ชัน ซึ่งจะช่วยเพิ่มความสามารถในการนำกลับมาใช้ใหม่ (reusability) และการดูแลรักษาโค้ด (maintainability)

ในการคอมไพล์โค้ดที่มีการใช้งาน aspects โค้ดจะถูกนำมารวมกันก่อนโดยใช้ aspects weaver ซึ่งจะได้โค้ดที่มี aspects รวมอยู่ในโค้ด จากนั้นก็จะถูกคอมไพล์และนำไปใช้งาน ดังแสดงในภาพที่ 5



ภาพที่ 5 อธิบายถึงขั้นตอนการสอดแทรกโค้ด

ที่มา: Manzalawy (2011)

3.2 โครงสร้างของโปรแกรมเชิงลักษณะ

โครงสร้างของโปรแกรมเชิงลักษณะ ประกอบไปด้วย 3 ส่วน ดังนี้

3.2.1 จุดตัด (Point cut) คือ จุดที่ใช้กำหนดตำแหน่งในการเรียกใช้งาน aspects โดยในการประกาศจุดตัดจะมีการเชื่อมโยงจุดตัดเข้ากับคำแนะนำ โดยคำแนะนำนี้สามารถทำงานก่อนหลังหรือรอบจุดร่วมได้

3.2.2 จุดร่วม (Join point) คือ จุดในโปรแกรมที่สามารถเพิ่มเติมพฤติกรรมหรือคำสั่งที่เป็นประโยชน์เข้ามาได้ โดยโปรแกรมเมอร์จะกำหนดตำแหน่งและการทำงานของจุดร่วม

3.2.3 คำแนะนำ (Advice) คือ การกำหนดการทำงานของ aspects และระบุการทำงานที่จุดตัดตามชนิดของคำแนะนำ ได้แก่

1. Before จะเริ่มทำงานก่อนที่จะมีการทำงานในส่วนของจุดร่วมตามที่ถูกระบุในจุดตัด
2. After จะเริ่มทำงานหลังจากที่มีการทำงานในส่วนของจุดร่วมเรียบร้อยแล้ว
3. Around จะเริ่มทำงานก่อนที่จุดร่วมที่ถูกระบุในจุดตัดนั้นๆ คือสามารถกำหนดให้ทำงานได้ทั้งก่อนและหลังจากการทำงานในจุดร่วม

```
after() : set() {
  Display.update();
}
```

ภาพที่ 6 แสดงตัวอย่างการเขียน aspect

จากภาพที่ 6 advice คือ Display.update() ถูกกำหนดให้มีการทำงานหลังจากทำงานในเมท็อด set() เรียบร้อยแล้ว ซึ่งเมท็อด set() ถูกกำหนดให้เป็นจุดตัดในโปรแกรม

งานวิจัยที่เกี่ยวข้อง

ในส่วนนี้จะนำเสนองานวิจัยที่เกี่ยวข้องในการทำวิจัย ซึ่งจะแบ่งออกเป็น 2 ส่วน คือ งานวิจัยที่เกี่ยวข้องกับการตรวจสอบข้อผิดพลาด (fault detection) และส่วนที่สอง คือ งานวิจัยที่เกี่ยวข้องกับการตรวจสอบข้อผิดพลาดในขณะที่ระบบกำลังทำงาน (runtime verification) ซึ่งมีรายละเอียดดังนี้

1. งานวิจัยที่เกี่ยวข้องกับการตรวจสอบข้อผิดพลาด (fault detection)

ในส่วนนี้จะนำเสนองานวิจัยที่เกี่ยวข้องกับการตรวจสอบและทำนายข้อผิดพลาดด้วยวิธีการต่างๆ จำนวนทั้งหมด 5 ชิ้น ซึ่งงาน 3 ชิ้นแรกจะใช้ข้อมูลที่ได้มาจากการทำงานของโปรแกรม (dynamic analysis) และอีก 2 ชิ้นจะใช้ข้อมูลที่ได้มาจากการวิเคราะห์การเขียนโปรแกรม (static analysis) ดังนี้

ในงานวิจัยของ Giljong Yoo และคณะ (Yoo et al., 2006) ได้นำเสนอแบบจำลองการทำนายแบบผสมผสานเพื่อใช้ในการปรับปรุงความน่าเชื่อถือของระบบกู้คืนตนเอง โดยแบบจำลองนี้จะนำมาใช้ในการทำนายเพื่อหาค่าความน่าจะเป็นของเหตุการณ์ที่เกิดขึ้นว่ามีโอกาสทำให้ระบบเกิดข้อผิดพลาดหรือไม่ ถ้าใช่ก็จะนำไปสู่การเลือกวิธีแก้ปัญหาที่เหมาะสมสำหรับเหตุการณ์นั้นๆ

ในงานวิจัยได้นำอัลกอริธึมได้แก่ ID3 algorithm, fuzzy Inference, fuzzy Neural Network และ Bayesian Network มาใช้ในการพิจารณาเหตุการณ์และเลือกวิธีการแก้ไข โดย

1. ID3 algorithm จะถูกใช้พิจารณาเหตุการณ์ที่ตรงกับข้อมูลในอดีตทั้งหมด
2. Fuzzy Inference ใช้กับเหตุการณ์ที่มีข้อมูลในอดีตไม่เพียงพอ
3. Neural Network ใช้กับเหตุการณ์ที่มีความใกล้เคียงกัน
4. Bayesian Network จะใช้สำหรับอนุมานและประเมินความน่าเชื่อถือของเหตุการณ์ที่มีความหลากหลายในระบบ

โครงสร้างของแบบจำลองประกอบด้วย 5 ส่วน ได้แก่ ส่วนสังเกตระบบ ส่วนประเมินระดับทรัพยากร ส่วนเลือกแบบจำลองในการทำนาย ส่วนทำงานตามขั้นตอนของแบบจำลองในการทำนาย และส่วนปรับปรุง ซึ่งมีการทำงานดังนี้

1. ส่วนสังเกตระบบ (system monitoring module) จะสังเกตระบบเป้าหมาย และเก็บข้อมูลที่ได้จากการทำงานของระบบในระดับระบบ เช่น เปอร์เซนต์การใช้งานหน่วยประมวลผลกลาง จำนวนผู้ใช้ ขนาดของไฟล์ เป็นต้น ดังแสดงในภาพที่ 7 เพื่อส่งไปยังส่วนถัดไป

CPU	RAM	Network Latency	...	Parameter			LogPath
				MaxClient	FileSize	...	
81%	64%	3.2sec	...	121/150	2.7MB	...	/home/gjyoo/logs/error_log
58%	40%	2.1sec	...	110/150	1.2MB	...	/home/gjyoo/logs/error_log
38%	77%	1.7sec	...	79/150	2.4MB	...	/home/gjyoo/logs/error_log
...	...	...	...	...	...	...	...

ภาพที่ 7 แสดงตัวอย่างข้อมูลที่ได้จากส่วนสังเกตระบบ

ที่มา: Yoo et al. (2006)

2. ส่วนประเมินระดับทรัพยากร (resource level evaluator) กำหนดระดับของทรัพยากรในระบบเป้าหมาย เพื่อใช้ในการทำงานของ fuzzy logic

3. ส่วนเลือกแบบจำลองในการทำนาย (prediction model selector) ทำหน้าที่ในการเลือกแบบจำลองในการทำนายที่เหมาะสมกับเหตุการณ์

4. ส่วนทำงานตามขั้นตอนของแบบจำลองในการทำนาย (prediction model algorithm executor) ทำงานตามแบบจำลองในการทำนาย

5. ส่วนปรับปรุง (model updater) ส่งผลการทำนายที่ได้ไปยังแบบจำลองในการทำนายทั้ง 4 ตัว เพื่อใช้ในการเรียนรู้และปรับปรุงข้อมูลในที่เก็บข้อมูล

ในงานวิจัยนี้ได้ประเมินประสิทธิภาพของแบบจำลองโดยเปรียบเทียบกับ ID3 algorithm พบว่าถ้าจำนวนข้อผิดพลาดมีน้อยแบบจำลองผสมผสานจะใช้เวลาในการทำงานใกล้เคียงกับการใช้ ID3 algorithm แต่ถ้าข้อผิดพลาดมีจำนวนมากแบบจำลองผสมผสานจะใช้เวลาน้อยกว่าและเมื่อใช้แบบจำลองผสมผสานจำนวนข้อผิดพลาดที่ไม่สามารถตรวจสอบได้มีน้อยกว่าการใช้ ID3 algorithm

งานวิจัยของ Shunshan Piao และคณะ (Piao et al., 2007) ได้นำเสนอวิธีการจัดการความผิดพลาดสำหรับระบบกู้คืนตนเองโดยใช้การเรียนรู้และวิเคราะห์ข้อมูลที่เวลาจริง (real time) เพื่อนำเสนอความสัมพันธ์ในรูปแบบความขึ้นต่อกันที่มีค่าความน่าจะเป็น (probabilistic dependency) ระหว่างปัจจัยต่างๆ ที่เกี่ยวข้องกับระบบ ดังแสดงในภาพที่ 8 ด้วยเบย์เซียนเน็ตเวิร์คและทำการอนุมานเพื่อแสดงให้เห็นถึงความผิดปกติของระบบที่จะเกิดขึ้นในรูปของความน่าจะเป็นของปัจจัยที่มีผลต่อปัญหานั้นๆ

ในงานวิจัยใช้ข้อมูลในระดับระบบ (system metrics) ที่สนใจซึ่งได้จากการทำงาน โดยแบ่งพารามิเตอร์ออกเป็น 2 ชุด คือ พารามิเตอร์ที่ได้จากการสังเกตจำนวน 6 ตัว (6 คอลัมน์แรก) และพารามิเตอร์ที่แสดงถึงปัญหาจำนวน 2 ตัว (2 คอลัมน์หลัง) ดังแสดงในภาพที่ 8 จากนั้นจึงนำพารามิเตอร์เหล่านี้ไปเรียนรู้ด้วยเบย์เซียนเน็ตเวิร์คเพื่อหาความสัมพันธ์ระหว่างพารามิเตอร์

CPU	RAM	Disk	Bandwidth	PacketVolume	ClientCount	Throughput	Responsetime
Medium	Low	Medium	Medium	High	Medium	Normal	Normal
Medium	Low	Low	Medium	High	Medium	Normal	Error
Low	High	High	Medium	High	High	Warning	Normal
Low	High	Medium	Medium	Medium	Medium	Normal	Error
High	Low	Medium	Medium	Medium	High	Normal	Normal
Medium	Low	High	High	Medium	High	Normal	Normal
Low	Low	Medium	High	High	Low	Warning	Normal
Low	Low	Medium	Medium	Medium	Low	Warning	Error
High	Low	Medium	Medium	Medium	High	Normal	Normal
Medium	Medium	Medium	Medium	Low	Low	Error	Normal
Medium	Low	High	Medium	Low	Medium	Normal	Warning
Medium	High	High	Medium	Low	Medium	Error	Normal

ภาพที่ 8 แสดงข้อมูลที่นำมาใช้ในการทดลอง

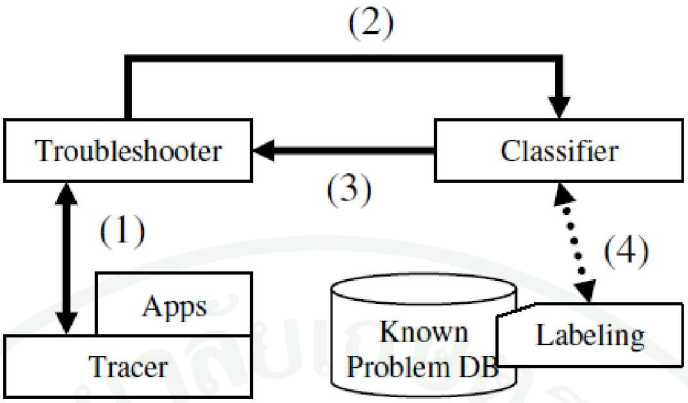
ที่มา: Piao et al. (2007)

ในงานวิจัยจะทดลองกับระบบคอมพิวเตอร์ที่มีระบบแสดงข้อมูลของเหตุการณ์แบบ real-time ซึ่งจะมีการสังเกตและคัดกรองเหตุการณ์ที่คาดว่าจะทำให้เกิดข้อผิดพลาด หลังจากนั้นจะมีการปรับปรุงแบบจำลองจากการเรียนรู้ผ่านทางข้อมูลที่ได้รวบรวมไว้และหาค่าความน่าจะเป็นของเหตุการณ์นั้น

การประเมินประสิทธิภาพแสดงให้เห็นว่าเมื่อมีข้อมูลในการเรียนรู้มากขึ้นจะใช้เวลาในการเรียนรู้นานแต่จำนวนข้อผิดพลาดลดลง

งานวิจัยของ Chun Yuan และคณะ (Yuan et al., 2006) นำเสนอการจัดการปัญหาอย่างอัตโนมัติ เนื่องจากเมื่อระบบเกิดข้อผิดพลาดขึ้นผู้ใช้มักจะทำการแก้ไขปัญหาด้วยตนเองโดยการค้นหาวิธีการแก้ปัญหาตามที่ต่างๆ เช่น หนังสือ อินเตอร์เน็ต เป็นต้น ซึ่งบางครั้งวิธีการแก้ปัญหาเหล่านี้ไม่ตรงกับปัญหาที่เกิดขึ้นหรือ ไม่สามารถแก้ไขปัญหาได้ เนื่องจากผู้ใช้ระบุปัญหาไม่ถูกต้อง ดังนั้น Yuan จึงใช้ข้อมูลพฤติกรรมของระบบ เช่น ร่องรอยเหตุการณ์ของระบบเพื่อที่จะหาความสัมพันธ์ในการแก้ปัญหาแทนการให้ผู้ใช้แก้ปัญหาเอง เพื่อลดการมีส่วนร่วมของผู้ใช้

ขั้นตอนการทำงานของ Chun Yuan และคณะ ดังแสดงในภาพที่ 9 โดยเมื่อผู้ใช้พบปัญหา ผู้ใช้สามารถเปิดใช้งานระบบ โดยตัวแก้ไขปัญหา (troubleshooter) จะทำงานที่ก่อให้เกิดปัญหาซ้ำอีกครั้งเพื่อเก็บรวบรวมอาการที่เกิดขึ้น ในขณะเดียวกันก็จะเริ่มต้นติดตามเหตุการณ์เพื่อเก็บรวบรวมลำดับเหตุการณ์ของระบบที่จะเกิดขึ้น เช่น การเรียกใช้ system call เป็นต้น จากนั้นข้อมูลจะถูกเก็บอยู่ใน trace log และจะถูกส่งไปยังตัวจำแนก (classifier) ซึ่งจะกรองและปรับปรุงรูปแบบของข้อมูลก่อนแล้วจึงวิเคราะห์ข้อมูลที่ได้รับมาและระบุสาเหตุโดยอาศัยความคล้ายคลึงกับเหตุการณ์ที่เก็บรวบรวมได้ก่อนหน้านี้ซึ่งอยู่ในที่เก็บข้อมูล (knowledge problem database) จากนั้นระบบจะส่งสาเหตุและวิธีการแก้ปัญหาที่สอดคล้องกันไปยังตัวแก้ไขปัญหา ซึ่งสามารถนำเสนอรายงานของคำแนะนำในการซ่อมแซมให้กับผู้ใช้หรือแม้กระทั่งแก้ไขปัญหาโดยอัตโนมัติได้



ภาพที่ 9 แสดงระบบแก้ไขปัญหาอัตโนมัติ

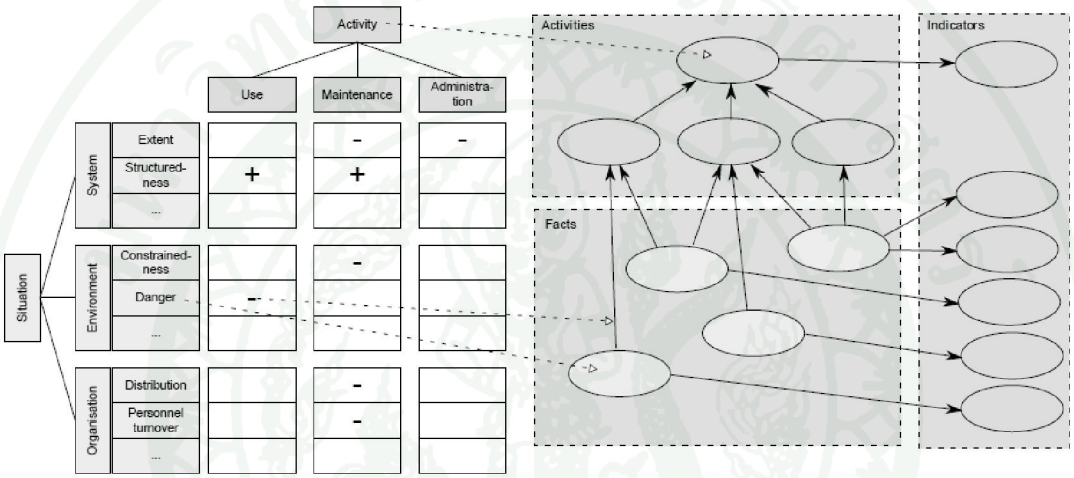
ที่มา: Yuan et al. (2006)

จากการทดลองพบว่าการแก้ไขปัญหามีความถูกต้องเพิ่มมากขึ้นเมื่อจำนวนข้อมูลในฐานข้อมูลมีมากขึ้น

งานวิจัยของ Stefan Wagner (Wagner, 2009) นำเสนอการสร้างแบบจำลองการวัดคุณภาพจากเบย์เซียนเน็ตเวิร์คเพื่อการประเมินและทำนายคุณภาพของโปรแกรมประยุกต์โดยใช้ข้อมูลจาก activity-based quality model ซึ่ง activity-based quality model เป็นแบบจำลองที่ใช้ในการอธิบายข้อกำหนดด้านคุณภาพและอธิบายความสัมพันธ์ระหว่างเทคนิคที่ใช้ในการประเมินคุณภาพกับข้อกำหนดด้านคุณภาพ เพื่อใช้ในการประเมินคุณภาพของโครงสร้างโปรแกรมและสร้างรายการตรวจสอบที่เกี่ยวข้องกันได้ เนื่องจาก activity-based quality model มีข้อจำกัด คือ หากโปรแกรมที่นำมาใช้เป็นโปรแกรมที่ใหญ่และมีความซับซ้อนมาก การสร้าง activity-based quality model ของโปรแกรมให้ครบถ้วนและถูกต้องนั้นสามารถทำได้ยาก ดังนั้น Wagner จึงได้นำเบย์เซียนเน็ตเวิร์คมาใช้เพื่อช่วยในการวัดคุณภาพโดยใช้ข้อมูลจาก activity-based quality model

