การสร้างสเตทแมชชีนแบบอัตโนมัติสำหรับการรู้จำตัวอักษรตัวพิมพ์ภาษาไทย

AUTOMATIC STATE MACHINE INDUCTION FOR THAI PRINTED CHARACTER RECOGNITION

ณัฐฉัฐ ปานตระการ
NATTACHAT PANTRAKARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2551

KMITL-2008-EN-M-070-007

การสร้างสเตทแมทชีนแบบอัตโนมัติสำหรับการรู้จำตัวอักษรตัวพิมพ์ภาษาไทย

AUTOMATIC STATE MACHINE INDUCTION FOR THAI PRINTED CHARACTER RECOGNITION

ณัฐฉัฐ ปานตระการ

NATTACHAT PANTRAKARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2551

KMITL-2008-EN-M-070-007

AUTOMATIC STATE MACHINE INDUCTION FOR THAI PRINTED CHARACTER RECOGNITION

NATTACHAT PANTRAKARN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2008

KMITL-2008-EN-M-070-007

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การสร้างสเตทแมทชีนแบบอัตโนมัติสำหรับการรู้จำตัวอักษรตัวพิมพ์ภาษาไทย

Automatic State Machine Induction for Thai Printed Character Recognition

นักศึกษา นางสาวณัฐณัฐ ปานตระการ

รหัสประจำตัว 48060718

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ ผศ.กฤตวัน ศิริบูรณ์

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วัชระ	ฉัตรวิริยะ	
ผศ.เกียรติกูล	เจียรนัยธนะกิจ	
รศ.ดร.ชม	กิ้มปาน	
รศ.ดร.บุญธีร์	เครือตราชู	
ผศ.กฤตวัน	ศิริบูรณ์	

วัน / เดือน/ ปี ที่สอบ 17 ธันวาคม 2550 เวลา 14.00-16.00 น.

สถานที่สอบ ณ อาคาร A ชั้น 5 ห้องประชุม 4

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.รวีวรรณ ชินะตระกูล)

คณบดีบัณฑิตวิทยาลัย

วันที่.....19.....**เดือน**...ธันวาคม...**พ.ศ.**.2551.....

หัวข้อวิทยานิพนธ์	การสร้างสเตทแมทชีนแบบอัตโนมัติสำหรับการรู้จำตัวอักษรตัวพิมพ์ภาษาไทย
นักศึกษา	นางสาวณัฐฉัฐ ปานตระการ
รหัสประจำตัว	48060718
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2551
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ. กฤตวัน ศิริบูรณ์

บทคัดย่อ

วิทยานิพนธ์นี้เสนอวิธีการสร้างสเตทแมทชีนเพื่อเรียนรู้จากสตริงตัวอย่าง สำหรับปัญหาที่มีขอบเขตเฉพาะ ในที่นี้ สตริงจะเป็นลำดับของเชนโค้ด (0 – 7) ที่ได้จากข้อมูลทิศทางในรูปภาพของตัวอักษรตัวพิมพ์ภาษาไทย โดยจะสามารถกำหนดรูปแบบสเตทเป็นสเตทเดี่ยวและคู่สเตท เพื่อให้เหมาะสมกับลักษณะของสตริง รวมทั้งอาศัยคุณลักษณะอื่นๆมาช่วยในการสร้างและการรู้จำ คือ ข้อมูลตำแหน่ง ความยาว และ อินพุตที่เกินของสตริง วิธีการสร้างแบบจำลองที่นำเสนอในวิทยานิพนธ์นี้ มี 3 วิธี ได้แก่ การสร้างแมทชีนจากสตริงที่เป็นบวก หรือ โพสิทีฟเทรนนิ่ง การสร้างแมทชีนจากสตริงที่เป็นลบด้วย หรือ เนกาทีฟเทรนนิ่ง และ การสร้างหลายแมทชีนต่อหนึ่งคลาส หรือ มัลติโมเดล โดยวิธีแรกจะเป็นการสร้างแบบจำลองตามรูปแบบสเตทที่กำหนด มีการใช้ข้อมูลตำแหน่งมาช่วยในการสร้างแบบจำลอง และเก็บค่าสัดส่วนความยาวของอินพุตเพื่อใช้ในการรู้จำ ส่วนวิธีที่สองจะเป็นการใช้อินพุตส่วนเกินที่มีอยู่ในสตริง มาช่วยสร้างสเตทกับดัก เพื่อป้องกันไม่ให้โมเดลยอมรับสตริงที่ไม่ต้องการได้ และวิธีสุดท้ายจะเป็นการแบ่งกลุ่มสตริงออกเป็นกลุ่มย่อยๆ และสร้างแบบจำลองตัวแทนสำหรับแต่ละกลุ่ม ทำให้เกิดแบบจำลองหลายตัวต่อหนึ่งคลาส ผลการรู้จำสำหรับข้อมูลขนาดเล็ก ปรากฏว่า วิธีที่ใช้เนกาทีฟเทรนนิ่งส่งผลให้มีการรู้จำได้ดีกว่าการใช้วิธีโพสิทีฟเทรนนิ่งเพียงอย่างเดียว แต่ไม่เหมาะที่จะนำมาแบ่งกลุ่มสตริงตัวอย่างโดยใช้วิธีมัลติโมเดล เนื่องจากจำนวนสตริงมีน้อยเกินกว่าจะนำมาแบ่งกลุ่มได้ ในขณะที่การทดสอบกับข้อมูลขนาดใหญ่ การใช้วิธีเนกาทีฟเทรนนิ่งมาช่วย กลับให้ผลที่ไม่แตกต่างกับวิธีโพสิทีฟเทรนนิ่งเพียงอย่างเดียวมากนัก แต่วิธีการแบ่งกลุ่มโดยวิธีมัลติโมเดลนั้น ทำให้ประสิทธิภาพการรู้จำดีขึ้นอย่างมาก เนื่องจากมีจำนวนสตริงตัวอย่างมากเพียงพอนั่นเอง