ในงานวิจัยได้นำข้อมูลจาก activity-based quality model ซึ่งประกอบด้วย activity ซึ่งเป็นสิ่งที่ส่งผลกระทบต่อคุณภาพของโปรแกรมและ fact คือ ความสัมพันธ์ระหว่าง activity และคุณสมบัติของ activity โดยทั้งสองถูกนำมาแปลงให้อยู่ในรูปโหนดในเบย์เซียนเน็ตเวิร์ค โดยเริ่มจากการกำหนดเป้าหมายในการวัดหรือทำนายประสิทธิภาพของ activity เช่น แนวโน้มของ

ผลกระทบ จำนวนการโจมตีที่เป็นอันตรายในรอบ 1 ปี เป็นต้น จากนั้นกำหนด indicator ที่จะใช้วัด เช่น จำนวนบรรทัดในโค้ด (LOC) หรือ ตัวชี้วัดของ Halstead (Halstead metrics) เป็นต้น โดยกำหนดให้เป็นโหนดเช่นกัน เมื่อกำหนดข้อมูลต่างๆ เช่น activity, fact, indicator ให้เป็นโหนดเรียบร้อยแล้ว จากนั้นจะจับคู่ indicator กับ activity และ fact โดยแสดงในรูปเครือข่ายกราฟของเบย์เซียนเน็ตเวิร์ค และสร้างตารางความน่าจะเป็นสำหรับแต่ละโหนดเพื่อแสดงถึงผลกระทบของความสัมพันธ์ระหว่างโหนดโดยใช้ข้อมูลจากผู้เชี่ยวชาญ ดังแสดงในภาพที่ 10



ภาพที่ 10 แสดงการแปลงสมาชิกใน quality model ไปเป็นโหนดในเบย์เซียนเน็ตเวิร์ค

ที่มา: Wagner (2009)

งานวิจัยของ Yuyang Liu และคณะ (Liu et al., 2008) นำเสนอการทำนายแนวโน้มในการเกิดข้อผิดพลาด (failure) โดยใช้เบย์เซียนเน็ตเวิร์ค โดยเลือกใช้พารามิเตอร์ที่ได้จาก ตัววัดประสิทธิภาพ (software metric) จากนั้นจะนำผลที่ได้มาเปรียบเทียบกับวิธี logistic regression และ Naive Bayes

พารามิเตอร์ที่ใช้ได้มาจากการประเมินโครงสร้างของ object –oriented code โดยเลือกตัวชี้วัดที่เข้าใจง่าย ดังนี้

1. Fan OUT (FOUT) : จำนวนของ method calls
2. Method Lines of Code (MLOC) : จำนวนบรรทัดของ code ภายใน method bodies ไม่รวมบรรทัดว่างและ comments
3. Total Lines of Code (TLOC) : จำนวนบรรทัดของ code ทั้งหมดในไฟล์ที่เลือก
4. Number Of Methods (NOM) : จำนวน methods ถูกเรียกใช้งานในไฟล์ที่กำหนด
5. Number Of Attributes (NOF) : จำนวน attributes ในไฟล์ที่กำหนด
6. Number Of Static Methods (NSM) : จำนวน static methods ในไฟล์ที่กำหนด
7. Number of Static Attributes (NSF) : จำนวน static attributes ในไฟล์ที่กำหนด
8. McCabe cyclomatic complexity (VG) : นับจำนวน flows ที่เกิดในแต่ละครั้งในไฟล์ที่กำหนด
9. Number of Parameters (PAR) : จำนวน parameters ใน scope ที่กำหนด
10. Nested Block Depth (NBD) : ความลึกของชั้น blocks ของ code ในไฟล์ที่กำหนด
11. Number of Interfaces (NOI) : จำนวนของ interfaces ทั้งหมดในไฟล์ที่กำหนด
12. Number of Classes (NOT) : จำนวนของ classes ทั้งหมดในไฟล์ที่กำหนด
13. Pre-release Defects (pre) : จำนวนของ non-trivial defects ที่ถูกรายงานภายใน 6 เดือนล่าสุดก่อนวางจำหน่าย

ผลลัพธ์ คือ

Post-release Defects (post): จำนวนของ non-trivial defects ที่ถูกรายงานภายใน 6 เดือนแรกหลังวางจำหน่าย

จากผลการทดลองพบว่าแบบจำลองนี้ให้ค่า False Negative เป็น 0.0522 และ True positive เป็น 0.6103 และให้ค่าความถูกต้องเป็น 0.8651 เมื่อนำไปเปรียบเทียบกับวิธีการอื่นจะเห็นว่าแบบจำลองนี้ทำนายแนวโน้มการเกิดข้อผิดพลาดได้ถูกต้องดังแสดงในตารางที่ 1

ตารางที่ 1 แสดงการเปรียบเทียบประสิทธิภาพกับวิธีอื่น

	FN rate	TN rate	Accuracy
logistic regression	0.337	0.621	0.711
Naive Bayes	0.2575	0.3327	0.7310
Our approach	0.0522	0.6103	0.8651

ที่มา: Liu et al. (2008)

2. งานวิจัยที่เกี่ยวข้องกับการตรวจสอบข้อผิดพลาดในขณะที่ระบบกำลังทำงาน (runtime verification)

งานวิจัยของ Insup Lee และคณะ (Lee et al., 1998) นำเสนอ framework ในการตรวจสอบข้อผิดพลาดในขณะที่ระบบกำลังทำงานอยู่เพื่อเพิ่มความมั่นใจให้กับผู้ใช้ว่าระบบสามารถที่จะทำงานได้อย่างถูกต้องตรงตามความต้องการของผู้ใช้ โดยทำการตรวจสอบระบบตามข้อมูลในขั้นตอนของการออกแบบ (requirement specification)

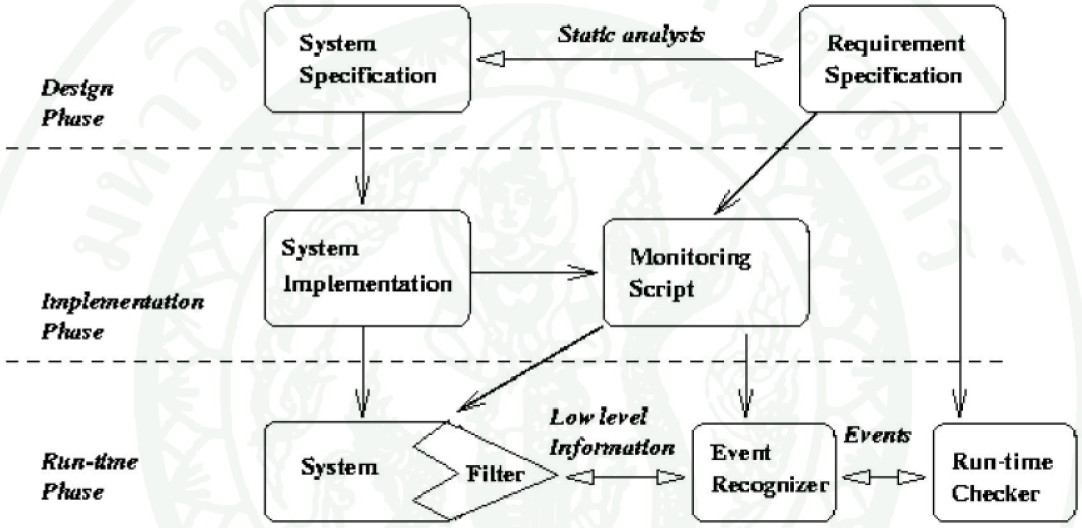
ในงานวิจัยจะสร้าง framework โดยมีพื้นฐานมาจากสถาปัตยกรรมแบบสังเกตและตรวจสอบ (monitoring and checking architecture: MaC) วิธีการนี้จะตรวจสอบเฉพาะการทำงานปัจจุบันของระบบ ซึ่ง monitoring and checking architecture ประกอบไปด้วย

1. filter เป็นส่วนที่ถูกสอดแทรกลงในโปรแกรมที่จะได้รับการตรวจสอบเพื่อสกัดเหตุการณ์ที่เกี่ยวข้องกับการตรวจสอบ เช่น ข้อมูลตัวแปร ไปยัง event recognizer

2. event recognizer จะรับข้อมูลจาก filter มาแปลงให้อยู่ใน high-level เพื่อให้ run-time checker นำข้อมูล high-level นี้ไปใช้ในการตรวจสอบได้

3. run-time checker จะตรวจสอบความถูกต้องของระบบโดยใช้ข้อมูลที่ได้รับมาจาก event recognizer และตรวจสอบลำดับของเหตุการณ์ตาม requirement specification

การทำงานของ framework ดังแสดงในภาพที่ 11 จะเริ่มตั้งแต่ขั้นตอนการออกแบบ (design phase) โดยมีการกำหนด requirement specification จากนั้นในขั้นตอนการพัฒนา (implement phase) จะมีการเขียนโค้ดใน monitoring script เพื่อใช้ในการอธิบายเหตุการณ์ที่ต้องการเฝ้าสังเกต และในขั้นตอนการทำงาน (run-time phase) จะมีการแทรกโค้ดเพื่อสังเกตและตรวจสอบ โดยเมื่อ run-time checker พิจารณาแล้วพบว่าระบบอยู่ในสถานะไม่ปกติก็จะมีการแจ้งเตือนเพื่อให้ผู้ใช้ทราบ

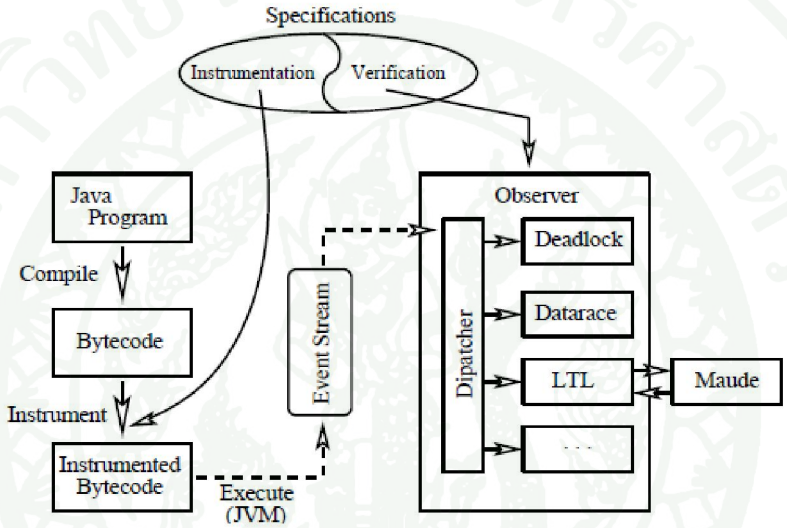


ภาพที่ 11 แสดงโครงสร้างของ monitoring and checking architecture: MaC

ที่มา: Lee et al. (1998)

งานวิจัยของ Klaus Havelund และ Grigore Rosu (Havelund และ Rosu., 2001) นำเสนอการตรวจสอบความถูกต้องในขณะที่ระบบกำลังทำงานด้วย Java PathExplorer (JPaX) เพื่อใช้ในการตรวจสอบการทำงานของโปรแกรมจาวา โดยการสกัดเหตุการณ์จากการทำงานของโปรแกรมและวิเคราะห์เหตุการณ์ขณะที่โปรแกรมกำลังทำงาน โดยตรวจสอบ 2 อย่าง คือ logic based monitoring และ error pattern analysis โดย logic based monitoring จะตรวจสอบและติดตามการทำงานโดยเปรียบเทียบกับ requirement specification ซึ่งเขียนอยู่ในรูป high-level logic ส่วน error pattern analysis จะวิเคราะห์เหตุการณ์ในการทำงานโดยใช้อัลกอริทึมที่สามารถระบุรูปแบบในการเกิดข้อผิดพลาดจากการเขียนโปรแกรมได้

ภาพที่ 12 แสดงโครงสร้างการทำงานของ Java PathExplorer โดยมีข้อมูลเข้าสู่ระบบประกอบด้วย 2 ส่วน คือ โปรแกรมจาวาที่อยู่ในรูปของ byte code และ specification script ซึ่งโปรแกรมจาวาที่อยู่ในรูปของ byte code คือ โปรแกรมที่จะนำมาใช้ในการตรวจสอบ และ specification script จะใช้ในการกำหนดขอบเขตในการตรวจสอบการทำงาน โดยจะแสดงผลในรูปของข้อความแจ้งเตือนบนหน้าจอ การทำงานจะเริ่มจากการสังเกตและดักจับเหตุการณ์จากนั้นจะส่งข้อมูลไปยังโปรเซสสังเกตการณ์เพื่อตรวจสอบการทำงานตาม specification script



ภาพที่ 12 แสดงโครงสร้างของ Java PathExplorer

ที่มา: Havelund และ Rosu. (2001)

ตารางที่ 2 สรุปงานวิจัยที่เกี่ยวข้องเรียงตามประเภทของงานวิจัยและปีที่พิมพ์

ที่	ปี	ชื่องานวิจัย	สิ่งที่นำเสนอ	ผลการทดลอง
1	2006	Hybrid Prediction Model for Improving Reliability in Self-Healing System (Yoo et al., 2006)	นำเสนอแบบจำลองการทำนายแบบผสมผสานเพื่อใช้ในการปรับปรุงความน่าเชื่อถือของระบบกู้คืนตนเอง โดยแบบจำลองนี้จะนำมาใช้ในการทำนายและเลือกวิธีการแก้ปัญหาซึ่งจะขึ้นอยู่กับเหตุการณ์ที่เกิดขึ้นในระบบ	แบบจำลองผสมผสานจะใช้เวลาในการทำงานน้อยกว่าและจำนวนข้อผิดพลาดที่บันทึกได้มีน้อยกว่าเมื่อเทียบกับการใช้ ID3 algorithm
2	2006	Automated Known Problem Diagnosis with Event Traces (Yuan et al., 2006)	นำเสนอการใช้พฤติกรรมของระบบ เช่น ร่องรอยเหตุการณ์ในระบบ มาใช้ในการหาความสัมพันธ์เพื่อแก้ปัญหาอย่างอัตโนมัติ เพื่อลดการมีส่วนร่วมของผู้ใช้	สามารถนำมาใช้ในการแก้ไขปัญหาได้แต่ความถูกต้องในการแก้ไขปัญหาจะขึ้นอยู่กับจำนวนข้อมูลในฐานข้อมูล
3	2007	Root Cause Analysis and Proactive Problem Prediction for Self-Healing (Piao et al.,2007)	นำเสนอวิธีการจัดการความผิดพลาดสำหรับระบบกู้คืนตนเองโดยใช้การเรียนรู้และวิเคราะห์ข้อมูลที่เวลาจริง (real time) เพื่อนำเสนอความสัมพันธ์ในรูปแบบความน่าจะเป็นอย่างมีเงื่อนไข ระหว่างปัจจัยต่างๆ ด้วยเบย์เซียนเน็ตเวิร์ค และอนุมานเพื่อแสดงให้เห็นถึงความผิดปกติที่จะเกิดขึ้น	การประเมินประสิทธิภาพแสดงให้เห็นว่าเมื่อมีข้อมูลในการเรียนรู้มากขึ้นจะใช้เวลาในการเรียนรู้นานแต่จำนวนข้อผิดพลาดลดลง

ตารางที่ 2 (ต่อ)

ที่	ปี	ชื่องานวิจัย	สิ่งที่นำเสนอ	ผลการทดลอง
4	2008	Predict Software Failure-prone by Learning Bayesian Network (Lui et al., 2008)	นำเสนอการทำนายแนวโน้มในการเกิดข้อผิดพลาด (failure) โดยใช้พารามิเตอร์ที่ได้มาจากการประเมินโครงสร้างของ code โดยใช้เบย์เซียนเน็ตเวิร์ค	แบบจำลองสามารถทำนายแนวโน้มการเกิดข้อผิดพลาดได้ถูกต้องมากกว่าเมื่อเปรียบเทียบกับวิธีการอื่น
5	2009	Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models (Wanger, 2009)	นำเสนอการสร้างแบบจำลองการวัดคุณภาพจากเบย์เซียนเน็ตเวิร์คเพื่อการประเมินและทำนายคุณภาพของโปรแกรมประยุกต์โดยใช้ข้อมูลจาก Activity-based quality model	ผลที่ได้จากการทดลองยังไม่เป็นที่น่าพอใจมากนักเนื่องจากยังมีผลกระทบจากปัจจัยอื่นๆที่ยังไม่ได้พิจารณาเนื่องจากไม่มีข้อมูลที่เพียงพอ
6	1998	A Monitoring and Checking Framework for Run-time Correctness Assurance (Lee et al., 1998)	นำเสนอ Framework ในการตรวจสอบข้อผิดพลาดในขณะที่ระบบกำลังทำงานอยู่ โดยตรวจสอบระบบตาม requirement specification	สามารถตรวจสอบข้อผิดพลาดในขณะที่ระบบกำลังทำงานอยู่ได้และเมื่อพบข้อผิดพลาดจะทำการแจ้งเตือน
7	2001	Java PathExplorer - A Runtime Verification Tool (Havelund and Rosu., 2001)	นำเสนอการตรวจสอบความถูกต้องในขณะที่ระบบกำลังทำงานด้วย Java PathExplorer (JPaX) เพื่อใช้ในการตรวจสอบการทำงานของโปรแกรมจาวา	สามารถตรวจสอบการทำงานและแก้ไขได้ก่อนที่จะมีข้อผิดพลาดเกิดขึ้นและทำการแจ้งเตือน

อุปกรณ์และวิธีการ

สมมติฐาน

กฎที่ได้จากการเรียนรู้ของเบย์เซียนเน็ตเวิร์คสามารถนำไปใช้ในการดักจับเหตุการณ์ที่จะส่งผลให้เกิดข้อผิดพลาดในโปรแกรมและสามารถหลีกเลี่ยงเหตุการณ์ที่ส่งผลให้เกิดข้อผิดพลาดในโปรแกรมซึ่งตรงกับกฎได้อย่างถูกต้อง

อุปกรณ์

1. ฮาร์ดแวร์

1.1 เครื่องคอมพิวเตอร์ Intel Celeron® CPU 2.20 GHz 2.19 GHz, 752 MB of RAM Hard disk 40 GB

2. ซอฟต์แวร์

2.1 Microsoft Window XP Professional Version 2000 Service Pack 3

2.2 Eclipse -java-galileo-win32

2.3 Java Development Kit (JDK) jdk-6u23-windows-i586

2.4 Racing Game 1.0

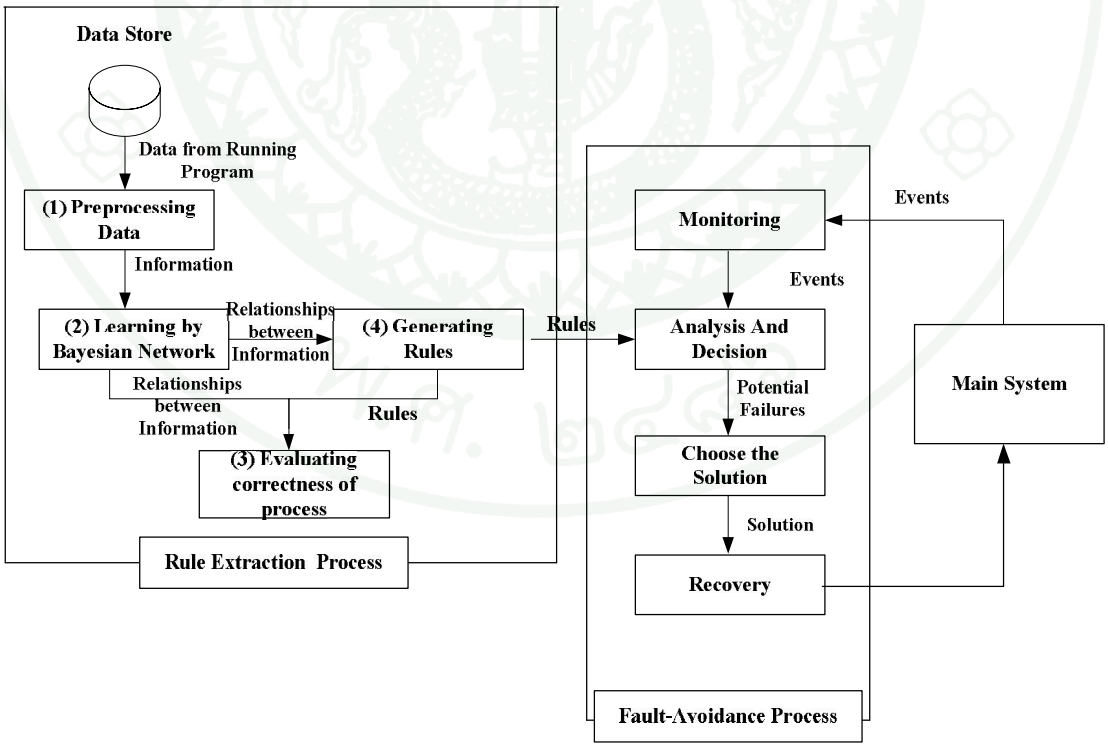
2.5 WEKA

2.6 AspectJ 1.5.4

วิธีการ

ในงานวิจัยได้พัฒนาระบบหลีกเลี่ยงข้อผิดพลาด โดยใช้ข้อมูลพฤติกรรมที่ทำให้เกิดข้อผิดพลาดและไม่ทำให้เกิดข้อผิดพลาดขึ้นภายในระบบมาใช้ในการเรียนรู้และสกัดออกมาเป็นกฎ โดยการเก็บรวบรวมข้อมูลพฤติกรรมและคัดกรองข้อมูลที่ได้จากการทำงานจริงของระบบ ซึ่งประกอบไปด้วยข้อมูลที่ได้รับจากผู้ใช้และข้อมูลการทำงานภายในระบบซึ่งมีส่วนทำให้เกิดข้อผิดพลาดในระบบ จากนั้นจึงนำไปใช้ในการเรียนรู้ของเบย์เซียนเน็ตเวิร์คเพื่อหาความสัมพันธ์ระหว่างข้อผิดพลาดและพฤติกรรมที่ทำให้เกิดข้อผิดพลาด และนำความสัมพันธ์ที่ได้มาสกัดเป็นกฎเพื่อใช้ในการหลีกเลี่ยงการเกิดข้อผิดพลาดที่ผู้ใช้สามารถมองเห็นได้ จากนั้นจะนำกฎที่ได้มาใช้ในการตรวจสอบและหลีกเลี่ยงข้อผิดพลาดที่เกิดขึ้นในระบบโดยอาศัยการทำงานของ AspectJ และได้ทำการทดลองเพื่อแสดงถึงความเป็นไปได้ของระบบหลีกเลี่ยงข้อผิดพลาดด้วยกรณีศึกษา

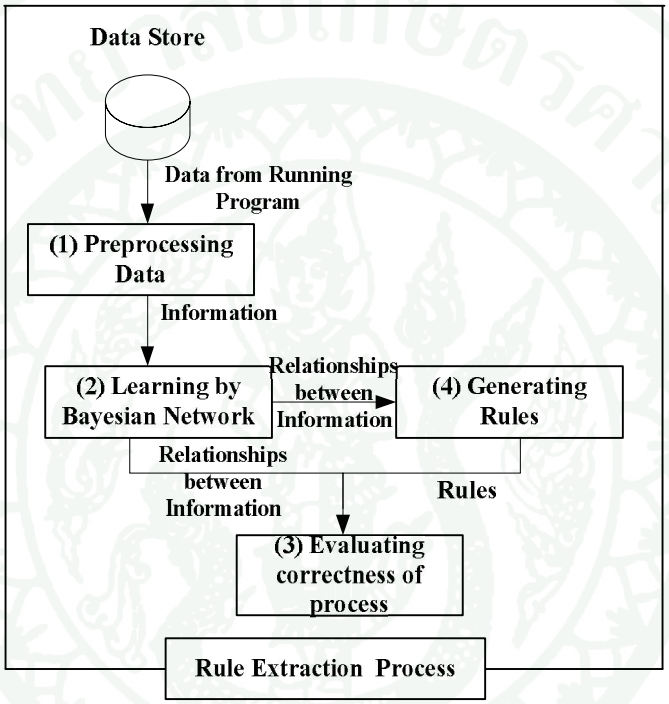
โดยงานวิจัยจะแบ่งการทำงานออกเป็น 2 ส่วน คือ ส่วนของการสกัดกฎและส่วนของการหลีกเลี่ยงข้อผิดพลาด ดังแสดงในภาพที่ 13 ซึ่งจะอธิบายในหัวข้อ 1 และ 2 ตามลำดับ



ภาพที่ 13 แสดงการทำงานของระบบหลีกเลี่ยงข้อผิดพลาด

1. ส่วนของการสกัดกฎ

ในส่วนของการสกัดกฎจะเป็นส่วนที่ใช้ในการรวบรวมข้อมูลและหาความสัมพันธ์ระหว่างข้อผิดพลาดและพฤติกรรมที่ทำให้เกิดข้อผิดพลาดโดยใช้เบย์เซียนเน็ตเวิร์ค และนำความสัมพันธ์เหล่านั้นไปใช้การสกัดกฎ ซึ่งสามารถแบ่งการทำงานได้เป็น 4 ขั้นตอน ดังแสดงในภาพที่ 14



ภาพที่ 14 แสดงขั้นตอนในการทำงานของส่วนของการสกัดกฎ

1.1 ขั้นตอนการเก็บและคัดกรองข้อมูล

ในส่วนนี้จะเป็นการเก็บและคัดกรองข้อมูล โดยข้อมูลที่เก็บมานั้นจะเป็นเหตุการณ์ที่เกิดขึ้นในระบบซึ่งมีความเกี่ยวข้องและส่งผลให้มีความผิดพลาดเกิดขึ้นในระบบ และจะมีการกำหนดตัวแปรขึ้นมาอีกหนึ่งตัวซึ่งเรียกว่า ผลเฉลย ซึ่งจะแสดงให้เห็นว่าเหตุการณ์นั้นจะทำให้ระบบเกิดข้อผิดพลาดหรือไม่ เพื่อใช้ในการเรียนรู้และตรวจสอบความถูกต้องของข้อมูลในขั้นตอนของการวัดความถูกต้องของกฎ โดยผู้วิจัยจะเก็บข้อมูลทั้งในส่วนที่ระบบมีสถานะของการทำงานปกติและผิดปกติเพื่อใช้เป็นข้อมูลฝึกสอน (training dataset) เมื่อได้ข้อมูลฝึกสอนมาแล้วจะทำการคัดกรองข้อมูล จากนั้นจะเข้าสู่กระบวนการเรียนรู้และสกัดกฎต่อไป

ในการเก็บข้อมูลนั้นจะต้องทำให้แน่ใจว่าได้เก็บข้อมูลอย่างครบถ้วนและครอบคลุมการทำงานให้ได้มากที่สุด เนื่องจากว่าข้อมูลฝึกสอนเหล่านี้จะส่งผลต่อความถูกต้องในการสกัดกฎ

1.2 ขั้นตอนการเรียนรู้

ในการเรียนรู้นั้นผู้วิจัยจะใช้ตัวจำแนกที่มีชื่อว่าเบย์เซียนเน็ตเวิร์คซึ่งมีความสามารถในการจำแนกและค้นหาความสัมพันธ์ระหว่างตัวแปรที่สนใจ และแสดงผลออกมาในรูปของกราฟแสดงความสัมพันธ์และตารางค่าความน่าจะเป็นระหว่างตัวแปร โดยการทำงานของเบย์เซียนเน็ตเวิร์คนั้นจะเริ่มจากการเรียนรู้ข้อมูลฝึกสอนที่มีผลเฉลยเพื่อหาสาเหตุและความสัมพันธ์ที่ทำให้เกิดข้อผิดพลาดโดยใช้เครื่องมือ WEKA (University of Waikato, 2011) ความสัมพันธ์ที่ได้จะถูกนำไปใช้ในการสกัดกฎต่อไป

1.3 ขั้นตอนการวัดประสิทธิภาพ

ในงานวิจัยจะวัดประสิทธิภาพการทำงานใน 2 ด้าน คือ ด้านความถูกต้องและด้านเวลาในการทำงาน โดยในด้านความถูกต้องจะเป็นการวัดเพื่อดูว่ากฎที่ได้สามารถตรวจสอบข้อผิดพลาดได้ถูกต้องมากน้อยเพียงใด และในการวัดประสิทธิภาพด้านเวลาจะเป็นการวัดว่าโปรแกรมจะใช้เวลามากน้อยเพียงใดในการตรวจสอบความถูกต้องของข้อมูลแต่ละชุด โดยจะอธิบายในหัวข้อ 1.3.1 และ 1.3.2 ตามลำดับ

1.3.1 การวัดประสิทธิภาพในด้านความถูกต้องในการทำงาน

เมื่อได้ความสัมพันธ์จากการเรียนรู้ข้อมูลฝึกสอนโดยใช้เครื่องมือ WEKA มาแล้ว จำเป็นต้องมีการวัดประสิทธิภาพในด้านความถูกต้องในการทำงานก่อนที่จะนำไปสกัดเป็นกฎ เพื่อให้ความสัมพันธ์ที่ได้นั้นสามารถใช้ในการตรวจสอบเหตุการณ์ที่เข้ามาได้อย่างถูกต้อง

จากการเรียนรู้โดยใช้เครื่องมือ WEKA ประสิทธิภาพในการเรียนรู้จะถูกแสดงออกมาในรูปของจำนวนข้อมูลที่เบย์เซียนเน็ตเวิร์คทำนายถูกและผิด ซึ่งสามารถแบ่งได้เป็น 2 กลุ่ม โดยข้อมูลกลุ่มหนึ่งให้ค่าผิดปกติ (positive) ซึ่งให้ค่าเป็นบวก และอีกกลุ่มให้ค่าปกติ (negative) ซึ่งให้ค่าเป็นลบ ดังนั้นในการวัดประสิทธิภาพจึงใช้ค่า Sensitivity และ Specificity มาใช้ในการ

ประเมินประสิทธิภาพ ซึ่ง Sensitivity และ Specificity เป็นตัวชี้วัดที่ใช้สำหรับการทดสอบข้อมูลที่แบ่งออกเป็น 2 ชุด โดย Sensitivity เป็นการวัดสัดส่วนค่าที่ถูกกำหนดเป็นบวกที่เกิดขึ้นจริงซึ่งมีการระบุไว้อย่างถูกต้อง ส่วน specificity เป็นการวัดสัดส่วนค่าที่ถูกกำหนดเป็นลบที่เกิดขึ้นจริงซึ่งมีการระบุไว้อย่างถูกต้อง โดยตัวชี้วัดทั้งสองมีความสัมพันธ์ใกล้เคียงกับ type I (false positive error) และ type II error (false negative error) ตามลำดับ ซึ่งสามารถคำนวณได้ดังแสดงในตารางที่ 3

ตารางที่ 3 แสดงการคำนวณหาค่า Sensitivity และ Specificity

	Condition Positive	Condition Negative
Test Outcome Positive	True Positive	False Positive
Test Outcome Negative	False Negative	True Negative
	Sensitivity = TP / (TP + FN)	Specificity = TN / (FP + TN)

จากตารางที่ 3 สามารถอธิบายค่าที่ใช้แบ่งแยกระหว่างข้อมูลทั้งสองกลุ่ม ได้ดังนี้

1. กรณีที่ข้อมูลมีความผิดปกติถูกจัดเป็นบวกอย่างถูกต้อง (TP = True Positive fraction)
2. กรณีที่ข้อมูลมีความผิดปกติแต่ถูกจัดเป็นลบ (FN = False Negative fraction)
3. กรณีที่ข้อมูลไม่มีความผิดปกติถูกจัดเป็นลบอย่างถูกต้อง (TN = True Negative fraction)
4. กรณีที่ข้อมูลไม่มีความผิดปกติแต่ถูกจัดเป็นบวก (FP = False Positive fraction)

1.3.2 การวัดประสิทธิภาพในด้านเวลาในการทำงาน

ในส่วนนี้จะเป็นการทดสอบประสิทธิภาพด้านเวลาในการทำงานของกฎแต่ละชุด เพื่อเปรียบเทียบเวลาในการทำงานของกฎ โดยในการวัดประสิทธิภาพจะใช้ข้อมูลทดสอบชุด

เดียวกันกับข้อมูลที่ใช้ทดสอบความถูกต้อง โดยจะวัดหลังจากที่ได้สกัดกฎออกมาแล้ว เพื่อดูว่าเมื่อนำกฎมาใช้ในโปรแกรมแล้วโปรแกรมจะใช้เวลาในการทำงานเพิ่มขึ้นมากน้อยเพียงใด

1.4 ขั้นตอนการสกัดกฎ

ในการสกัดกฎนั้น ผู้วิจัยจะใช้ค่าความน่าจะเป็นที่อยู่ในตารางค่าความน่าจะเป็นมาเป็นเกณฑ์ (threshold) ในการเลือกความสัมพันธ์เพื่อนำมาสกัดเป็นกฎ เพื่อลดจำนวนกฎที่จะนำมาใช้ในการตรวจสอบซึ่งจะทำให้เวลาที่ต้องใช้ในการตรวจสอบลดลงและลดความซ้ำซ้อนของกฎ ซึ่งในงานวิจัยเราสนใจที่จะตรวจสอบข้อผิดพลาดโดยใช้ข้อผิดพลาดที่มีค่าผลเฉลยเป็น TRUE ดังนั้นในการสกัดกฎ ผู้วิจัยจึงเลือกใช้ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5 เนื่องจากว่าหากค่าความน่าจะเป็นของผลเฉลยมากกว่า 0.5 แสดงว่ากฎที่ได้มีความน่าจะเป็นที่จะทำให้ระบบมีโอกาสเกิดข้อผิดพลาดขึ้นได้ และจะแบ่งกฎที่ได้จากการสกัดออกเป็น 4 ชุด ดังนี้

1. กฎที่สกัดจากความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5
2. กฎที่สกัดจากความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.6
3. กฎที่สกัดจากความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.7
4. กฎที่สกัดจากความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.8

ในการนำความสัมพันธ์มาสกัดเป็นกฎนั้น ผู้วิจัยได้รวมความสัมพันธ์ที่มีค่าใกล้เคียงกันให้อยู่ในรูปของความสัมพันธ์เดียวกันโดยไม่ให้ส่งผลต่อความถูกต้องในการทดสอบ ในการรวมความสัมพันธ์บางข้อเข้าด้วยกันนี้จะช่วยลดจำนวนกฎที่ต้องใช้ในการทดสอบ ตัวอย่างเช่น ความสัมพันธ์ทั้งสามในตารางที่ 4 ซึ่งมีค่า drive, speed, bxw, byw และ x เป็นค่าเดียวกัน และค่า y มีช่วงค่าต่อเนื่องที่สามารถรวมให้เป็นความสัมพันธ์เดียวกันได้ ดังนั้นจึงรวมความสัมพันธ์ทั้งสามให้เป็นความสัมพันธ์เดียวกัน ดังแสดงในตารางที่ 4

ตารางที่ 4 ตัวอย่างความสัมพันธ์ของข้อมูลทางตรงที่สามารถรวมเป็นความสัมพันธ์เดียวกันได้

drive	speed	bxw	byw	x	y
motoron	(-inf,3.5]	(-inf,96]	(-inf,256]	(0.5,11.5]	(96.5,109.5]
motoron	(-inf,3.5]	(-inf,96]	(-inf,256]	(0.5,11.5]	(109.5,207.5]
motoron	(-inf,3.5]	(-inf,96]	(-inf,256]	(0.5,11.5]	(207.5,222.5]
motoron	(-inf,3.5]	(-inf,96]	(-inf,256]	(0.5,11.5]	(96.5,222.5]

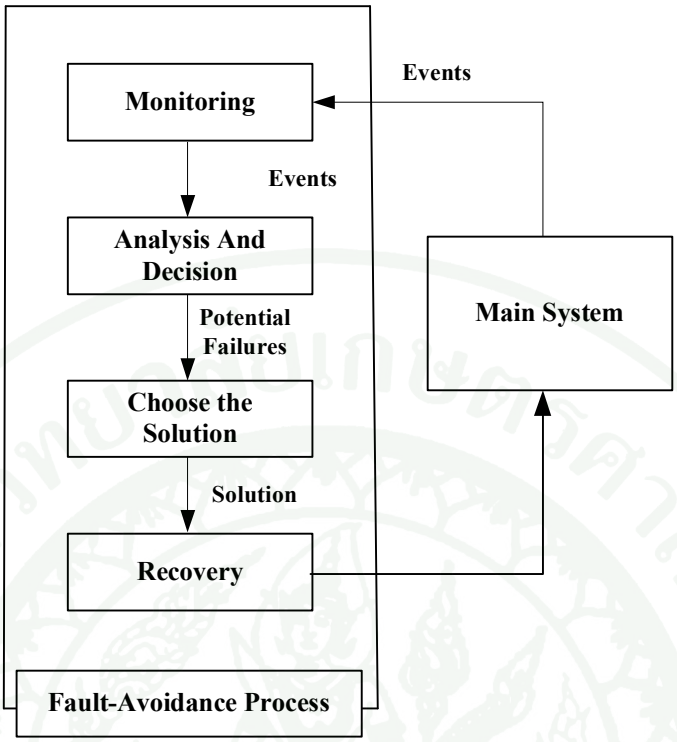
เมื่อได้ความสัมพันธ์มาแล้วจากนั้น ผู้วิจัยได้นำความสัมพันธ์เหล่านี้มาเขียนเป็นกฎในรูป IF-THEN เพื่อนำไปใช้ในส่วนหลีกเลี่ยงข้อผิดพลาดต่อไป