Thesis Title	Automatic State Machine Induction for Thai Printed Character Recognition
Student	Miss Nattachat Pantrakarn
Student ID.	48060718
Degree	Master of Engineering
Program	Computer Engineering
Year	2008
Thesis Advisor	Asst. Prof. Kritawan Siriboon

ABSTRACT

This thesis focus on generating a model in the form of a state machine from training strings. In this case, the string is a sequence of chain code direction (0 - 7), derived from the direction information of printed Thai character's images. The model generalization can be controlled using loops and loop-pair of states. Moreover, other features, such as position, length, and noises, are used to generate models and improve recognition rate. Three state machine induction approaches proposed in this thesis are Positive Training, Negative Training, and Multimodal. In Positive Training, models are generated from training strings as a sequence of loop and loop pair states. The chain code positions are used in this states generating step. The length information is collected for each chain code direction in each state. In Negative Training, the noises in negative-string are used to create trap state to prevent model from accepting non-target strings. In Multimodal, all the training strings are clustered, a model is generated specifically from training strings in a cluster. The recognition results of the small data set show that the Negative Training step improves the accuracy rate from using Positive Training step only, but the Multimodal method is not suitable for small data set. On the contrary for large data set, the recognition rates are about the same whether use the Negative Training or not. Otherwise, the Multimodal method can improve the accuracy rate noticeably, because they have enough data to cluster into several groups.

กิตติกรรมประกาศ

คุณความดีอันใดที่บังเกิดจากวิทยานิพนธ์ฉบับนี้ ขอมอบแด่บิดาและมารดาของผู้วิจัย ผู้ซึ่งได้ให้โอกาสทางการศึกษาและสนับสนุนทั้งแรงกาย แรงใจ เป็นผู้คอยห่วงใย เอาใจใส่ตลอดมา ทำให้ผู้วิจัยได้รับโอกาสที่จะแสวงหาความรู้และพัฒนาตนเอง จนเกิดวิทยานิพนธ์ฉบับนี้ขึ้นมาได้ ผู้วิจัยสำนึกถึงพระคุณในข้อนี้เป็นอย่างสูง

วิทยานิพนธ์ฉบับนี้สามารถประสบความสำเร็จลุล่วงมาได้เป็นอย่างดี โดยได้รับความกรุณาจาก รองศาสตราจารย์ ดร. บุญธีร์ เครือตราชู และผู้ช่วยศาสตราจารย์ กฤตวัน ศิริบูรณ์ อาจารย์ผู้ควบคุมวิทยานิพนธ์ ซึ่งได้แนะแนวทางในการทำวิจัย ทั้งยังให้ความรู้ในข้อทฤษฎีต่างๆและประสบการณ์ที่ได้พบจากการทำวิจัย คอยเอาใจใส่ ให้คำปรึกษาและให้ความช่วยเหลือในเรื่องต่างๆเสมอมา ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณกรรมการสอบวิทยานิพนธ์ทุกท่านที่ได้กรุณาให้คำแนะนำในเรื่องต่างๆ ทั้งวิธีการแก้ไขปัญหาที่เกิดขึ้นในวิทยานิพนธ์และมุมมองความรู้ในด้านอื่นๆ ทำให้ผู้วิจัยมีวิสัยทัศน์ที่กว้างไกลยิ่งขึ้น

ขอขอบพระคุณครูบาอาจารย์ทุกท่าน ที่ประสิทธิ์ประสาทวิชาความรู้ นับตั้งแต่พื้นฐานความรู้วิชาการต่างๆ จนกระทั่งการประยุกต์ใช้ ทำให้ผู้วิจัยมีความรู้ความสามารถในการทำวิจัยให้สำเร็จลงได้ด้วยดี ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณเพื่อนๆ พี่ๆน้องๆทุกคน สำหรับกำลังใจ และความห่วงใยที่มีให้ รวมทั้งความช่วยเหลือในเรื่องต่างๆ ทำให้ผู้วิจัยมีแรงใจที่จะมุมานะพยายามทำวิทยานิพนธ์ให้สำเร็จลงได้

ขอขอบคุณบัณฑิตวิทยาลัย และบัณฑิตศึกษา คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ให้การสนับสนุนในการทำวิทยานิพนธ์ฉบับนี้

สุดท้ายนี้ หากวิทยานิพนธ์ฉบับนี้มีข้อผิดพลาดแต่ประการใด ข้าพเจ้าขอน้อมรับไว้แต่เพียงผู้เดียว

ณัฐฉัฐ ปานตระการ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การแยกแยะความแตกต่างของวัตถุต่างๆนั้น เราอาจจะสามารถทำได้โดยการพิจารณาจากคุณสมบัติประจำตัวของวัตถุนั้นๆที่มีความแตกต่างกัน แต่ยังมีลักษณะของวัตถุบางประเภท ที่สามารถแบ่งกลุ่มได้ยาก เนื่องจากคุณลักษณะนั้นไม่ใช่ค่าเดียว แต่ประกอบด้วยค่าต่อเนื่องเป็นลำดับ

ตัวอย่างของรูปแบบที่พิจารณาลำดับก่อนหลังเป็นสำคัญ (Sequential Feature) เช่น การเรียงลำดับของตัวอักษรภายในคำ เราจะทราบว่าเป็นคำใดก็พิจารณาจากลำดับของตัวอักษรที่เรียงต่อๆกัน คำว่า "การ" คือ อักษร ก-ไก่ ตามด้วย สระอา ตามด้วย ร-เรือ เป็นต้น หรือการเรียงลำดับของโปรตีนภายใน DNA ก็นับเป็นรูปแบบวัตถุที่พิจารณาจากลำดับเช่นกัน

คุณลักษณะลำดับเหล่านี้ จะสามารถแทนวัตถุแต่ละตัวด้วยการนำสัญลักษณ์ (Symbol) มาเรียงต่อๆกันเข้า ก็จะเรียกลำดับของสัญลักษณ์หรืออินพุตนั้นว่า สตริง (String) อย่างเช่น สตริงของคำในภาษาไทย อาจจะเป็น "ความสามารถ" ซึ่งเป็นการนำพยัญชนะ สระ และวรรณยุกต์มาประกอบกันให้เป็นคำที่ถูกต้องนั่นเอง

นอกจากนี้ เรายังสามารถนำข้อมูลของวัตถุอื่นๆมาแปลงให้อยู่ในรูปแบบสตริงเช่นเดียวกันนี้ได้อีกด้วย อย่างเช่น การเขียนตัวอักษรภาษาไทย เราอาจจะแทนด้วยลำดับทิศทางของเส้นอักขระในการเขียนตัวอักษรให้กลายเป็นสตริง เช่น เขียนไปทางทิศซ้าย ขวา ขึ้นบน หรือลงล่าง ซึ่งทำให้ลำดับนี้ถือเป็นตัวแทนข้อมูลของตัวอักษรตัวนั้นๆไป

ปัญหาที่เกิดขึ้น ก็คือ เราจะทำอย่างไรจึงจะสามารถแยกแยะความแตกต่างของสตริงแต่ละตัวได้ นั่นคือ เราจะต้องสร้างโมเดลขึ้นมาเป็นตัวแทนของกลุ่มสตริงที่มีลักษณะใกล้เคียงกัน หรือที่เราเรียกว่า คลาส (Class) ซึ่งเป็นสตริงที่มีลำดับของอินพุตคล้ายคลึงกันนั่นเอง การสร้างโมเดลเป็นตัวแทนของแต่ละคลาสของสตริง อาจทำได้หลายวิธีด้วยกัน วิธีหนึ่งที่เป็นที่นิยม ก็คือ การสร้างสเตทแมชชีน (State Machine) หรือออโตมาตา (Automata) เพื่อแสดงลำดับของอินพุตแต่ละตัวภายในสตริงนั้นๆ

เมื่อได้โมเดลเป็นตัวแทนของแต่ละคลาสแล้ว ซึ่งโมเดลนี้ควรจะมีความสามารถในการยอมรับสตริงที่มีลักษณะใกล้เคียงกันกลุ่มสตริงต้นแบบได้ด้วย แม้ว่าจะเป็นสตริงที่ไม่ทราบค่ามาก่อนก็ตาม การแยกแยะจัดกลุ่มสตริงให้ถูกต้อง เรียกว่า การรู้จำ (Recognition) ซึ่งทำให้เราสามารถทราบได้ว่า วัตถุหรือสตริงเป้าหมายนั้น ควรจะเรียกว่าอะไร

งานวิจัยที่จะกล่าวถึงในวิทยานิพนธ์ฉบับนี้ จะเป็นการพัฒนาวิธีการสร้างสเตทแมทชีนเพื่อเป็นตัวแทนของสตริงที่ได้จากปัญหาที่มีขอบเขตเฉพาะ โดยจะยกสตริงที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทยเป็นกรณีศึกษา เมื่อเราทราบลักษณะของสตริงที่จะนำมาใช้สร้างโมเดลแล้ว ก็จะสามารถกำหนดรูปของสเตทแมทชีนให้เป็นไปตามต้องการได้ ซึ่งทำให้ได้โมเดลที่มีความเหมาะสมกับลักษณะของปัญหามากขึ้น

สตริงที่นำใช้ในการศึกษาวิจัยนั้น จะได้มาจากลำดับของทิศทางของเส้นขอบในอักขระ หรือเรียกว่า เชนโค้ด (Chain Code) จึงสามารถกำหนดรูปแบบของสเตทแมทชีนที่จะมารับสตริงเหล่านี้ได้ เป็น 2 ลักษณะ คือ รูปแบบที่รับสตริงที่ได้จากทิศทางเดิมซ้ำๆ และรูปแบบที่รับสตริงที่ได้จากทิศทางที่อยู่ระหว่างสองทิศที่อยู่ติดกัน [1] นอกจากนั้น โมเดลยังต้องมีลักษณะทั่วไป (Generalization) มากพอที่จะยอมรับสตริงที่ใกล้เคียงกัน แต่ต้องมีลักษณะจำเพาะเจาะจงพอที่จะไม่ยอมรับสตริงที่ต่างกลุ่มหรือต่างคลาสกัน ไปพร้อมๆกันด้วย

เป้าหมายของงานวิจัยนี้ จึงเป็นการพัฒนาวิธีการโมเดลในรูปแบบของสเตทแมทชีน ซึ่งมีลักษณะเหมาะสมกับปัญหาที่จำเพาะเจาะจง โดยโมเดลที่ได้จะต้องมีความสามารถในการยอมรับสตริงที่อยู่ในคลาสเดียวกัน และปฏิเสธสตริงที่กลุ่มต่างคลาสได้ โมเดลควรจะมีรูปแบบที่เข้าใจได้ง่าย และใช้เวลาน้อยในขั้นตอนการสร้างและการรู้จำ