2. ส่วนของการหลีกเลี่ยงข้อผิดพลาด

ในส่วนนี้จะนำกฎที่ได้จากการเรียนรู้ของเบย์เซียนเน็ตเวิร์คมาใช้ในการดักจับเหตุการณ์และพิจารณาว่าเหตุการณ์ที่ดักจับนั้นมีส่วนทำให้เกิดความผิดพลาดในระบบหรือไม่ หากเหตุการณ์ที่ดักจับนั้นมีส่วนทำให้เกิดความผิดพลาดก็จะทำการแก้ไขข้อผิดพลาดตามกฎที่ได้สกัดขึ้น แต่หากเหตุการณ์ที่ดักจับนั้นไม่มีส่วนทำให้ระบบเกิดความผิดพลาด ระบบก็จะทำงานตามปกติ

2.1 ขั้นตอนการพัฒนาส่วนหลีกเลี่ยงข้อผิดพลาด

ในขั้นตอนนี้ผู้วิจัยจะใช้ AspectJ ซึ่งมีความสามารถในการทำงานแบบ aspect oriented programming มาใช้ในการพัฒนาระบบในส่วนนี้ขึ้นมา โดย AspectJ จะสอดแทรกกฎเพื่อตรวจสอบเหตุการณ์ภายในระบบโดยจะตรวจสอบเหตุการณ์ทุกครั้งที่มีการรับข้อมูลจากผู้ใช้เข้ามา โดยในกรณีที่เหตุการณ์ที่ดักจับได้นั้นตรงกับกฎซึ่งระบุว่าเหตุการณ์นั้นจะทำให้ระบบเกิดความผิดพลาดขึ้น ระบบจะปรับปรุงการทำงานบางอย่างเพื่อหลีกเลี่ยงข้อผิดพลาด จากนั้นระบบจะทำงานต่อไปตามปกติ ดังแสดงในภาพที่ 15

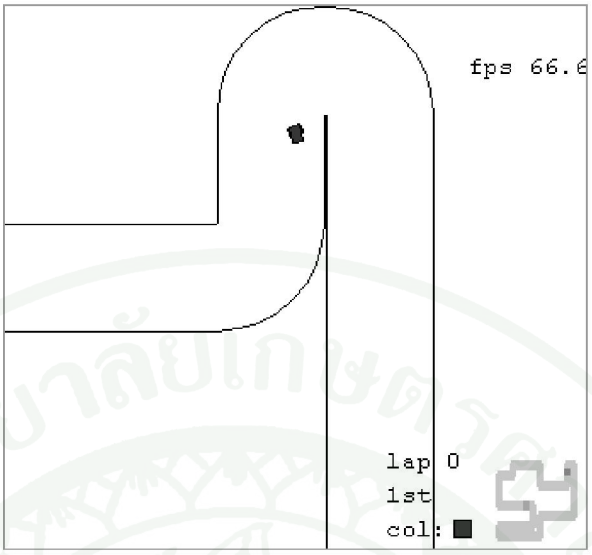


ภาพที่ 15 แสดงการทำงานในส่วนของการหลีกเลี่ยงข้อผิดพลาด

3. กรณีศึกษา

ในงานวิจัยได้ใช้โปรแกรม Racing Game 1.0 (J. montgomery, 2011) มาทดสอบสมมติฐานในการสกัดกฎโดยใช้เบย์เซียนเน็ตเวิร์ค โดยโปรแกรม Racing Game 1.0 (J. montgomery, 2011) มีลักษณะการทำงานดังนี้

โปรแกรม Racing Game 1.0 (J. montgomery, 2011) เป็นโปรแกรมแข่งรถที่มีรถแข่งจำนวน 3 คัน ออกวิ่งพร้อมกัน แต่ละคันจะวิ่งไปเรื่อยๆ คันไหนถึงเส้นชัยก่อนก็จะชนะ โดยรถแต่ละคันจะมีการคำนวณค่า x, y ซึ่งจะใช้เป็นตัวระบุตำแหน่งภายในสนาม ดังแสดงในภาพที่ 16



ภาพที่ 16 แสดงโปรแกรม Racing Game 1.0

ที่มา: (J. montgomery, 2011)

ในงานวิจัย ผู้วิจัยได้ปรับปรุงการทำงานบางส่วนเพื่อให้การอธิบายหลักการทำงานในการสกัดกฎเป็นไปได้ง่ายและเห็นภาพได้ชัดเจน โดยหัวข้อ 3.1 จะอธิบายการปรับปรุงโปรแกรมและในหัวข้อที่ 3.2 และ 3.3 จะอธิบายการสกัดกฎและการหลีกเลี่ยงข้อผิดพลาด ตามลำดับ

3.1 การปรับปรุงโปรแกรม Racing Game 1.0

ในงานวิจัยได้มีการปรับปรุงโค้ดและได้กำหนดให้ข้อผิดพลาดในโปรแกรม Racing Game 1.0 คือ เหตุการณ์ที่รถชนเข้ากับขอบสนาม เพื่อให้สามารถทำความเข้าใจหลักการในการสกัดกฎได้ง่าย โดยได้มีการปรับปรุงดังนี้ คือ กำหนดให้สถานะปกติ คือ รถสามารถวิ่งรอบสนามได้โดยไม่มีการชนเกิดขึ้นแต่หากมีการชนเข้ากับขอบสนามรถก็จะอยู่ในสถานะไม่ปกติและโปรแกรมจะหยุดการทำงานทันที ในการพิจารณาการชนขอบสนามของรถจะใช้ค่า x, y ของสนามและรถแข่งเป็นตัวกำหนด โดยค่า x, y เป็นค่าที่ได้จากการทำงานของโปรแกรม ถ้าค่า x, y ของรถแข่งมีค่าอยู่ระหว่างค่า x, y ของสนามแสดงว่ารถแข่งไม่มีการชนเกิดขึ้น แต่ถ้าค่า x, y ของรถแข่งมีค่ามากกว่าหรือน้อยกว่าค่า x, y ของสนาม แสดงว่ารถแข่งมีการชนเกิดขึ้น

3.2 ส่วนของการสกัดกฎ

ในส่วนของการสกัดกฎจะแบ่งการทำงานออกเป็น 4 ขั้นตอนย่อย ซึ่งจะอธิบายในหัวข้อ 3.2.1-3.2.4 ตามลำดับ

3.2.1 การเก็บข้อมูลและการคัดกรองข้อมูล

ในการเก็บข้อมูลเพื่อนำมาใช้ในการสกัดกฎนั้น ผู้วิจัยได้เก็บข้อมูลซึ่งเป็นเหตุการณ์ที่เกี่ยวข้องกับข้อผิดพลาดจากการทำงานของโปรแกรม โดยตัวแปรแต่ละตัวในเหตุการณ์จะใช้แทนปัจจัยที่ส่งผลต่อการชนและผลเฉลยจากการประมวลผลของโปรแกรม ซึ่งจะเก็บข้อมูลทั้งในส่วนที่มีการชนและไม่มีการชนมาใช้เป็นตัวอย่างในการเรียนรู้ให้กับเบย์เซียนเน็ตเวิร์คเพื่อใช้ในการสร้างกฎต่อไป ตัวอย่างข้อมูลที่ได้จากการเก็บข้อมูลนั้นจะขึ้นอยู่กับสภาพและโครงสร้างของสนามเพื่อให้สามารถมองเห็นการทำงานในการสกัดกฎได้อย่างชัดเจน ผู้วิจัยจึงได้ใช้รูปแบบของสนามรูปสี่เหลี่ยมที่มีมุมโค้ง

เนื่องจากมีตัวแปรบางตัวที่จะส่งผลให้เกิดสถานะปกติและสถานะไม่ปกติในกรณีที่ต่างกัน ดังนั้นจึงจะแยกการเก็บข้อมูลออกเป็น 2 ส่วน คือ ตัวอย่างเหตุการณ์ในกรณีที่สนามเป็นทางตรงและตัวอย่างเหตุการณ์ในกรณีที่สนามเป็นทางโค้ง โดยตัวแปรที่ใช้ในการทดลองสามารถแบ่งได้เป็น 2 ประเภท คือ ตัวแปรที่ได้จากผู้ใช้และตัวแปรที่ได้จากการประมวลผลของโปรแกรม ซึ่งตัวแปรที่ได้รับจากผู้ใช้มีดังนี้ drive, steer, speed และตัวแปรที่ได้รับจากการประมวลผลของโปรแกรมมีดังนี้ bx, x, bxw, by, y, byw, left, top, dis และ colliding ซึ่งตัวแปรแต่ละตัวจะอธิบายในหัวข้อ 3.2.1.1 และ 3.2.1.2 ตามลำดับ

3.2.1.1 การเก็บข้อมูลในกรณีที่สนามเป็นทางตรง ข้อมูลที่ใช้ได้มาจากการเก็บข้อมูลการทำงานของโปรแกรม ดังแสดงในภาพที่ 17 ซึ่งประกอบด้วยตัวแปรทั้งหมด ดังนี้

1. drive: รถขับตรงไปข้างหน้า - ถอยหลัง
2. steer: รถเลี้ยวซ้าย - ขวา
3. speed: ความเร็วของรถแข่ง กำหนดไว้ไม่เกิน 10
4. bx: ตำแหน่งของขอบถนนด้านในตามแนวแกน x

5. x: ตำแหน่งของรถตามแนวแกน x
6. bxw: ตำแหน่งของขอบถนนด้านนอกตามแนวแกน x
7. by: ตำแหน่งของขอบถนนด้านในตามแนวแกน y
8. y: ตำแหน่งของรถตามแนวแกน y
9. byw: ตำแหน่งของขอบถนนด้านนอกตามแนวแกน y

ผลลัพธ์ คือ

colliding: ผลเฉลยที่บอกว่ารถชนกับขอบถนนหรือไม่ โดยค่า true หมายถึง การที่รถชนขอบถนน และ false หมายถึง การที่รถไม่ชนขอบถนน

1	drive	steer	speed	bx	x	bxw	by	y	byw	colliding
2	motoron	noKeyStee	0	192	245	256	96	133	160	FALSE
3	motoron	noKeyStee	0	192	245	256	96	123	160	FALSE
4	motoron	noKeyStee	0	192	251	256	96	123	160	FALSE
5	motoron	noKeyStee	0	192	251	256	96	133	160	FALSE
6	noKeyPss	noKeyStee	3	0	52	64	160	230	224	FALSE
7	motoron	noKeyStee	6	192	245	256	96	134	160	FALSE
8	motoron	noKeyStee	6	192	245	256	96	124	160	FALSE
9	motoron	noKeyStee	6	192	251	256	96	124	160	FALSE
10	motoron	noKeyStee	6	192	251	256	96	134	160	FALSE
11	motoron	noKeyStee	10	192	245	256	96	128	160	FALSE
12	motoron	noKeyStee	10	192	251	256	96	128	160	FALSE
13	motoron	noKeyStee	10	192	251	256	96	138	160	FALSE
14	motoron	noKeyStee	10	192	245	256	96	139	160	FALSE
15	motoron	noKeyStee	10	192	245	256	96	129	160	FALSE
16	motoron	noKeyStee	10	192	251	256	96	129	160	FALSE
17	motoron	noKeyStee	10	192	251	256	96	139	160	FALSE
18	motoron	noKeyStee	10	192	245	256	96	140	160	FALSE
19	motoron	noKeyStee	10	192	245	256	96	130	160	FALSE
20	motoron	noKeyStee	10	192	251	256	96	130	160	FALSE
21	motoron	noKeyStee	10	192	251	256	96	140	160	FALSE
22	motoron	noKeyStee	9	192	251	256	96	149	160	FALSE
23	motoron	noKeyStee	9	192	255	256	96	158	160	FALSE
24	motoron	noKeyStee	10	192	250	256	96	161	160	FALSE

ภาพที่ 17 แสดงตัวอย่างข้อมูลที่ใช้ในการเรียนรู้ในส่วนของทางตรง

3.2.1.2 การเก็บข้อมูลในกรณีที่สนามเป็นทางโค้ง ข้อมูลที่ใช้ได้มาจากการเก็บข้อมูลการทำงานของโปรแกรม ซึ่งตัวแปรส่วนใหญ่จะเหมือนกับข้อมูลทางตรงแต่มีตัวแปรที่เพิ่มขึ้น ดังแสดงในภาพที่ 18 ดังนี้

1. left: ตำแหน่งซ้ายของสนาม โดยค่า true หมายถึง รถอยู่บริเวณตำแหน่งซ้ายของสนาม และ false หมายถึง รถไม่อยู่บริเวณตำแหน่งซ้ายของสนาม

2. top: ตำแหน่งบนของสนาม สนาม โดยค่า true หมายถึง รถอยู่บริเวณตำแหน่งบนของสนาม และ false หมายถึง รถไม่อยู่บริเวณตำแหน่งบนของสนาม

3. dis: ค่ารัศมีความโค้งของขอบถนนทางโค้ง

1	drive	steer	speed	top	by	y	byw	left	bx	x	bxw	dis	colliding
2	motoron	steerRight	10	FALSE	32	97	96	FALSE	0	11	64	2745	FALSE
3	motoron	steerRight	10	FALSE	32	107	96	FALSE	0	8	64	3185	FALSE
4	motoron	steerRight	10	FALSE	32	105	96	FALSE	0	2	64	3816	FALSE
5	motoron	steerRight	10	FALSE	32	97	96	FALSE	0	11	64	2710	FALSE
6	motoron	steerRight	10	FALSE	32	106	96	FALSE	0	8	64	3184	FALSE
7	motoron	steerRight	10	FALSE	32	104	96	FALSE	0	2	64	3803	FALSE
8	motoron	steerRight	10	FALSE	32	97	96	FALSE	0	12	64	2674	FALSE
9	motoron	steerRight	10	FALSE	32	106	96	FALSE	0	8	64	3183	FALSE
10	motoron	steerRight	10	FALSE	32	104	96	FALSE	0	3	64	3789	FALSE
11	motoron	steerRight	10	FALSE	32	97	96	FALSE	0	12	64	2639	FALSE
12	motoron	steerRight	10	FALSE	32	106	96	FALSE	0	8	64	3180	FALSE
13	motoron	steerRight	10	FALSE	32	103	96	FALSE	0	3	64	3773	FALSE
14	motoron	steerRight	10	FALSE	32	97	96	FALSE	0	12	64	2603	FALSE
15	motoron	steerRight	10	FALSE	32	106	96	FALSE	0	8	64	3176	FALSE
16	motoron	steerRight	10	FALSE	32	98	96	FALSE	0	20	64	1939	FALSE
17	motoron	steerRight	10	FALSE	32	100	96	FALSE	0	10	64	2899	FALSE
18	motoron	steerRight	10	FALSE	32	98	96	FALSE	0	20	64	1915	FALSE
19	motoron	steerRight	10	FALSE	32	100	96	FALSE	0	10	64	2877	FALSE
20	motoron	steerRight	10	FALSE	32	98	96	FALSE	0	20	64	1891	FALSE
21	motoron	steerRight	10	FALSE	32	100	96	FALSE	0	10	64	2854	FALSE
22	motoron	steerRight	10	FALSE	32	98	96	FALSE	0	20	64	1869	FALSE
23	motoron	steerRight	10	FALSE	32	100	96	FALSE	0	10	64	2831	FALSE
24	motoron	steerRight	10	FALSE	32	99	96	FALSE	0	21	64	1848	FALSE

ภาพที่ 18 แสดงตัวอย่างข้อมูลที่ใช้ในการเรียนรู้ในส่วนของทางโค้ง

3.2.2 การเรียนรู้

ในขั้นตอนของการเรียนรู้ ผู้วิจัยได้นำข้อมูลฝึกสอนซึ่งเป็นเหตุการณ์ที่มีความเกี่ยวข้องกับการทำให้โปรแกรมอยู่ในสถานะปกติและไม่ปกติในถนนทางตรงจำนวน 8,579 เรคคอร์ด แบ่งตามผลเฉลยที่แสดงให้เห็นว่าโปรแกรมอยู่ในสถานะไม่ปกติ คือ มีการชนเกิดขึ้นจำนวน 4,799 เรคคอร์ดและผลเฉลยที่แสดงให้เห็นว่าโปรแกรมอยู่ในสถานะปกติ คือ ไม่มีการชนจำนวน 3,780 เรคคอร์ด และข้อมูลฝึกสอนซึ่งเป็นเหตุการณ์ที่มีความเกี่ยวข้องกับการทำให้โปรแกรมอยู่ในสถานะปกติและไม่ปกติในถนนทางโค้งจำนวน 7,611 เรคคอร์ด แบ่งตามผลเฉลยที่แสดงให้เห็นว่าโปรแกรมอยู่ในสถานะไม่ปกติมีจำนวน 4,495 เรคคอร์ดและผลเฉลยที่แสดงให้เห็นว่าโปรแกรมอยู่ในสถานะปกติมีจำนวน 3,116 เรคคอร์ด เมื่อนำข้อมูลฝึกสอนเหล่านี้มาให้เบย์เซียนเน็ตเวิร์คเรียนรู้โดยใช้เครื่องมือ WEKA แล้วจะได้กราฟแสดงความสัมพันธ์ระหว่างตัวแปรและตารางค่าความน่าจะเป็นอย่างมีเงื่อนไขของตัวแปรแต่ละตัว

3.2.3 การวัดประสิทธิภาพการเรียนรู้ในด้านความถูกต้องและเวลา

เมื่อได้ความสัมพันธ์จากการเรียนรู้แล้ว จำเป็นต้องมีการวัดประสิทธิภาพของความสัมพันธ์ก่อนและหลังการสกัดกฎ โดยในการวัดความสัมพันธ์ก่อนการสกัดกฎนั้น ค่า Sensitivity และค่า Specificity จะได้จากการใช้เครื่องมือ WEKA เพื่อพิจารณาว่าข้อมูลฝึกสอนชุดนั้นมีความน่าจะเป็นที่จะนำไปสกัดเป็นกฎหรือไม่ เมื่อได้ค่า Sensitivity และค่า Specificity ที่พอใจแล้ว ก็จะเข้าสู้กระบวนการสกัดกฎต่อไป