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เพื่อศึกษาและทำความเข้าใจในการรู้จำสตริงโดยใช้วิธีการต่างๆในงานวิจัยก่อนหน้านี้
2. เพื่อนำกลวิธีการรู้จำสตริงมาประยุกต์ใช้กับการรู้จำตัวอักษรได้อย่างมีประสิทธิภาพ
3. เพื่อคิดค้นและพัฒนาวิธีการสร้างโมเดลในรูปแบบสเตทแมทชีนจากสตริงตัวอย่าง เพื่อใช้ในการรู้จำ ในที่นี้ จะศึกษาการรู้จำสตริงที่ได้จากเชนโค้ดของตัวอักษรตัวพิมพ์ภาษาไทย
4. ทำการวิเคราะห์เปรียบเทียบข้อดีข้อเสีย ระหว่างวิธีที่ได้พัฒนาขึ้นในงานวิจัยกับวิธีที่มีการนำเสนอไปแล้ว
5. เพื่อเป็นแนวทางในการศึกษาวิจัยทางด้านการรู้จำ และการเรียนรู้จากสตริงเพื่อสร้างโมเดลเป็นตัวแทนของกลุ่มตัวอย่างต่อไปในอนาคต

1.3 สมมติฐานของการศึกษา

เราสามารถสร้างโมเดลเพื่อเป็นตัวแทนของกลุ่มสตริงตัวอย่างได้ ในงานวิจัยนี้ จะเป็นการสร้างสเตทแมทชีนในรูปแบบดีเทอร์มินิสติคไฟไนท์ออโตมาตา (Deterministic Finite Automata) หรือ DFA เพื่อเป็นตัวแทนของสตริงที่เข้ารหัสแบบเชนโค้ด (Chain Code) 8 ทิศทางของตัวอักษรตัวพิมพ์ภาษาไทย ซึ่ง DFA ที่ได้ จะสามารถยอมรับ (Accept) สตริงที่อยู่ในคลาสเดียวกันได้ และ

ปฏิเสธ (Reject) สตริงอื่นๆที่ไม่ได้อยู่คลาสเดียวกันได้ด้วย นอกจากนี้ เราสามารถกำหนดรูปแบบของสเตทภายใน DFA ให้มีลักษณะเป็นไปตามต้องการได้ ซึ่งจะทำให้ DFA มีความเหมาะสมกับสตริงมากขึ้น รวมทั้งทำให้รูปแบบของสเตทมีลักษณะทั่วไป (Generalization) มากพอที่จะสามารถรู้จำสตริงที่มีลักษณะใกล้เคียงกันได้ เพื่อระบุค่าของสตริงนั้นๆว่ามาจากตัวอักษรอะไร โดยสตริงที่นำมาเป็นตัวอย่างนั้น ควรจะเป็นสตริงที่มีลักษณะทั่วไป (Generalization) สูง คือ มีจุดร่วมที่เหมือนกัน ไม่มีรูปแบบที่แตกต่างกันมากเกินไป เพื่อให้สามารถสร้างโมเดลที่เป็นตัวแทนของกลุ่มได้

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

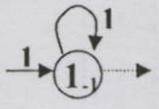
เทคนิคการสร้างโมเดลเพื่อเป็นตัวแทนของกลุ่มสตริงสามารถทำได้หลายรูปแบบด้วยกัน ซึ่งวิธีหนึ่งที่มีการศึกษาวิจัยกัน คือ การสร้างสเตทแมทชีนหรือออโตมาตาเป็นตัวแทนของสตริงแต่ละคลาส โดยอาศัยการเรียนรู้จากสตริงตัวอย่าง ซึ่งจะสร้างเป็นออโตมาตาได้หลายประเภท เช่น สร้างเป็น Deterministic Finite Automata หรือ DFA (DFA Learning) อย่างวิธี Trakhtenbrot-Barzdin [2], SAGE [3], EDSM [4] หรือ ALGERIA [5] เป็นต้น หรือสร้างเป็น Probabilistic Finite Automata หรือ PFA (PFA Learning) อย่างวิธี Hidden Markov Model (HMM) [6] เป็นต้น

งานวิจัยนี้ จะทำการสร้างสเตทแมทชีนหรือออโตมาตาในรูปแบบของ DFA จากกลุ่มสตริงตัวอย่าง โดยอาศัยหลักการสร้างโมเดลตามงานวิจัยของนางสาวปองเกษม [1] ซึ่งจะมีลักษณะเด่น คือ ไม่จำเป็นต้องใช้การคำนวณเหมือนดังเช่นการสร้าง PFA ที่ต้องมีการหาค่าความน่าจะเป็น อาศัยเพียงลำดับของสตริงเพียงอย่างเดียว ซึ่งสามารถเข้าใจและเรียนรู้ได้ง่าย ทั้งยังใช้เวลาน้อยในการสร้างโมเดลและการรู้จำ

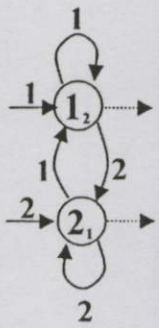
สตริงตัวอย่างที่จะนำมาทดสอบการสร้างโมเดล จะใช้สตริงที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทย โดยจะทำการเข้ารหัสรูปภาพของตัวอักษรแต่ละตัวให้กลายเป็นสตริงของเชนโค้ด (Chian Code) ซึ่งจะมี 8 ทิศทาง ตามการวนขอบของตัวอักษร

การสร้างสเตทแมทชีนในงานวิจัยนี้ จะมีการกำหนดรูปแบบของสเตทให้เหมาะสมกับลักษณะของปัญหาตาม [1] โดยจะเริ่มจากพิจารณาลักษณะของสตริงตัวอย่าง ในที่นี้ คือ สตริงที่ได้จากทิศทางของขอบอักขระในตัวอักษรภาษาไทย จึงแยกพิจารณาได้เป็น 2 กรณี ดังนี้