หลังจากขั้นตอนของการสกัดกฎ ผู้วิจัยได้วัดประสิทธิภาพด้านความถูกต้องของกฎที่ได้จากการเรียนรู้โดยใช้ข้อมูลทางตรงและข้อมูลทางโค้ง โดยวัดความถูกต้องตามกลุ่มของกฎที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับ เมื่อได้กฎที่มีความถูกต้องแล้ว ผู้วิจัยได้วัดประสิทธิภาพในด้านเวลาของกฎแต่ละกลุ่มโดยใช้ข้อมูลทดสอบชุดเดิม เพื่อเปรียบเทียบเวลาที่โปรแกรมจะต้องใช้ไปในการตรวจสอบเหตุการณ์กับกฎแต่ละชุด

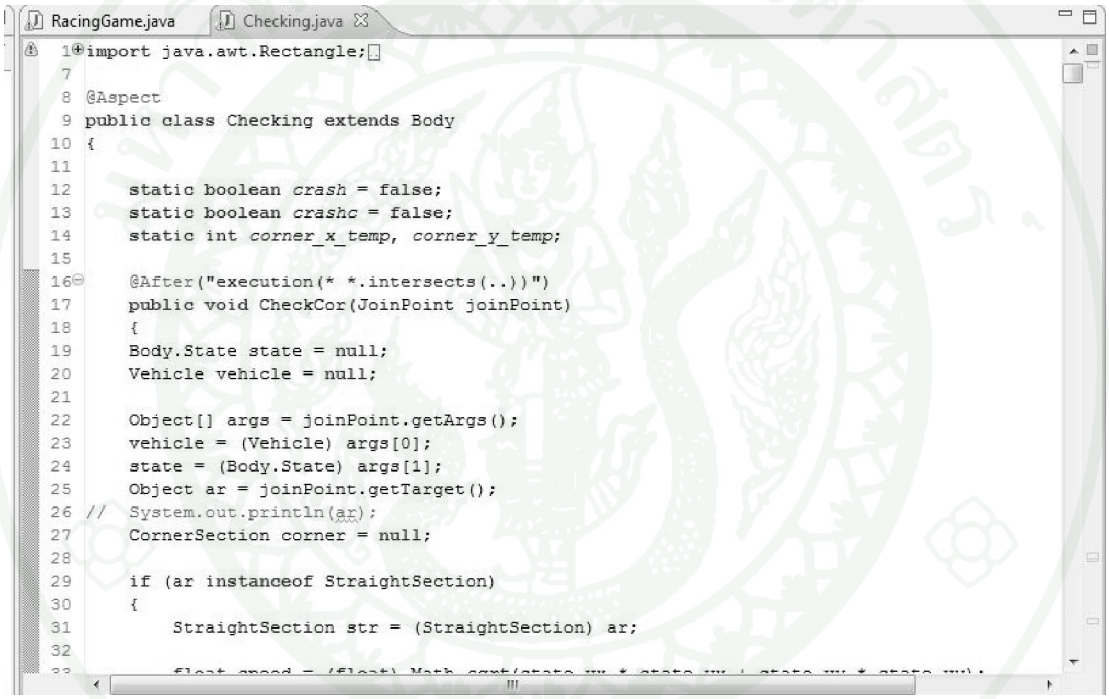
3.2.4 การสกัดกฎ

ในขั้นตอนของการสกัดกฎ ผู้วิจัยได้นำความสัมพันธ์ที่มีผลเฉลยที่แสดงให้เห็นว่าโปรแกรมอยู่ในสถานะไม่ปกติและมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับมาเขียนเป็นกฎให้อยู่ในรูป IF-THEN เพื่อนำไปใช้ในการตรวจสอบเหตุการณ์ซึ่งอยู่ในส่วนของการหลีกเลี่ยงข้อผิดพลาดต่อไป

3.3 ส่วนของการหลีกเลี่ยงข้อผิดพลาด

ในการสร้างส่วนหลีกเลี่ยงข้อผิดพลาด ผู้วิจัยจะพัฒนาส่วนนี้เพื่อให้มีความสามารถในการตรวจสอบเหตุการณ์และหลีกเลี่ยงสถานะไม่ปกติ โดยจะดักจับเหตุการณ์ที่เข้ามาและเมื่อเหตุการณ์เหล่านั้นได้รับการตรวจสอบกับกฎแล้วพบว่ามีความน่าจะเป็นที่จะทำให้โปรแกรมอยู่ในสถานะไม่ปกติ ก็จะทำการหลีกเลี่ยงข้อผิดพลาดทันที โดยการเลือกวิธีการหลีกเลี่ยงไม่ให้โปรแกรมอยู่ในสถานะไม่ปกตินั้นจะขึ้นอยู่กับความต้องการของผู้ใช้ โดยในงานวิจัยนี้ ผู้วิจัยจะใช้วิธีการหลีกเลี่ยงสถานะไม่ปกติโดยการย้อนสถานะกลับไปยังสถานะปกติก่อนหน้าสถานะปัจจุบันซึ่งเป็นสถานะไม่ปกติ โดยใช้ AspectJ เพื่อแทรกกฎเข้าไปในโปรแกรม

ผู้วิจัยได้ประกาศ class Checking เป็น aspect ดังแสดงในภาพที่ 19 เพื่อตรวจสอบกฎทุกครั้งก่อนที่โปรแกรมจะทำงานในส่วนของเมท็อด intersects ซึ่งเป็นเมท็อดที่จะประมวลผลข้อมูลก่อนนำไปใช้ในส่วนต่าง ๆ ของโปรแกรม โดยใน class Checking จะมีการตรวจสอบตัวแปรในเหตุการณ์ที่รับเข้ามากับกฎ ถ้าเหตุการณ์ที่รับเข้ามาตรงกับกฎที่แสดงว่าเหตุการณ์นั้นอาจจะส่งผลให้โปนแกรมเข้าสู่สถานะไม่ปกติ class Checking จะสั่งให้โปรแกรมนำข้อมูลที่อยู่ใน previousState ซึ่งเป็นสถานะปกติมาใส่ใน currentState ดังนั้นเมื่อเข้าสู่เมท็อด intersects โปรแกรมก็จะสามารถทำงานต่อไปได้



ภาพที่ 19 แสดงการเขียน class Checking

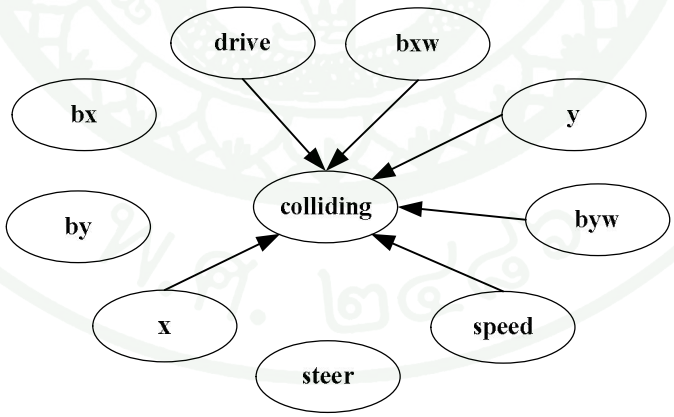
ผลและวิจารณ์

ผล

เมื่อนำข้อมูลที่ได้จากการเก็บข้อมูลจากการทำงานของโปรแกรม Racing Game 1.0 มาใช้ในการเรียนรู้แล้ว ผลการทดลองในส่วนการสกัดกฎสามารถแบ่งเป็น 2 ส่วน คือ ผลการทดลองที่ได้จากการเรียนรู้ข้อมูลทางตรงและผลการทดลองที่ได้จากการเรียนรู้ข้อมูลทางโค้ง โดยผลการทดลองทั้งสองสามารถอธิบายได้ดังในหัวข้อ 1 และ 2 และผลการทดลองในส่วนการหลีกเลี่ยงข้อผิดพลาดจะอธิบายในหัวข้อที่ 3 ดังนี้

1. ผลการทดลองที่ได้จากการเรียนรู้ข้อมูลทางตรง

ผู้วิจัยได้ใช้ข้อมูลในการทดลองจำนวน 8,579 เรคคอร์ด เมื่อแบ่งตามผลเฉลยแล้วจะเห็นว่ามีข้อมูลที่แสดงให้เห็นว่าระบบมีการชนเกิดขึ้นมีจำนวน 4,799 เรคคอร์ดและผลเฉลยที่แสดงให้เห็นว่าระบบไม่มีการชนเกิดขึ้นมีจำนวน 3,780 เรคคอร์ด เมื่อนำข้อมูลเหล่านี้ไปเรียนรู้โดยใช้เบย์เซียนเน็ตเวิร์คแล้ว สามารถแสดงในรูปแบบของกราฟได้ดังแสดงในภาพที่ 20



ภาพที่ 20 แสดงกราฟที่ได้จากการเรียนรู้โดยใช้ข้อมูลทางตรง

จากกราฟเป็นการแสดงความสัมพันธ์ของตัวแปรที่มีความขึ้นตรงต่อกันกับตัวแปรที่เราสนใจ (ผลเฉลย) ซึ่งในที่นี้ คือ colliding ซึ่งแสดงให้เห็นว่าค่าของ colliding มีผลมาจากตัวแปร drive, speed, x, bxw, y และ byw โดยสามารถแสดงค่าความน่าจะเป็นของผลเฉลยที่ได้จากกราฟใน

รูปตารางความน่าจะเป็นได้ดังแสดงในภาพที่ 21 ตัวอย่างความสัมพันธ์ของตัวแปร เช่น ในเรคคอร์ดที่ 2 มีเหตุการณ์ที่เข้ามาคือ รถขับตรงมาด้วยความเร็วน้อยกว่าหรือเท่ากับ 3.5 และอยู่ในตำแหน่งที่ x มีค่าอยู่ระหว่าง 134.5-190.5 และอยู่ในตำแหน่งที่ y มีค่าอยู่ระหว่าง 235.5-287.5 โดยขอบสนามด้านนอกตามแนวแกน x มีค่าน้อยกว่าหรือเท่ากับ 96 และขอบสนามด้านนอกตามแนวแกน y มีค่ามากกว่า 256 และมีค่าความน่าจะเป็นที่จะเกิดการชน คือ 0.5

	A	B	C	D	E	F	G	H
1	drive	speed	x	bxw	y	byw	FALSE	TRUE
2	motoron	(-inf-3.5]	(134.5-190.5]	(-inf-96]	(235.5-287.5]	(256-inf)	0.5	0.5
3	motoron	(-inf-3.5]	(134.5-190.5]	(-inf-96]	(287.5-inf)	(-inf-256]	0.5	0.5
4	motoron	(-inf-3.5]	(134.5-190.5]	(-inf-96]	(287.5-inf)	(256-inf)	0.5	0.5
5	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(-inf-31.5]	(-inf-256]	0.016129	0.983871
6	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(-inf-31.5]	(256-inf)	0.5	0.5
7	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(31.5-83.5]	(-inf-256]	0.5	0.5
8	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(31.5-83.5]	(256-inf)	0.5	0.5
9	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(83.5-86.5]	(-inf-256]	0.022727	0.977273
10	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(83.5-86.5]	(256-inf)	0.5	0.5
11	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(86.5-93.5]	(-inf-256]	0.083333	0.916667
12	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(86.5-93.5]	(256-inf)	0.5	0.5
13	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(93.5-95.5]	(-inf-256]	0.071429	0.928571
14	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(93.5-95.5]	(256-inf)	0.5	0.5
15	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(95.5-96.5]	(-inf-256]	0.00625	0.99375
16	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(95.5-96.5]	(256-inf)	0.5	0.5
17	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(96.5-109.5]	(-inf-256]	0.5	0.5
18	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(96.5-109.5]	(256-inf)	0.5	0.5
19	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(109.5-207.5]	(-inf-256]	0.5	0.5
20	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(109.5-207.5]	(256-inf)	0.5	0.5
21	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(207.5-222.5]	(-inf-256]	0.5	0.5
22	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(207.5-222.5]	(256-inf)	0.5	0.5
23	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(222.5-223.5]	(-inf-256]	0.5	0.5
24	motoron	(-inf-3.5]	(134.5-190.5]	(96-224]	(222.5-223.5]	(256-inf)	0.006329	0.993671

ภาพที่ 21 แสดงตารางความน่าจะเป็นของความสัมพันธ์ที่ได้จากการเรียนรู้ข้อมูลทางตรง

จากตารางความน่าจะเป็นในภาพที่ 21 สามารถหาความสัมพันธ์ที่เกี่ยวข้องกับการชนได้ทั้งหมด 29,161 กรณี และเมื่อเลือกความสัมพันธ์ที่มีผลเฉลยเป็น TRUE ซึ่งเป็นเหตุการณ์ที่อาจมีการชนเกิดขึ้นและมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับ มาเขียนให้อยู่ในรูปของกฎเพื่อตรวจสอบความถูกต้องในการทำงานนั้น จะได้ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5 มีจำนวน 403 กรณี เมื่อนำมาเขียนเป็นกฎจะได้กฎจำนวน 205 ข้อ ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.6 มีจำนวน 402 กรณี เมื่อนำมาเขียนเป็นกฎจะได้กฎจำนวน 204 ข้อ ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.7 มีจำนวน 401 กรณี เมื่อนำมาเขียนเป็นกฎจะได้กฎจำนวน 203 ข้อและ

ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.8 มีจำนวน 255 กรณี เมื่อนำมาเขียนเป็นกฎจะได้กฎจำนวน 123 ข้อ ตามลำดับ ดังแสดงในตารางที่ 5

ตารางที่ 5 แสดงการสกัดกฎที่ได้จากข้อมูลทางตรง

	ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE				
ความน่าจะเป็น	ทั้งหมด	มากกว่า 0.5	มากกว่า 0.6	มากกว่า 0.7	มากกว่า 0.8
ความสัมพันธ์	29,161	403	402	401	255
กฎ	29,161	205	204	203	123

1.1 การวัดประสิทธิภาพในการทำงานในด้านความถูกต้องในการทำงาน

ในการวัดประสิทธิภาพผู้วิจัยจะนำข้อมูลทดสอบจำนวน 18,057 ตัวอย่าง ซึ่งมีผลเฉลยเป็น TRUE (รถชนขอบสนาม) จำนวน 2,376 ตัวอย่าง และมีผลเฉลยเป็น FALSE จำนวน 15,680 ตัวอย่าง โดยจะทดสอบโดยใช้ความสัมพันธ์ทั้งหมด และแบ่งความสัมพันธ์ออกเป็น 4 ชุด คือ ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับ โดยจำนวนความสัมพันธ์แต่ละชุดให้ค่าการทดสอบและการวัดประสิทธิภาพด้วยค่า Sensitivity และ Specificity ดังแสดงในตารางที่ 6 - 10 ตามลำดับ

ตารางที่ 6 แสดงผลการทดสอบข้อมูลทางตรงโดยใช้ความสัมพันธ์ทั้งหมด

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	2,361	1,891
FALSE	15	13,789
Sensitivity	0.9936	
Specificity	0.8794	

ตารางที่ 7 แสดงผลการทดสอบข้อมูลทางตรงโดยใช้ความสัมพันธ์ที่มีค่าความน่าจะเป็นมากกว่า 0.5

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	2,335	1,051
FALSE	41	14,629
Sensitivity	0.9827	
Specificity	0.9329	

ตารางที่ 8 แสดงผลการทดสอบข้อมูลทางตรงโดยใช้ความสัมพันธ์ที่มีค่าความน่าจะเป็นมากกว่า 0.6

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	2,334	1,051
FALSE	42	14,629
Sensitivity	0.9823	
Specificity	0.9329	

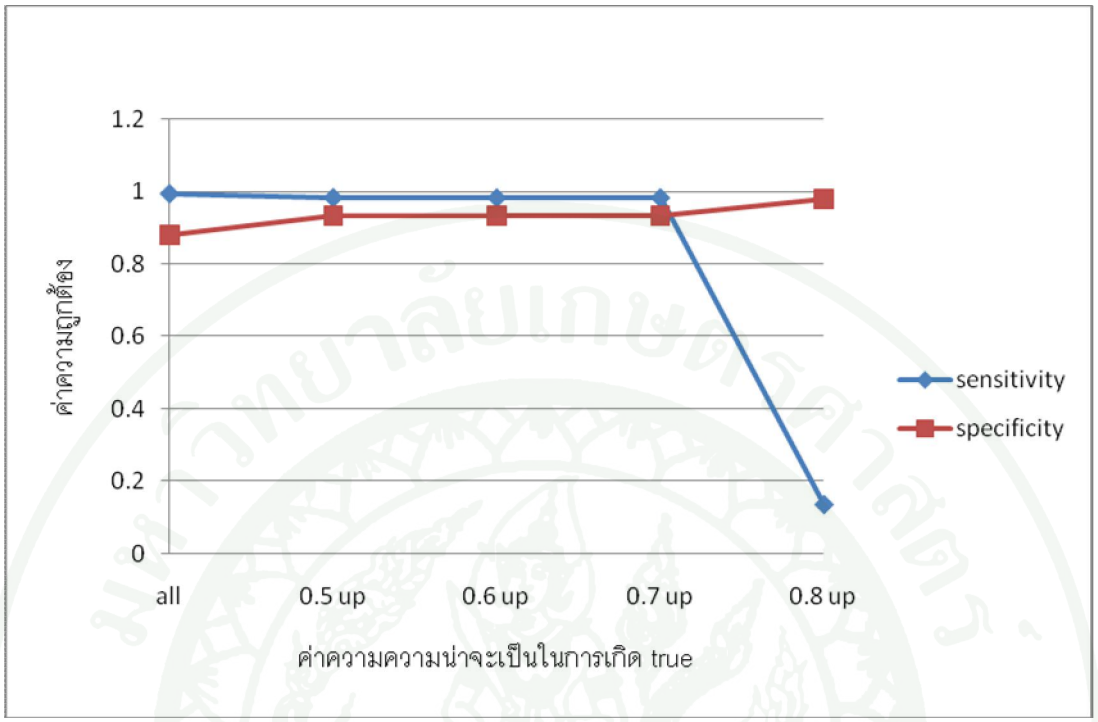
ตารางที่ 9 แสดงผลการทดสอบข้อมูลทางตรงโดยใช้ความสัมพันธ์ที่มีค่าความน่าจะเป็นมากกว่า 0.7

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	2,334	1,034
FALSE	42	14,646
Sensitivity	0.9823	
Specificity	0.9340	