1. ส่วนของสตริงที่มาจากทิศทางเดียวกันซ้ำๆ เช่น 1, 11, 111, 1111111, ... หรือ 1* สามารถแทนด้วยสัญลักษณ์



2. ส่วนของสตริงที่มาจากทิศทางที่อยู่ระหว่างสองทิศที่ติดกัน เช่น 111222, 1112221122, 1221122, ... หรือ (1*2*)* แทนด้วยสัญลักษณ์



จากนั้น สร้างรูปแบบสเตทสำหรับสตริงทั้ง 2 กรณีดังกล่าว โดยจะต้องมีลักษณะทั่วไป (Generalized) มากพอที่จะยอมรับสตริงที่เข้ารูปแบบใกล้เคียงกันได้ด้วย

โมเดลที่ได้ควรจะสามารถยอมรับสตริงที่อยู่ภายในคลาสเดียวกันได้ และปฏิเสธสตริงจากกลุ่มอื่นๆได้ด้วย ซึ่งในงานวิจัยเดิม จะสร้างโมเดลจากตัวอักษรเป้าหมายเท่านั้น โดยไม่สามารถปฏิเสธสตริงจากคลาสอื่นๆได้ จึงมีแนวคิดเพิ่มเติมที่จะสร้างโมเดลที่สามารถป้องกันไม่ให้โมเดลยอมรับสตริงจากคลาสอื่นได้ โดยการนำสตริงของคลาสนั้นๆ (Positive Sample) และสตริงที่อยู่นอกคลาสเป้าหมาย (Negative Sample) มาประกอบกันเพื่อใช้ในการสร้างโมเดล

ในขั้นแรก จะทำการสร้างโมเดลจากสตริงของคลาสเป้าหมายทีละสตริง โดยจะสร้างสเตทตามรูปแบบที่กำหนดตาม [1] เพื่อใช้แทนอินพุตแต่ละตัวภายในสตริง แล้วนำสตริงตัวถัดไปมาเทียบเข้ากับโมเดลที่สร้างขึ้น และค่อยๆเพิ่มสเตทหรือทรานสิชั่นเข้าไป โดยพยายามจะใช้สเตทและทรานสิชั่นที่มีอยู่เดิมให้มากที่สุด ทั้งนี้ เพื่อให้โมเดลสามารถเพิ่มสเตทและทรานสิชั่นได้เหมาะสม จึงได้เพิ่มข้อมูลตำแหน่งของอินพุตในสตริงเข้ามาช่วยกำหนดตำแหน่งสเตทที่ถูกต้องด้วย

ในขั้นที่สอง เพื่อจะทำให้โมเดลสามารถปฏิเสธสตริงอื่นๆที่ไม่ต้องการได้ด้วย จึงใช้วิธีการสร้างสเตทกับดัก (Trap State) ขึ้นมา และสร้างทรานสิชั่นในตำแหน่งที่อาจจะทำให้โมเดลไม่ยอมรับสตริงที่เราไม่ต้องการไปยังสเตทกับดักนั้น ซึ่งจะกันสตริงที่มีลำดับของอินพุตไม่ถูกต้องไม่ให้ไปถึงสเตทจบ (Final State) ได้

เมื่อได้โมเดลสำหรับแต่ละคลาสแล้ว ก็จะทำการรู้จำโดยการพาร์สสตริงเข้าไปใน DFA แต่ละตัว หากโมเดลนั้น สามารถยอมรับสตริงได้ แสดงว่าสตริงนั้นเป็นสตริงที่อยู่กลุ่มเดียวกับโมเดล โดยในขั้นตอนการรู้จำ นอกจากสตริงจะต้องสิ้นสุดที่สเตทสุดท้าย (Final State) แล้ว ยังได้อาศัยข้อมูลอื่นๆในการรู้จำ เช่น Noise ที่อยู่ในสตริง หรือตำแหน่งและความยาวที่ผิดเพี้ยนไป เพื่อใช้ในการเลือก DFA ในกรณีที่สตริงนั้นๆสามารถไปถึงสเตทจบ (Final State) ได้มากกว่า 1 โมเดล โดยถ้ามีความผิดพลาดต่างๆนี้น้อย ก็จะมีโอกาสเป็นผู้ชนะมากขึ้นนั่นเอง

1.5 ขอบเขตการวิจัย

งานวิจัยนี้จะทำการศึกษาและพัฒนาวิธีการสร้างโมเดลเพื่อการรู้จำสตริง โดยกำหนดขอบเขตปัญหาให้เฉพาะเจาะจง ซึ่งจะมีขอบเขตการวิจัย ดังนี้

1. งานวิจัยนี้เป็นการสร้างสเตทแมทชีนในรูปแบบดีเทอร์มินิสติคไฟไนท์ออโตมาตา (Deterministic Finite Automata: DFA) เพื่อใช้ในการรู้จำสตริงที่มีลักษณะเฉพาะ
2. สตริงที่ใช้งานวิจัยนี้ จะเป็นสตริงที่แทนตัวอักษรตัวพิมพ์ภาษาไทยแบบออฟไลน์ โดยการแปลงรูปภาพตัวอักษรให้เป็นสตริง และเข้ารหัสสตริงในแบบเชนโค้ด ซึ่งได้จากการวนเส้นขอบของอักขระ
3. สตริงนำมาเป็นกลุ่มตัวอย่าง จะผ่านกระบวนการเข้ารหัสแบบเชนโค้ด โดยมี 8 ทิศทาง
4. ตัวอักษรภาษาไทยที่นำมาใช้เป็นกลุ่มตัวอย่าง มีทั้งหมด 77 ตัวอักษร ประกอบด้วย 0 1 2 3 4 5 6 7 8 9 ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ฤ ล ฦ ว ศ ษ ส ห ฬ อ ฮ ฯ ั า ิ ี ึ ื ุ ู เ โ ใ ไ ๅ ๆ ็ ่ ้ ๊ ๋ ์
5. ทำการเปรียบเทียบกับการสร้างโมเดลในการรู้จำวิธีอื่น พร้อมทั้งวิเคราะห์ข้อดีข้อเสีย ในที่นี้ จะเปรียบเทียบกับวิธี Hidden Markov Model (HMM)
6. ศึกษาการทดสอบกับสตริงแบบอื่นๆ เช่น สตริงที่ได้จากลายมือเขียนภาษาไทย และวิเคราะห์ผล

1.6 ขั้นตอนของการศึกษา

1. ศึกษาทฤษฎีและความรู้พื้นฐานที่เกี่ยวข้องกับการสร้างโมเดลในรูปแบบของออโตมาตา
2. ศึกษาวิธีการสร้างออโตมาตาหรือสเตทแมทชีนเพื่อเป็นตัวแทนสตริง โดยการเรียนรู้จากกลุ่มตัวอย่าง รวมทั้งงานวิจัยที่เกี่ยวข้อง ซึ่งมีผู้นำเสนอมาก่อนหน้านี้
3. พัฒนาวิธีการสร้างโมเดลจากกลุ่มตัวอย่าง เพื่อใช้ในการรู้จำ โดยนำแนวคิดบางส่วนจากงานวิจัยอื่นๆมาประยุกต์ใช้ ประกอบกับแนวความคิดที่พัฒนาขึ้นเอง
4. ทดลองสร้างโมเดลหรือสเตทแมทชีนตามแนวคิดที่วางเอาไว้ และวัดผลการรู้จำของโมเดลที่สร้างขึ้น
5. ปรับปรุงโมเดลให้มีความสามารถในการรู้จำเพิ่มขึ้น โดยการปรับข้อกำหนดต่างๆ พร้อมทั้งวัดผลการรู้จำและวิเคราะห์ผลที่ได้ เพื่อนำมาพัฒนาต่อไป
6. ทดลองเปรียบเทียบผลการรู้จำกับวิธีอื่นๆ เพื่อศึกษาข้อดีข้อเสียของการสร้างโมเดลด้วยวิธีต่างๆ
7. สรุปผลการสร้างโมเดลด้วยวิธีที่พัฒนาขึ้น พร้อมวิเคราะห์ผลที่ได้
8. จัดทำเอกสารประกอบวิทยานิพนธ์

บทที่ 2
นิยามและทฤษฎีพื้นฐานที่เกี่ยวข้อง

ในบทนี้ จะกล่าวถึงนิยามและทฤษฎีพื้นฐาน ที่เกี่ยวข้องกับการสร้างออโตมาตา จากสตริงตัวอย่าง ทั้งสัญลักษณ์และนิยามทางคณิตศาสตร์ของสตริงและออโตมาตาในรูปแบบต่างๆ รวมทั้งทฤษฎีอื่นๆที่ใช้ในงานวิจัย

2.1 สตริง (String)

2.1.1 นิยามทั่วไป

นิยาม 2.1 : กำหนดให้ Σ เป็นเซตจำกัด (Finite Set) ที่ไม่ใช่เซตว่างของสัญลักษณ์ที่เรียกว่า Alphabet แล้ว สตริง (String) คือ ลำดับจำกัด (Finite Sequence) ของสัญลักษณ์ใน Σ

สตริงว่าง (Empty String) คือ สตริงที่มีจำนวนสัญลักษณ์เป็นศูนย์ แทนด้วยสัญลักษณ์ λ และความยาวสตริง (Length) เป็นจำนวนสัญลักษณ์ที่เกิดขึ้นภายในสตริง สตริง ω ใดๆ จะมีความยาวสตริงเป็น $|\omega|$ ดังนั้น $|\lambda| = 0$

กำหนดให้ Σ^k แทนเซตของสตริงที่มีความยาว k เมื่อสัญลักษณ์แต่ละตัวในสตริงอยู่ใน Σ ตัวอย่างเช่น ให้ $\Sigma = \{0,1\}$ จะได้ว่า $\Sigma^0 = \{\lambda\}$, $\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00,01,10,11\}$, ... และเซตของสตริงทั้งหมดที่เกิดขึ้นบน Σ เรียกว่า Σ^* โดยที่ $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ...$

2.1.2 สตริงที่เป็นบวกและลบ (Positive and Negative String)

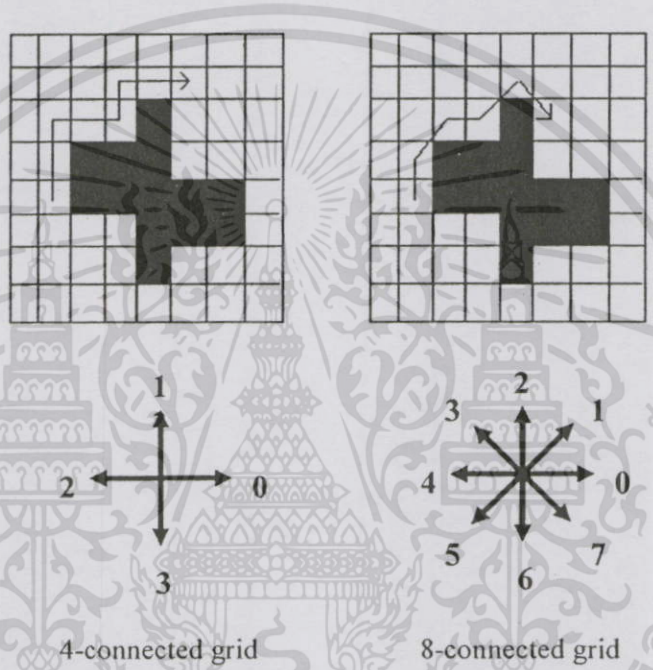
ให้ S_+ และ S_- เป็นเซตที่ไม่ Intersect กัน ซึ่งต่างเป็นซับเซตของ Σ^* จะเรียก S_+ ว่า สตริงที่เป็นบวก (Positive Training Set) และเรียก S_- ว่า สตริงที่เป็นลบ (Negative Training Set) โดย S_+ จะเป็นเซตของสตริงใดๆบน alphabet Σ ซึ่งยอมรับ (Accept) โดย DFA M หรือกล่าวได้ว่า เป็นสตริงภายในคลาสที่เราต้องการนั่นเอง