ตารางที่ 10 แสดงผลการทดสอบข้อมูลทางตรงโดยใช้ความสัมพันธ์ที่มีค่าความน่าจะเป็นมากกว่า 0.8

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	328	308
FALSE	2,048	15,372
Sensitivity	0.1380	
Specificity	0.9803	

จากผลการทดสอบในด้านความถูกต้องในการเปรียบเทียบข้อมูลทดสอบกับความสัมพันธ์ที่เลือกนำมาเขียนให้อยู่ในรูปของกฎพบว่าค่า Sensitivity ของการทดสอบมีค่าที่เรียงลำดับจากมากไปน้อย คือ 0.9936, 0.9827, 0.9823, 0.9823 และ 0.1380 ตามลำดับซึ่งจะเห็นว่ายังคงมีค่าที่ใกล้เคียงกัน ยกเว้นในตารางที่ 10 ที่ค่า sensitivity มีค่าน้อยลงอย่างเห็นได้ชัด ในขณะที่ค่า Specificity มีค่าที่ตรงกันข้าม คือ มีค่าที่เรียงลำดับจากน้อยไปมาก คือ 0.8794, 0.9329, 0.9329, 0.9340 และ 0.9803 ซึ่งค่าที่ได้ไม่ได้มีความแตกต่างกันอย่างชัดเจน แสดงในรูปแบบกราฟดังภาพที่ 22 ซึ่งจะมีการวิเคราะห์ผลการทดลองในส่วนของการวิจารณ์ต่อไป



ภาพที่ 22 แสดงการเปรียบเทียบค่า Sensitivity และ Specificity ของกฎแต่ละชุด

1.2 การวัดประสิทธิภาพในการทำงานในด้านเวลาในการทำงาน

ในการวัดประสิทธิภาพในด้านเวลา ผู้วิจัยได้ใช้คอมพิวเตอร์ Intel Celeron® CPU 2.20 GHz มีแรม 752 MB และฮาร์ดดิสก์ 40 GB โดยนำข้อมูลทดสอบซึ่งเป็นชุดเดียวกันกับที่ใช้ในการวัดความถูกต้องในการทำงานมาใช้เป็นข้อมูลทดสอบ โดยทำการวัดประสิทธิภาพในด้านเวลาในการตรวจสอบกฎ เพื่อเปรียบเทียบเวลาที่ใช้ในการตรวจสอบของกฎแต่ละชุด ซึ่งสามารถแสดงได้ดังในตารางที่ 11

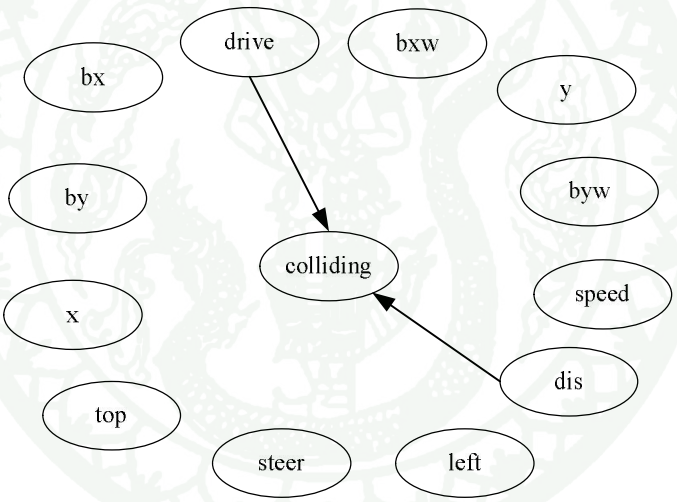
ตารางที่ 11 แสดงการวัดประสิทธิภาพในด้านเวลาในการทำงานของกฎแต่ละชุด

กฎที่ได้จาก	ความสัมพันธ์ทั้งหมด	มีความน่าจะเป็นมากกว่า			
		0.5	0.6	0.7	0.8
เวลาที่ใช้ (sec)	0.2700	0.1667	0.1510	0.1539	0.1001

จากการทดสอบการทำงานของกฎโดยใช้ข้อมูลทดสอบจำนวน 18,057 ตัวอย่าง ผลปรากฏว่าในการตรวจสอบข้อผิดพลาดของกฎใช้เวลาลดลงเมื่อจำนวนกฎลดลง ซึ่งแสดงให้เห็นว่าจำนวนกฎที่น้อยจะทำให้เวลาในการตรวจสอบกฎน้อยลงไปด้วย

2. ผลการทดลองที่ได้จากการเรียนรู้ข้อมูลทางโค้ง

ข้อมูลที่ใช้ในการเรียนรู้มีจำนวน 7,611 เรคคอร์ด แบ่งตามผลเฉลยที่เป็น TRUE มีจำนวน 4,495 เรคคอร์ดและแบ่งตามผลเฉลยที่เป็น FALSE มีจำนวน 3,116 เรคคอร์ด เมื่อนำข้อมูลที่ส่งผลต่อการเกิดการชนของเส้นทางโค้งไปเรียนรู้โดยใช้เบย์เซียนเน็ตเวิร์คแล้ว สามารถแสดงในรูปแบบของกราฟได้ดังแสดงในภาพที่ 23



ภาพที่ 23 แสดงกราฟที่ได้จากการเรียนรู้โดยใช้ข้อมูลทางโค้ง

จากกราฟสามารถแสดงความสัมพันธ์ที่มีความขึ้นตรงต่อกันของตัวแปรที่เราสนใจซึ่งในที่นี้ คือ colliding ซึ่งมีโหนดพ่อแม่ คือ drive และ dis และแสดงค่าความน่าจะเป็นของผลเฉลยได้จากกราฟ ตัวอย่างความสัมพันธ์ของตัวแปรจากตารางความน่าจะเป็นในภาพที่ 24 เช่น เรคคอร์ดที่ 2 คือ ผู้ใช้ไม่ได้กดปุ่มใดในขณะที่ตอนนั้นค่ารัศมีความโค้งของสนามมีค่าระหว่าง 4095.5 - 4.112.5 จากเหตุการณ์นี้ค่าความน่าจะเป็นในการชนมีค่าเป็น 0.998

	A	B	C	D
1	drive	dis	FALSE	TRUE
2	noKeyPss	(4095.5-4112.5]	0.001736111	0.998263889
3	reversing	(4095.5-4112.5]	0.004132231	0.995867769
4	motoron	(4095.5-4112.5]	0.007894737	0.992105263
5	motoron	(4166.5-inf)	0.02	0.98
6	noKeyPss	(4112.5-4139.5]	0.025	0.975
7	motoron	(4112.5-4139.5]	0.051507538	0.948492462
8	reversing	(4112.5-4139.5]	0.071428571	0.928571429
9	noKeyPss	(4077.5-4095.5]	0.5	0.5
10	noKeyPss	(4139.5-4166.5]	0.5	0.5
11	noKeyPss	(4166.5-inf)	0.5	0.5
12	reversing	(-inf-2900]	0.5	0.5
13	reversing	(2900-2972]	0.5	0.5
14	reversing	(2972-3095.5]	0.5	0.5
15	reversing	(3095.5-4077.5]	0.5	0.5

ภาพที่ 24 แสดงตารางความน่าจะเป็นของความสัมพันธ์ที่ได้จากการเรียนรู้ข้อมูลทางโค้ง

จากตารางความน่าจะเป็นในภาพที่ 24 จะเห็นว่าสามารถหาความสัมพันธ์ที่เกี่ยวข้องกับการชนของทางโค้งได้คล้ายคลึงกับการหาความสัมพันธ์ที่เกี่ยวข้องกับการชนของทางตรงโดยมีความสัมพันธ์ทั้งหมด 28 กรณี ซึ่งประกอบไปด้วยความสัมพันธ์ที่เกี่ยวข้องกับผลเฉลยที่ให้ค่าเป็น TRUE และ FALSE และเมื่อนำความสัมพันธ์เหล่านั้นมาสกัดกฎโดยเลือกใช้ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นที่มากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับ ซึ่งความสัมพันธ์ที่มีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 มีจำนวน 8 กรณี

2.1 การวัดประสิทธิภาพในการเรียนรู้

ในการวัดประสิทธิภาพผู้วิจัยจะนำข้อมูลทดสอบจำนวน 6,354 ตัวอย่าง ซึ่งมีผลเฉลยเป็น TRUE (รถชนขอบสนาม) จำนวน 1,314 ตัวอย่าง และมีผลเฉลยเป็น FALSE จำนวน 5,040 ตัวอย่าง โดยจะทดสอบโดยใช้ความสัมพันธ์ทั้งหมด และแบ่งความสัมพันธ์ออกเป็น 4 ชุด คือ ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ตามลำดับ แต่เนื่องจากความสัมพันธ์ความสัมพันธ์ที่มีผลเฉลยเป็น TRUE และมีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8 ให้ค่าผลการทดสอบเดียวกัน ดังนั้นจึงเขียนแสดงในตารางที่ 12-13 ตามลำดับ

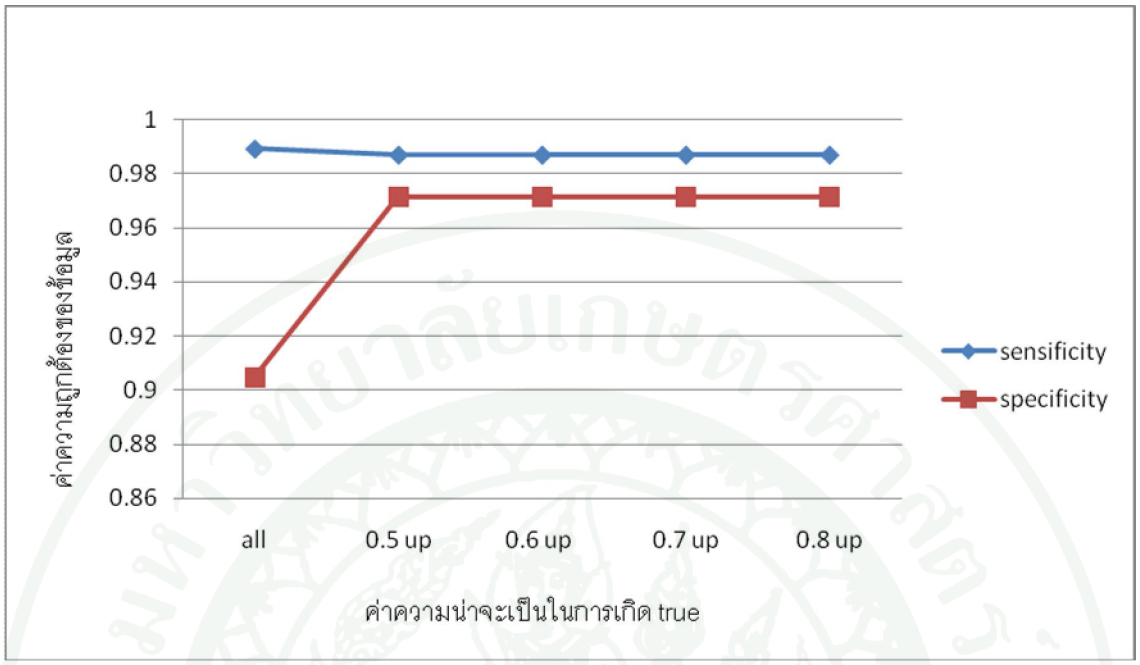
ตารางที่ 12 แสดงผลการทดสอบข้อมูลทางโค้งโดยใช้ความสัมพันธ์ทั้งหมด

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	1,300	481
FALSE	14	4,559
Sensitivity	0.9893	
Specificity	0.9045	

ตารางที่ 13 แสดงผลการทดสอบข้อมูลทางโค้งที่มีค่าความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7 และ 0.8

Test outcome	Condition Positive (TRUE)	Condition Negative (FALSE)
TRUE	1,296	143
FALSE	18	4,897
Sensitivity	0.9870	
Specificity	0.9714	

จากผลการทดสอบในด้านความถูกต้อง เมื่อนำข้อมูลทดสอบมาเปรียบเทียบกับความสัมพันธ์ที่เลือกนำมาใช้ในการเขียนให้อยู่ในรูปของกฎพบว่าค่า Sensitivity ของการทดสอบมีค่าที่เรียงลำดับจากมากไปน้อย คือ 0.9893, 0.9870, 0.9870, 0.9870 และ 0.9870 ตามลำดับซึ่งจะเห็นว่ายังคงมีค่าที่ใกล้เคียงกัน ในขณะที่ค่า Specificity มีค่าที่ตรงกันข้าม คือ มีค่าที่เรียงลำดับจากน้อยไปมาก คือ 0.9045, 0.9714, 0.9714, 0.9714 และ 0.9714 ซึ่งค่าที่ไม่ได้แตกต่างกันอย่างชัดเจน แสดงในรูปแบบกราฟดังภาพที่ 25



ภาพที่ 25 แสดงการเปรียบเทียบค่า Sensitivity และ Specificity ของกฎแต่ละชุด

2.2 การวัดประสิทธิภาพในการทำงานในด้านเวลาในการทำงาน

ในการวัดประสิทธิภาพในด้านเวลา ผู้วิจัยได้นำข้อมูลทดสอบซึ่งเป็นชุดเดียวกันกับที่ใช้ในการวัดความถูกต้องในการทำงานมาใช้เป็นข้อมูลทดสอบ โดยทำการวัดประสิทธิภาพในด้านเวลาในการตรวจสอบกฎ เพื่อเปรียบเทียบเวลาที่ใช้ในการตรวจสอบของกฎแต่ละชุด ซึ่งสามารถแสดงได้ดังตารางที่ 14

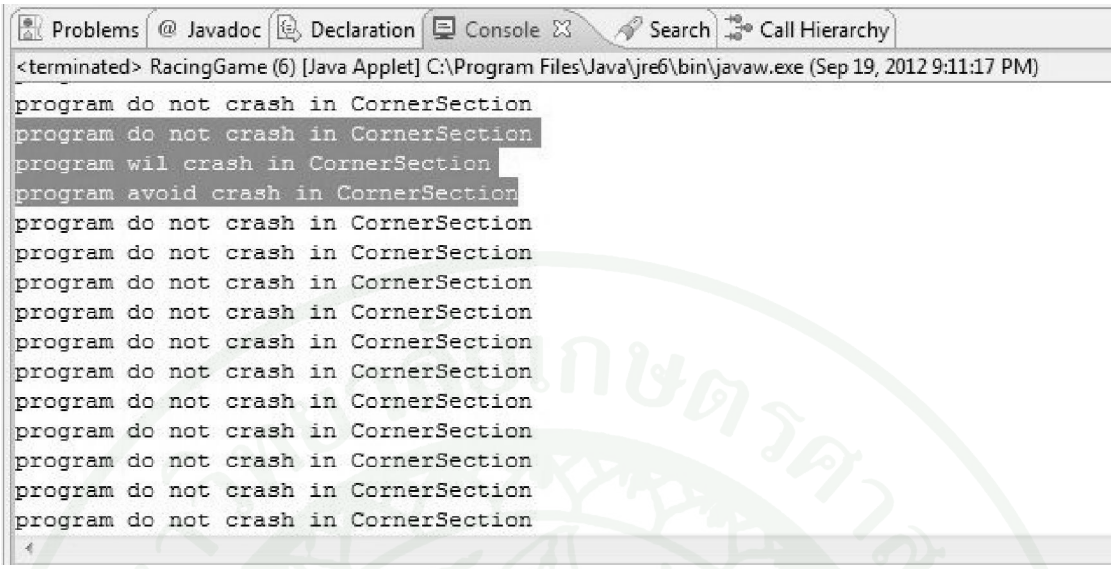
ตารางที่ 14 แสดงการวัดประสิทธิภาพในด้านเวลาในการทำงานของกฎแต่ละชุด

กฎที่ได้จาก	ความสัมพันธ์ทั้งหมด	มีความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7, 0.8
เวลาที่ใช้ (sec)	0.4300	0.0191

จากการทดสอบการทำงานของกฎโดยใช้ข้อมูลทดสอบจำนวน 6,354 ตัวอย่าง ผลที่ได้คล้ายกับการทดสอบกฎโดยใช้ข้อมูลทางตรง คือ จำนวนกฎน้อยลงจะทำให้ใช้เวลาในการตรวจสอบน้อยลง

3. ผลการทดลองในส่วนหลีกเลี่ยงข้อผิดพลาด

ในงานวิจัยได้เลือกวิธีการหลีกเลี่ยงข้อผิดพลาดโดยการย้อนสถานะกลับไปยังสถานะก่อนหน้า โดยเมื่อโปรแกรมกำลังจะเข้าสู่สถานะไม่ปกติ class Checking จะนำข้อมูลที่เก็บใน previousState ซึ่งเป็นสถานะปกติมาใส่ใน currentState ดังนั้นเมื่อเข้าสู่เมท็อด intersects โปรแกรมก็จะประมวลผลข้อมูลที่เป็นสถานะปกติทำให้โปรแกรมอยู่ในสถานะปกติและสามารถทำงานต่อไปได้ เพื่อให้สามารถเห็นความแตกต่างในการเปลี่ยนสถานะได้อย่างชัดเจน ผู้วิจัยได้เขียนโค้ดเพื่อให้โปรแกรมมีการแจ้งเตือนบนหน้าจอ โดยเมื่อโปรแกรมกำลังจะเข้าสู่สถานะไม่ปกติ โปรแกรมจะแจ้งเตือนผู้ใช้บนหน้าจอ ด้วยข้อความ “program will crash in CornerSection” เมื่อโปรแกรมกำลังจะเข้าสู่สถานะไม่ปกติบริเวณเส้นทางโค้ง หรือข้อความ “program will crash in StraightSection” เมื่อโปรแกรมกำลังจะเข้าสู่สถานะไม่ปกติบริเวณเส้นทางตรง และเมื่อมีการหลีกเลี่ยงสถานะไม่ปกติเรียบร้อยแล้ว โปรแกรมจะแจ้งเตือนผู้ใช้บนหน้าจอ ด้วยข้อความ “program avoid crash in CornerSection” เมื่อโปรแกรมหลีกเลี่ยงสถานะไม่ปกติและเข้าสู่สถานะปกติบริเวณเส้นทางโค้งเรียบร้อยแล้ว หรือข้อความ “program avoid crash in StraightSection” เมื่อโปรแกรมหลีกเลี่ยงสถานะไม่ปกติและเข้าสู่สถานะปกติบริเวณเส้นทางตรงเรียบร้อยแล้ว และเมื่อโปรแกรมอยู่ในสถานะปกติ โปรแกรมจะแจ้งเตือนผู้ใช้บนหน้าจอ ด้วยข้อความ “program do not crash in CornerSection” เมื่อโปรแกรมอยู่ในสถานะปกติบริเวณเส้นทางโค้ง หรือข้อความ “program avoid crash in StraightSection” เมื่อโปรแกรมอยู่ในสถานะปกติบริเวณเส้นทางตรงดังแสดงในภาพที่ 26