ส่วน S_- จะเป็นเซตของสตริงใดๆบน alphabet Σ ซึ่งปฏิเสธ (Reject) โดย DFA M หรือกล่าวได้ว่า เป็นสตริงอื่นๆที่อยู่นอกเหนือจากคลาสที่เราต้องการ

ดังนั้น เซตของสตริงทั้งหมด คือ $S = S_+ \cup S_-$ ซึ่งแมทชีนที่ได้จะยอมรับสตริงที่เป็นบวกและปฏิเสธสตริงที่เป็นลบไปพร้อมๆกัน ตัวอย่างเช่น ถ้าให้ S เป็นเซตของสตริงที่มาจากตัวอักษรภาษาไทย และ S_+ เป็นสตริงที่ได้จากอักษร “ก” แล้ว S_- จะเป็นสตริงที่ได้จากตัวอักษรอื่นๆที่ไม่ใช่ “ก” หรือก็คือ อักษร “ข” – “ฮ” รวมทั้งสระ วรรณยุกต์อื่นๆ เป็นต้น

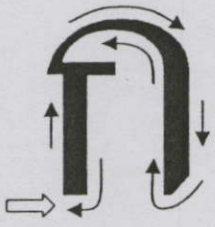
2.1.3 เชนโค้ด (Chain Code)

เชนโค้ด (Chain Code) เป็นวิธีการที่จะพยายามแทนรูปภาพให้อยู่ในรูปแบบสตริง ซึ่งเดิมรูปภาพจะอยู่ในรูปโคออดิเนต $[(x_1, y_1), (x_2, y_2), \ldots]$ เชนโค้ดอาจจะเป็นได้ทั้งจุดแต่ละจุดในพิกเซล (Pixel) หรือเป็นเส้นขอบของพิกเซลก็ได้ พิกเซลจะมีลักษณะเป็นกริด (Grid) รูปสี่เหลี่ยมซึ่งเราสามารถกำหนดให้มีทิศทางโดยรอบ 4 จุด หรือ 8 จุด ก็ได้ จากรูปที่ 2.1 จะเป็นการวนไปตามขอบของรูปสีดำ ซึ่งจะมีเชนโค้ดเป็น 1100100... สำหรับ 4-connected grid และ 21017... สำหรับ 8-connected grid



รูปที่ 2.1 แสดงเชนโค้ดที่ใช้แทนเส้นขอบของรูปภาพ

ในงานวิจัยนี้ จะใช้สตริงที่ได้จากตัวอักษรภาษาไทย ดังนั้น เราจะนำอักขระตัวพิมพ์ภาษาไทยแต่ละตัวนำมาแสกนเป็นรูปภาพ จากนั้นนำมาเข้ารหัสเป็นเชนโค้ดแบบ 8 ทิศทางข้างต้น โดยการวนเส้นขอบของรูปภาพอักขระ เริ่มจากมุมซ้ายล่างวนไปจนบรรจบกับจุดเริ่มต้นอีกครั้งหนึ่ง รูปที่ 2.2 แสดงตัวอย่างเชนโค้ดที่ได้จากอักษร “ก”



```
22222222222222444212121111100101007
00770767677667666666666666655554
22222222222222223323234334344554
56556560000066444466666666666444
```

รูปที่ 2.2 แสดงตัวอย่างเชนโค้ดของตัวอักษร “ก”

จะเห็นว่า สตริงในงานวิจัยนี้ ก็คือเชนโค้ดนั่นเอง ซึ่งจะเป็นสตริงที่มี Alphabet เป็นสัญลักษณ์ 0 – 7 ตามทิศทางของเชนโค้ดแบบ 8-connected grid และแต่ละคลาสของสตริง คือ ตัวอักษรแต่ละตัว เช่น คลาสของอักษร ก-ไก่, คลาสของอักษร ข-ไข่, ... ดังนั้น เราสามารถพิจารณาการรู้จำตัวอักษรภาษาไทย ให้อยู่ในรูปแบบการรู้จำสตริงทั่วไปได้

2.2 ออโตมาตา (Automata)

ออโมมาตา (Automata) หรือสเตทแมทชีน (State Machine) นั้น มีหลากหลายประเภทด้วยกัน ขึ้นอยู่กับองค์ประกอบและความสามารถในการยอมรับสตริง แต่ในที่นี้ จะขอกล่าวถึงเพียง 3 ประเภท ที่ได้มีการกล่าวถึงในวิทยานิพนธ์นี้ และนำมาใช้ประกอบในงานวิจัย สำหรับรายละเอียดเกี่ยวกับออโตมาตาประเภทต่างๆ สามารถอ่านได้จาก [7], [8]