ภาพที่ 26 แสดงข้อความเมื่อโปรแกรมมีการหลีกเลี่ยงสถานะไม่ปกติ

วิจารณ์

ผลการทดลองแสดงให้เห็นว่าการใช้เบย์เซียนเน็ตเวิร์คในการสร้างกราฟแสดงความสัมพันธ์ระหว่างตัวแปรที่ส่งผลให้ระบบเกิดความผิดพลาดนั้น สามารถนำไปสู่การสร้างกฎเพื่อตรวจสอบข้อผิดพลาดได้อย่างถูกต้อง ผู้วิจัยได้แสดงให้เห็นโดยการแบ่งชุดของกฎที่ได้จากความสัมพันธ์ของกราฟที่ได้จากการทำงานที่มีข้อผิดพลาด ณ เวลาจริง ในขณะที่โปรแกรมกำลังทำงานอยู่ โดยแบ่งกฎออกเป็น 4 ชุด เปรียบเทียบกับการใช้กฎที่ได้ความสัมพันธ์ทั้งหมดจากเบย์เซียนเน็ตเวิร์ค ซึ่งกฎเหล่านี้สามารถตรวจสอบข้อผิดพลาดได้ค่อนข้างถูกต้อง โดยดูจาก Sensitivity และ Specificity ของการทดสอบข้อมูลทดสอบทั้งสองชุด (ข้อมูลทางตรงและทางโค้ง) ซึ่งความถูกต้องของกฎจะขึ้นอยู่กับเกณฑ์ที่ใช้ในการเลือกกฎ ซึ่งในที่นี้ผู้วิจัยได้แบ่งโดยใช้ความสัมพันธ์ที่มีผลเฉลยที่แสดงว่าระบบเกิดการชนและมีความน่าจะเป็นมากกว่า 0.5, 0.6, 0.7, 0.8 ตามลำดับ ซึ่งจากกฎแต่ละชุดจะเห็นว่าจำนวนกฎยิ่งมีมากค่า Sensitivity ก็ยิ่งมีมากขึ้นตามไปด้วย และเมื่อจำนวนกฎลดลงค่า Sensitivity ก็ลดลงเนื่องจากมีบางเหตุการณ์ที่ทำให้เกิดข้อผิดพลาดแต่ไม่ตรงกับเหตุการณ์ในกฎ ดังนั้นกฎจึงไม่สามารถตรวจสอบได้ ทำให้ค่า true positive ซึ่งเป็นค่าที่บอกว่ากฎสามารถระบุเหตุการณ์ที่ทำให้เกิดข้อผิดพลาดได้อย่างถูกต้องมีค่าลดลง ในขณะที่ค่า Specificity มีค่าเพิ่มขึ้นเนื่องจากเมื่อจำนวนกฎลดลงทำให้เหตุการณ์ที่กฎไม่สามารถตรวจจับได้มีเพิ่มขึ้น ซึ่งเหตุการณ์ทั้งหมดจะถูกตรวจสอบและบอกว่าเป็นเหตุการณ์ที่ไม่มีข้อผิดพลาด ซึ่งเหตุการณ์บางส่วนนี้เป็นเหตุการณ์ที่ไม่ส่งผลให้เกิดข้อผิดพลาดจริง ๆ ทำให้ค่า true negative ซึ่งเป็นค่าที่บอกว่ากฎสามารถระบุเหตุการณ์ที่ไม่ทำให้เกิดข้อผิดพลาดได้อย่างถูกต้องมีค่าเพิ่มขึ้นดังในตารางที่ 15-16

จากการทดลองนี้สามารถหาเกณฑ์ที่เหมาะสมที่จะนำมาใช้ในการสกัดกฎเพื่อให้ได้ค่า Sensitivity และค่า Specificity ที่มีค่ามากและจำนวนกฎที่น้อย นั่นก็คือ การเลือกสกัดกฎจากความสัมพันธ์ที่มีผลเฉลยที่แสดงว่าระบบเกิดการชนและมีค่าความน่าจะเป็นมากกว่า 0.5 เนื่องจากมีค่า Sensitivity และค่า Specificity ที่มีค่ามากกว่าและเวลาไม่ได้แตกต่างจากกรณีอื่นมากนัก

ตารางที่ 15 แสดงค่าที่ได้จากการวัดประสิทธิภาพในการเรียนรู้ของข้อมูลทางตรง

Test outcome	All (29,161)	0.5 up (205)	0.6 up (204)	0.7 up (203)	0.8 up (123)
Sensitivity	0.9936	0.9827	0.9823	0.9823	0.1380
Specificity	0.8794	0.9329	0.9329	0.9340	0.9803

ตารางที่ 16 แสดงค่าที่ได้จากการวัดประสิทธิภาพในการเรียนรู้ของข้อมูลทางโค้ง

Test outcome	All (28)	0.5 up (8)	0.6 up (8)	0.7 up (8)	0.8 up (8)
Sensitivity	0.9893	0.9870	0.9870	0.9870	0.9870
Specificity	0.9045	0.9714	0.9714	0.9714	0.9714

สรุปผลและข้อเสนอแนะ

สรุป

งานวิจัยชิ้นนี้นำเสนอแนวทางของการสกัดกฎโดยใช้เบย์เซียนเน็ตเวิร์คในการจำแนกเหตุการณ์ที่จะทำให้เกิดข้อผิดพลาดขึ้นกับระบบ ซึ่งการสกัดกฎแบบนี้จะช่วยลดเวลาที่จะใช้ในการศึกษาและทำความเข้าใจในตัวโปรแกรมที่มีขนาดใหญ่ โดยนักพัฒนาระบบสามารถศึกษาโปรแกรมแบบคร่าวๆ เพื่อใช้ในการเลือกตัวแปรที่จะทำมาใช้และกฎที่ได้ยังครอบคลุมเหตุการณ์ที่นักพัฒนาคาดไม่ถึง ซึ่งเป็นการช่วยโปรแกรมมีการทำงานที่ถูกต้องมากยิ่งขึ้น โดยเบย์เซียนเน็ตเวิร์คสามารถแสดงให้เห็นว่ามีตัวแปรใดบ้างที่ส่งผลและไม่ส่งผลให้เกิดความผิดพลาด ซึ่งจะช่วยให้ลดจำนวนตัวแปรที่จะต้องใช้ลง ทำให้ความสัมพันธ์ที่จะนำมาใช้ในการสกัดกฎมีความถูกต้องแม่นยำและใช้เวลาน้อยลงในการตรวจสอบกฎ โดยได้ทำการทดสอบกับโปรแกรม Racing Game 1.0 (J. Montgomery, 2011) เพื่อแสดงวิธีการที่จะนำมาใช้ในการสกัดกฎสำหรับหลีกเลี่ยงข้อผิดพลาดที่จะเกิดขึ้นในระบบ โดยใช้ความสัมพันธ์และค่าในตารางความน่าจะเป็นแบบมีเงื่อนไขที่ได้จากเบย์เซียนเน็ตเวิร์คมาใช้ในการสกัดกฎ ซึ่งได้มีการแบ่งกฎออกเป็น 4 ชุด และนำไปเปรียบเทียบกับกฎที่ได้จากความสัมพันธ์ทั้งหมดเพื่อหาเกณฑ์ในการสกัดกฎที่ดีที่สุดที่จะทำให้การตรวจสอบระบบมีความถูกต้องและใช้เวลาน้อยที่สุด

จากผลการทดลองที่ได้แสดงให้เห็นว่ากฎที่ได้สามารถนำไปใช้ตรวจสอบระบบและแก้ไขข้อผิดพลาดที่เกิดขึ้นได้อย่างถูกต้องโดยระบบสามารถหลีกเลี่ยงเหตุการณ์ที่อาจจะส่งผลให้เกิดข้อผิดพลาดซึ่งทำให้ระบบสามารถทำงานต่อไปได้

ข้อเสนอแนะ

ในงานวิจัยนี้เป็นการนำเสนอการเรียนรู้เพื่อการสกัดกฎโดยใช้เบย์เซียนเน็ตเวิร์ค โดยเลือกกรณีศึกษาที่มีขนาดเล็ก คือ มีจำนวนตัวแปรที่เกี่ยวข้องน้อยและมีผลลัพธ์ 2 กลุ่ม (binary classification) คือ ปกติและไม่ปกติ แต่ในความเป็นจริงยังมีปัญหาที่มีผลลัพธ์มากกว่าสอง (multiclass classification) ซึ่งเป็นกรณีศึกษาที่มีความซับซ้อนมากกว่า ดังนั้นจึงควรมีการทดสอบกับปัญหาที่มีผลลัพธ์มากกว่าสอง

เอกสารและสิ่งอ้างอิง

บุญเจริญ ศิริเนาวกุล. 2551. **ปัญญาประดิษฐ์ (Artificial Intelligence).** สำนักพิมพ์ท็อป จำกัด. กรุงเทพฯ.

Cagatay Catal. 2011. **Software fault prediction: A literature review and current trends**, pp 4626–4636. In Expert Systems with Applications, Vol. 38, Issue 4.

Angelos D. Keromytis . 2003. **The Case for Self-Healing Software**. Fairfax, VA: IOS Press.

Yasser EL-Manzalawy. 2011. **Aspect Oriented Programming**. Available Source: http://www.developer.com/design/article.php/3308941, October 30, 2011.

Jeffrey G. Blodgett and Ronald D. Anderson. 2000. **A Bayesian Network Model of the Consumer Complaint Process**, pp 321-338. In Journal of Service Research, vol. 2 no. 4.

Debanjan Ghosh, Raj Sharman, H. Raghav Rao and Shambhu Upadhyaya. 2007. **Self-healing systems — survey and synthesis**, pp 2164-2185. In Decision Support Systems in Emerging Economies, Volume 42, Issue 4.

Maggie Hamill, Katerina Goseva-Popstojanova. 2009. **Common Trends in software fault and failure data**. In IEEE transactions on software engineering, vol. 35, no. 4.

Klaus Havelund and Grigore Rosu. 2001. **Java PathExplorer - A Runtime Verification Tool**. In The 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey.

Sarah Hebert, Valerie Lee, Matthew Morabito and Jamie Polan. 2007. **The Michigan Chemical Process Dynamics and Controls Open Text Book Retrieved**. Available Source : https://controls.engin.umich.edu/wiki/index.php/Bayesian_network_theory. January 5, 2012.

Stuart J. Russell and Peter Norving. 2003**. Artificial Intelligence A Modern Approach Second Edition**. Pearson Education, Inc., Upper Saddle River, New Jersey, U.S.A.

Insup Lee, H. Ben-Abdallah, S. Kannan, M. Kim, O. Sokolsky and M. Viswanathan. 1998. **A Monitoring and Checking Framework for Run-time Correctness Assurance**. the Korea-U.S. Technical Conference on Strategic Technologies. Vienna, Austria**.**

Yuyang Liu, Wooi Ping Cheah, Byung-Ki Kim and Hyukro Park. 2008. **Predict Software Failure-prone by Learning Bayesian Network**. In International Journal of Advanced Science and Technology.

Luis M. de Campos, Juan M. Fernindez and Juan F. Huete. 2000. **Building Bayesian Network-Based Information Retrieval Systems**. In 11th International Workshop on Database and Expert Systems Applications. London, U.K.

J. Montgomery. 2011. **Racing Game 1.0**. Available Source: http://website.lineone.net/~john.montgomery/RacingGame/index.html, September 21, 2011.

Sanguk Noh , Gihyun Jung , Kyunghee Choi and Cheolho Lee. 2008. **Compiling network traffic into rules using soft computing methods for the detection of flooding attacks**, pp 1200-1210.In Soft Computing ,Volume 8, Issue 3.

Shunshan Piao, Jeongmin Park and Eunseok Lee. 2007. **Root Cause Analysis and Proactive Problem Prediction for Self-Healing**. In International Conference on Convergence Information Technology. Gyeongui, Korea.

Corsten Riggelsen. 2008. **Approximation methods for Efficient Learning of Bayesian Network**. The author and IOS Press: 5-36.

University of Waikato. 2011. **WEKA**. Available Source: http://www.cs.waikato.ac.nz/ml/weka/., Sep, 2011.

Stefan Wagner. 2009. **A Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models**. In The 5th International Conference on Predictor Models in Software Engineering.

Qianxiang Wang. 2005. **Towards a Rule Model for Self-adaptive Software**. In ACM SIGSOFT Software Engineering. Volume 30 Issue 1. New York, USA.

Giljong Yoo, Jeongmin Park amd Eunseok Lee. 2006. **Hybrid Prediction Model for improving Reliability in Self-Healing System**. In the 4th International Conference on Software Engineering Research Management and Applications. Washington, U.S.A.

Chun Yuan, Ni Lao, Ji-Rong Wen, Jiwei Li, Zheng Zhang, Yi-Min Wang and Wei-Ying Ma. 2006**. Automated Known Problem Diagnosis with Event Traces**. In The 1st ACM SIGOPS/EuroSys European Conference on Computer Systems. Leuven, Belgium.

ภาคผนวก

ภาคผนวก ก

บทความวิทยานิพนธ์ที่ได้รับการตีพิมพ์เผยแพร่

Predicting Behaviors that Lead to Failures using Bayesian Networks

Kanokwan Kaewthong, Usa Sammapun*
Department of Computer Science
Faculty of Science, Kasetsart University
Bangkok, Thailand
kanokwan.kaewthong@gmail.com, fsciusa@ku.ac.th

Abstract

Nowadays, keeping applications running with as few failures as possible is important and necessary. If failures are less, reliability of the applications would likely increase. In an uncontrolled environment, testing is not enough to ensure reliability because it is impossible to test all exhaustive cases. One way to increase reliability is to try to detect possibilities of failures before occurring and try to take different actions to avoid failures. This paper proposes rule extraction for detecting possible failures using a Bayesian Network, a classifier used to find correlation of interesting events where correlation is a conditional probability between the events. In this paper, interesting events represent failures and behaviors that could lead to the failures in the applications. Their correlation is then computed as conditional probabilities which are then transformed into rules of the form IF-THEN statements. Our experiment is done by learning the correlation and extracting rules from a log file of a running application. We evaluate our approach using sensitivity and specificity measures. The results of the experiments show our approach is accurate.

Keywords: fault-avoidance, fault-prediction, runtime verification, Bayesian networks, rule extraction

1. Introduction

The challenging problem in the application development is to find and fix all the faults. Testing can help discover only part of the faults because it is difficult for testers to imagine and test all possible scenarios in all environments. Therefore, no matter how careful and comprehensive the developers and testers are, there can still be faults left in the applications, especially when the applications are used in an untested sequence of operations or in a different environment from which the applications are tested. Therefore, a technique that can help avoid faults or failures after the applications are deployed can increase reliability. Various researches have been done to detect faults and potential failures and/or avoid them such as works in the fields of fault prediction and avoidance [1,2,5,9,14,15,16] and the field of runtime verification [4,7,8,12].

In this paper, we use the words 'fault' and 'failure' in a similar manner as how they are defined in [3] as follows. "A failure is a departure of the system or system component behavior from its required behavior. On the other hand, a fault is an accidental condition, which if encountered, may cause the system or system component to fail to perform as required." From the definition, we define further that a fault is like a root cause of the failure. A fault can produce internal behaviors that could lead to external behaviors that deviate from required behaviors. These external behaviors that deviate from required behaviors can be considered as failures. We also define abnormal states of the application as failures. We use the terms failures and abnormal states interchangeably here.

Researches in the fault prediction/avoidance field are interested in faults while researches in the runtime verification field are more interested in behaviors that lead to failures since failures can be avoided even though faults are present. This paper is more similar to the latter group of researches.

In this latter group of researches, developers need to know about a set of events that lead to normal states and another set of events that can lead to abnormal states. Once we differentiate the two sets, we should be able to detect abnormal internal behaviors and provide alternative actions to avoid potential failures. To determine which events or behaviors lead to normal or abnormal states, we need to have knowledge about the application in terms of relationships between failures and behaviors that could lead to failures so that we can collect or log data such as variable assignments or method invocation appropriately. The relationship can be obtained in two ways: (1) extracting from various sources such as documents, specifications, or experts and (2) using existing learning techniques. In runtime verification, this relationship is presented as logical rules derived from software specification.

One of the problems in runtime verification is how the developer comes up with specific rules in the first place, especially if the document is not available or not easily understood. Our proposed work tries to solve this problem by using learning techniques such as Bayesian Networks to extract the rules from behaviors of the applications. After

* *Corresponding Author*

we learn the rule, our next goal (to be done in our future work) is to use the rule to detect failures and try to avoid the failures before they actually occur.

We evaluate our approach with experiments using Bayesian Networks to determine the relationships and probabilities of failures and events that could lead to such failures. The performance measures which are sensitivity and specificity show Bayesian Networks are quite accurate in determining relationships between failures and events leading to failures. These relationships are then used to generate rules to be used in monitoring a running application.

The remainder of this paper is organized as follows. Sec. 2 provides related work. Sec. 3 describes related theories. Sec. 4 proposes our approach. Sec. 5 describes experiments and analysis. Section 6 presents the conclusion and future work.

2. Related works

We divide related works into two groups which are runtime verification and software fault prediction.

The goal of runtime verification [4,6,8,12] is to verify that a program runs correctly by detecting whether the program reaches abnormal states in a very similar manner as our work. In runtime verification, developers must specify rules in various types of logics such as propositional logics or linear temporal logics to represent behaviors that lead abnormal states. The rules are then transformed into a runtime checker, mostly in the form of a state machine. When the program is executing, the runtime checker is running in parallel and receiving log events from the running program. If the runtime checker detects abnormal states, it will notify the developer or try to avoid the abnormal states. The events collected in runtime verification are usually internal behaviors such as variable assignments or method invocations. The similarities and differences of this field and our work are already stated in Section 1.

There are several existing works in software fault prediction as surveyed by Catal [1]. Catal studied statistical-based and machine learning-based fault prediction work published between 1990 to 2009. The survey shows works in this field use metrics such as a number of methods, McCabe cyclomatic complexity or a combination of many of them with techniques such as logistic regression, classification trees, etc., to predict faults. More specific works using Bayesian Network to predict faults are presented as follows. Yuyang Liu et al. [9] use a suite of software metrics such as a number of method calls, a number of lines of code inside method bodies, and other metrics as parameters for Bayesian Networks to learn whether there could be defects within the module or not. Wagner [14] uses Bayesian Networks to assess and predict software quality by learning which attributes of a system, environments, and an organization such as system redundancy, environment constrained-ness, or personnel turnover could lead to activities such as increase in maintenance effort. This group of work is similar to our work since they use Bayesian Networks or similar learning technique to predict/assess reliability of the system. They differ from our work since they use software metrics instead of internal application events as parameters to the Bayesian Networks.

Other works close to the software fault prediction are presented in this paragraph. Yoo et al. [15] analyze four prediction algorithms: ID3, Fuzzy Logic, Fuzzy Neural Network, and Bayesian Networks, and present an approach to combine the algorithms together to predict error occurrences where parameters to the techniques are system-level rather than application-level data such as CPU usage, memory usage, and bandwidth. Our work uses application-level data such as variable assignments. Piao et al. [11] present learning and analysis to determine root causes of failures using Bayesian Network. Again, parameters used in their paper are system-level performance metrics such as CPU usage. Yuan et al. [16] use an n-gram model and the SVM classification algorithm to analyze the trace log of events and predict root causes of problems. This work is similar to our work since it collects events such as system calls rather than performance or software metrics. However, this work uses different approaches with a different goal from ours.

3. Theories

3.1. Failure detection and avoidance

Our work relies on failure detection and avoidance which are usually done as follows.

Monitoring: While the system is running, the monitor collects the events related to failures.

Analysis and decision: Events are then analyzed to determine whether they could lead to failures using specified or learned rules.

Avoidance or recovery: Once an abnormal state is detected, the fault-avoidance technique will select a possible response mechanism to avoid each possible failure.

3.2. Bayesian Networks

A Bayesian network [6] is a causal graphical modeling that describes the relationship between events that we are interested in. A Bayesian network is represented by a Directed Acyclic Graph (DAG) where nodes represent variables or events and edges represents relationships between events as shown in Fig 1. The process of Bayesian Networks can be divided into two phases: (1) structure learning and (2) parameter learning. In the structure learning, nodes and edges are determined based on dependency between the nodes. Parameter learning quantifies the dependency by computing Conditional Probability Table (CPT), which is a conditional probability distribution for each node in Bayesian Networks.

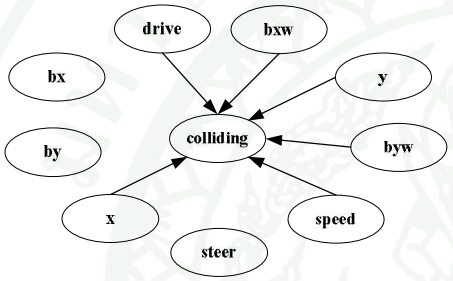


Figure 1. A Bayesian network graph

4. Overview of the our approach

Our goal illustrated in Fig. 2 is to learn causal relationships between failures of an application and events that could be behaviors leading to failures from our collected data. Then, we use the learned relationships to implement rules to monitor the actual application during runtime to detect events that could lead to failures in order to avoid the failures before they actually occur. In this paper, we present the first part of our goal, which is the rule extraction process, to learn causal relationships and build resulting rules to be used in monitoring the application.

Our approach can be divided into four steps as follows. First, we collect and log data where log data contains application-level information such as variable assignment events with their corresponding values and failure events. Second, we use Bayesian Networks to learn relationships and probabilities of the variables. Next, we evaluate the results to validate that Bayesian Network can help determine which behaviors could lead to failures. We use the WEKA tool [17] for the learning process. Lastly, we generate rules from the relationships.

In this paper, a case study of a racing game [10] is used in our experiments to show our approach. Although the application seems trivial, the concept and the result can be easily applied to other more complicated applications. The following sections describe the steps of the rule extraction process in more details. To help illustrate the steps, the case study is described in parallel.

4.1. Data collection

The data collection is a very important step in this paper. To choose which data to collect, we assume developers have general knowledge about the system and know important variables to log but do not know exactly and specifically which variables and which values could lead to failures and thus need to use Bayesian Networks to learn that.

We collect data by manually instrumenting the code with log statements and adding an additional variable *colliding* to keep track of our defined failure state. In our work, we do know what the failures or abnormal states are but we do not know which events could lead to these abnormal states. We thus need to use Bayesian Network to learn these events. Although a collision is not exactly an abnormal state in a racing game application since a collision is in the game logic, we however define collision as abnormal state here so that we could illustrate our concept using this example. As our future work when presenting our avoidance part, a larger case study with a more realistic abnormal state will be used.

In this case study, we collect data from the racing game. This racing game written in Java has a GUI window with a racetrack and a car. A user uses four keys up, down, left, and right on a keyboard to maneuver the car. This keyboard input causes changes to variables such as *drive, steer*, and *speed* where *drive* has three possible values *noKeyPss*, *motorOn* and *reversing*, *steer* has three possible values *noKeyPss*, *steerRight*, and *steerLeft*, and *speed* is an integer with value from 0 to 10. These variables consequently cause changes in the position of the car, which is represented by variables *x* and *y*.

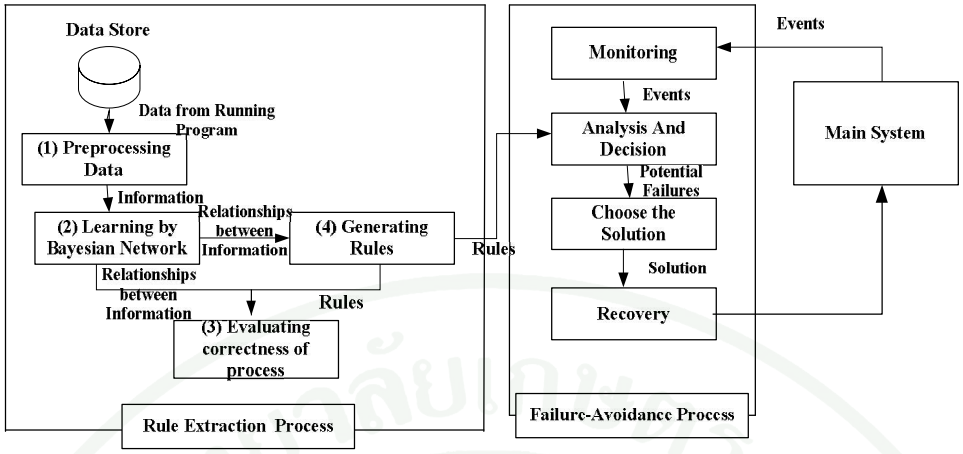


Figure 2. The architecture of fault avoidance system.

We define the abnormal state as a collision with a border of the racetrack where variables *bx* and *by* represent the inner border of the racetrack along the x axis and the y axis respectively, and variables *bxw* and *byw* represent the outer border of the racetrack. The racetrack has a rectangle shape with rounded corners and is implemented with two subclasses of the same superclass representing the straight part and the curvy part of the track. We therefore distinguish between the straight track and the curve track. Both types of tracks contain the same variables described earlier, but the curve track has additional three variables *left*, *top*, and *dis* where *left* and *top* indicate which corner and *dis* represents radius of the curve. Finally, the variable *colliding*, described earlier, with values *Positive* or *Negative* represents the state of the program whether it is in the abnormal state.

4.2 Learning by Bayesian Networks

Our next step is to use the WEKA tool to run Bayesian Networks to determine relationships between variables and the normal/abnormal states and compute the conditional probability table (CPT).

4.3 Evaluation

After the relationships are learned, we need to evaluate such relationships. In this paper, we use sensitivity and specificity as performance measures for our binary classifier. Sensitivity is a probability that a test result will be positive when the failure is present and is equal to true positive / (true positive + false negative). Specificity is a probability that a test result will be negative when the failure is not present and is equal to true negative / (false positive + true negative).

4.4 Generating Rules

From the graph resulting from Bayesian Networks, we choose interesting relationships where their probability values are significant and generate rules from the chosen relationships. To determine which probability values are significant, we have done a simple experiment to compare sensitivity and specificity after using the chosen relationships to find failures instead of all relationships from Bayesian Networks. The chosen relationships have probability values greater than 0.5, 0.6, 0.7, and 0.8. The results of the comparison are described in Section 5. This number can be easily adjusted based on how safe one wants the application to be, and it should be different for different applications. A more sophisticated technique such as statistical testing or fuzzy logic can be used to determine this value.

After we have chosen the relationships, we post-process the relationships by combining some of overlapping or continuing relationships together. For example, consider Table 1. Here, there are three relationships with variables *drive*, *speed*, *bxw*, *byw*, *x, y* and their ranges of values that leads to an abnormal state. The relationships have the same values for *drive*, *speed*, *bxw*, *byw*, and *x* whereas the values for *y* are different but continuing from one relationship to another. Thus, we can combine, without changing the meaning, all three relationships into one where the resulting values for *drive*, *speed*, *bxw*, *byw*, *x*, and *y* are (-inf, 3.5], (-inf, 96], (-inf, 256], (0.5, 11.5], (96.5, 222.5].

The generated rules are in the form of IF-THEN statements as follows: IF <conditions> and THEN <actions to avoid failures> where the IF clause is the chosen relationships to detect failures, and the THEN clause is the second part of our research to avoid failures, not presented in this paper.

Table 1. The examples of relationships of the straight track.

drive	speed	bxw	byw	x	y
motoron	(-inf-3.5]	(-inf-96]	(-inf-256]	(0.5-11.5]	(96.5-109.5]
motoron	(-inf-3.5]	(-inf-96]	(-inf-256]	(0.5-11.5]	(109.5-207.5]
motoron	(-inf-3.5]	(-inf-96]	(-inf-256]	(0.5-11.5]	(207.5-222.5]

Since to avoid failures we need to put the IF-THEN rules within the application, we therefore measure the execution time for the IF clauses to see how much the application would slow down.

5. Experiments

This section presents the experiment of the racing game [10] case study and its results done on a computer with Intel Celeron® 2.20 GHz CPU, 752 MB RAM, 40 GB hard disk. As described earlier, events of user inputs' variable assignments and events of collision are collected as our datasets to be learned by the Bayesian Network using WEKA software. Since the user inputs depend on the structure of the racetrack, we provide separate results for the straight and the curve section of the track. The resulting graph and its nodes probabilities are then used to select interesting correlation and to generate rules.

5.1. Results from straight track data

The dataset used in learning in the experiment consists of 8,579 records. The records can be divided into the actual class of Positive of 4,799 records and the actual class of Negative of 3,780 records where Positive means collision. The dataset used in testing consists of 18,056 records total. The records can be divided into the actual class of Positive of 2,376 records and the actual class of Negative of 15,680 records. The result is presented as True Positive, True Negative, Sensitivity, Specificity and execution time used at the IF clauses of rules as shown in Table 2.

Table 2 shows the performance of learning behaviors leading to failures using Bayesian Networks having all relationships and having some relationships with a probability more than specified thresholds. Here we choose the threshold to be from 0.5 to 0.8. There are 205 rules generated from 0.5 probability value threshold. Similarly, there are 204 rules, 203 rules, and 123 rules for the 0.6, 0.7, and 0.8 probability value thresholds, respectively.

5.2. Results from curve track data.

The dataset used in learning in the experiment for the curve track data consists of 7,611 records. The records can be divided into the actual class of Positive of 4,495 records and the actual class of Negative of 3,116 records where Positive means collision. The dataset used in testing consists of 6,354 records. The records can be divided into the actual class of Positive of 1,314 records and the actual class of Negative of 5,040 records.

The performance of using Bayesian Networks is shown in Table 3. For the curve track data, because all the relationships that lead to collision have probability values more than 0.9, all 0.5-0.8 probability values result in the same number of rules and thus the same values for sensitivity, specificity, and execution time.

5.3. Discussions

The experimental results show that using Bayesian Networks to build a quantified relationship graph between failures and behaviors leading to failures produces quite accurate prediction of behaviors. We have also shown that we can use a subset of resulting relationship graph to build rules. The results of experiments show that when using all relationships and a subset of relationships from Bayesian Networks, we can catch behaviors leading to failures accurately based on high values of sensitivity and specificity. The performance using the subset of rules is based on the thresholds which present tradeoff between accuracy and efficiency. When using all relationships, both sensitivity and specificity are high. The higher the thresholds, the smaller number of rules and the lower the specificity because true positive decreases. The specificity on the other hand increases as the thresholds are higher because true negative is higher. From the experiments we can conclude that we should find the right value of threshold so that there is a balance where both specificity and sensitivity are high and the number of rules is low. We have also done the experiments to evaluate whether the real-time monitoring using the rules is sufficiently fast to not interfere with the underlying system too much.

As a next step, once the failure is predicted, the actions in which to avoid failure such as rolling back to previous normal states will be studied as well. Once the study has been done, we will use an aspect-oriented programming tool to implement the more complete monitoring system. In addition, it would be useful for the error prediction to be dynamic so that new data can be incorporated into learning the relationship between failures and application events.

One question that could be raised is how the learning process could slow down the running application since we need to log and collect data from all potential variables. Yes, the logging would definitely slow down the system. We could lessen the problem by sending data to be logged by another computer given that it is faster than

Table 2. The performance of testing from using straight track data.

Test outcome	All	0.5 up	0.6 up	0.7 up	0.8 up
True Positive	2,361	2,335	2,334	2,334	328
True Negative	13,789	14,629	14,629	14,646	15,372
Sensitivity	0.9936	0.9827	0.9823	0.9823	0.1380
Specificity	0.8794	0.9329	0.9329	0.9340	0.9803
Time (sec)	0.2700	0.1667	0.1510	0.1539	0.1001

Table 3. The performance of testing from using curve track data.

Test outcome	All	0.5 up	0.6 up	0.7 up	0.8 up
True Positive	1,300	1,296	1,296	1,296	1,296
True Negative	4,559	4,897	4,897	4,897	4,897
Sensitivity	0.9893	0.9870	0.9870	0.9870	0.9870
Specificity	0.9045	0.9714	0.9714	0.9714	0.9714
Time (sec)	0.4300	0.0191	0.0191	0.0191	0.0191

writing to its own hard disk. In general, this is the problem of all the research areas that need to monitor systems or applications. Runtime verification is also subject to this problem where most papers state that it is a tradeoff between having correct behaviors or less failures and having slowed-down applications.

6. Conclusions

This paper presents an approach of using Bayesian Networks to classify which events can lead to failures and provides experimental results showing that the classifier is quite accurate. A subset of the resulting relationships is then used to build monitoring rules to detect upcoming failure. Our immediate next step is described in Section 5.3 where the result will be ultimately used to help avoid failures.

As our future work, a larger case study with a more realistic abnormal state will be used. Various studies can also be done such as comparing Bayesian Networks to other binary classifiers. Since the case study presented in this paper results in only two classes of states, we can find case studies that results in various types of errors meaning that a multiclass classifier must be used instead of a binary classifier.

7. Acknowledgment

This research is supported by Department of Computer Science, Faculty of Science, Kasetsart University, Bangkok Thailand.

8. References

[1] C. Catal. Software fault prediction: A literature review and current trends. In Expert Systems with Applications, Vol. 38, Issue 4, 2011. pp 4626–4636.

[2] D. Ghosh, R. Sharman, H. R. Rao and S. Upadhyaya. Self-healing systems — survey and synthesis. In Decision Support Systems in Emerging Economies, Volume 42, Issue 4, Jan 2007. pp 2164-2185.

[3] M. Hamill, K. Goseva-Popstojanova.Common Trends in software fault and failure data. In IEEE transactions on software engineering, vol. 35, no. 4, 2009.

[4] K.Havelund and G. Rosu. Java PathExplorer - A Runtime Verification Tool. In The 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey, 2001.

[5] J. H. Sigurdsson, L. A. Walls and J. L. Quigley. Bayesian belief nets for managing expert judgement and modelling reliability. In Quality and Reliability Engineering International Special Issue: 14th advances in reliability technology symposium (ARTS). Volume 17, Issue 3, May/June 2001. pp 181–190.

[6] S. J. Russell and P. Norving. Artificial Intelligence A Modern Approach Second Edition. Pearson Education, Inc., Upper Saddle River, New Jersey, U.S.A. 2003.

[7] M. Kim, I. Lee, U. Sammapun, J. Shin and O. Sokolsky. Monitoring, Checking, and Steering of Real-Time Systems. In the 2nd International Workshop on Runtime Verification, 2002,

[8] I. Lee, H. Ben-Abdallah, S. Kannan, M. Kim, O. Sokolsky and M. Viswanathan. A Monitoring and Checking Framework for Run-time Correctness Assurance. In the Korea-U.S. Technical Conference on Strategic Technologies,Vienna, VA, October 1998.

[9] Y. Liu, W. Ping Cheah, B. Kim and H. Park. Predict Software Failure-prone by Learning Bayesian Network. In International Journal of Advanced Science and Technology. 2008.

[10] J. Montgomery. 2011. Racing game 1.0. Available Source:http://website.lineone.net/~john.montgomery/RacingGame/index.html., Sep 21, 2011.

[11] S. Piao, J. Park and E. Lee. Root cause analysis and proactive problem prediction for self-healing. In International Conference on Convergence Information Technology (ICCIT 2007), Gyeongui, Korea. 2007.

[12] U. Sammapun, I. Lee and O. Sokolsky. RT-MaC: Runtime Monitoring and Checking of Quantitative and Probabilistic Properties. In 11th IEEE International Conference of Embedded and Real-Time Computing Systems and Applications,. Aug, 2005. pp 147 – 153.

[13] Q. Wang. Towards a rule model for self-adaptive software. In ACM SIGSOFT Software Engineering, New York, USA. Volume 30, Issue 1, Jan 2005.

[14] S. Wagner. A Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models. In the 5th International Conference on Predictor Models in Software Engineering. 2009.

[15] G. Yoo, J. Park and E. Lee. Hybrid prediction model for improving reliability in self-healing system. In the 4th International Conference on Software Engineering Research Management and Applications, 2006.

[16] C. Yuan, N. Lao, J. Wen, J. Li, Z. Zhang, Y. Wang and W. Ma. Automated known problem diagnosis with event traces. In the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, 2006.

[17] Weka. University of Waikato. Available Source: http://www.cs.waikato.ac.nz/ml/weka/., Sep, 2011.

ประวัติการศึกษาและการทำงาน

ชื่อ – นามสกุล	นางสาวกนกวรรณ แก้วทอง
วัน เดือน ปี ที่เกิด	29 มิถุนายน 2530
สถานที่เกิด	อำเภอหาดใหญ่ จังหวัดสงขลา
ประวัติการศึกษา	วท.บ.(สงขลานครินทร์) มหาวิทยาลัยสงขลานครินทร์
ตำแหน่งหน้าที่การงานปัจจุบัน	-
สถานที่ทำงานปัจจุบัน	-
ผลงานดีเด่นและรางวัลทางวิชาการ	-
ทุนการศึกษาที่ได้รับ	-