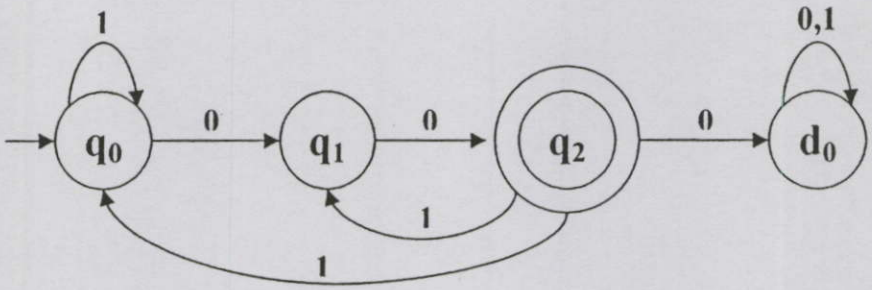
2.2.1 Deterministic Finite Automata (DFA)

นิยาม 2.2 : Deterministic Finite Automata (DFA) คือ แมทชีน $M = (\Sigma, Q, \delta, q_0, F)$ โดย

- Σ เป็นเซตของสัญลักษณ์ที่เป็นอินพุต (Alphabet)
- Q คือ เซตของสเตท
- $\delta : Q \times \Sigma \to Q$ คือ ทรานสิชั่นฟังก์ชั่น (Transition Function)
- $q_0 \in Q$ เป็นสเตทเริ่มต้น
- $F \subseteq Q$ เป็นเซตของ Final State

สำหรับสเตท $d_0 \in Q$ ซึ่ง $\forall a \in \Sigma, \delta(d_0, a) = d_0$ จะเรียกว่า สเตทกับดัก (Trap State) ซึ่งเป็นสเตทที่มีแต่ทรานสิชั่นเข้ามายังสเตทนั้น แต่ไม่มีทรานสิชั่นออกไปยังสเตทอื่นๆเลย

ตัวอย่าง Transition Diagram ของ DFA แสดงในรูปที่ 2.3 ซึ่งจะสามารถรู้จำสตริงที่มีอินพุตที่ลงท้ายด้วย "00" (ศูนย์สองตัวติดกัน) แต่ไม่ยอมรับสตริงที่มี "000" (ศูนย์สามตัวติดกัน) อยู่ในภายในสตริง เมื่อ $\Sigma = \{0,1\}$ โดยแต่ละโหนดจะแทนสเตท สเตทเริ่มต้น (Starting State) แทนด้วยสเตทที่มีลูกศรชี้เข้า ในรูปคือ สเตท q_0 สเตทสุดท้าย (Final State) แทนด้วยโหนดที่มีวงกลมซ้อนกันสองวง จากรูป ได้แก่ สเตท q_2 ส่วนลูกศร (Arc) จะแทนทรานสิชั่น (Transition) จากสเตทหนึ่งไปอีกสเตทหนึ่ง และ Alphabet ที่ทำให้เกิดทรานสิชั่นนั้นๆจะเขียนอยู่บนลูกศร



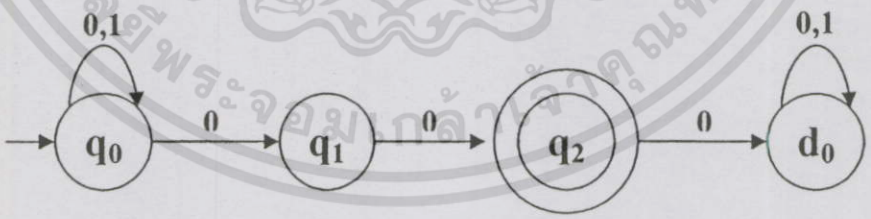
รูปที่ 2.3 แสดงตัวอย่างของ Deterministic Finite State Automata (DFA)

2.2.2 Non-Deterministic Finite Automata (NFA)

นิยาม 2.3 : Non-Deterministic Finite Automata (NFA) คือ แมทชีน $M = (\Sigma, Q, \delta, q_0, F)$ โดย

- Σ เป็นเซตของสัญลักษณ์ที่เป็นอินพุต (Alphabet)
- Q คือ เซตของสเตท
- $\delta : Q \times \Sigma \to P(Q)$ คือ ทรานสิชั่นฟังก์ชั่น (Transition Function)
- $q_0 \in Q$ เป็นสเตทเริ่มต้น
- $F \subseteq Q$ เป็นเซตของ Final State

เราอาจกล่าวได้ว่า NFA คล้ายคลึงกับ DFA เพียงแต่ NFA จะสามารถมีหลายทรานสิชั่นสำหรับอินพุตตัวเดียวกันได้ และสำหรับทุกๆ NFA จะมี DFA ที่สามารถยอมรับสตริงในภาษาเดียวกันได้ เราสามารถแปลง NFA ให้เป็น DFA ได้ รูปที่ 2.4 จะแสดงตัวอย่าง Transition Diagram ของ NFA ซึ่งยอมรับสตริงชุดเดียวกันกับตัวอย่าง DFA ในรูปที่ 2.3



รูปที่ 2.4 แสดงตัวอย่างของ Non-Deterministic Finite State Automata (NFA)

2.2.3 Probabilistic Finite Automata (PFA)

Probablilistic Finite Automata (PFA) เป็นออโตมาตาที่นำความน่าจะเป็นมาใช้ โดยทรานสิชั่นจะมีความน่าเป็นในการข้ามจากสเตทหนึ่งไปอีกสเตทหนึ่ง และที่สเตทเริ่มต้น ก็จะมีเวกเตอร์ความน่าจะเป็น (Probabilistic Vector หรือ Stochastic Vector) ที่จะเกิดออโตมาตานั้น เมื่อเริ่มที่สเตทที่กำหนด รายละเอียดเกี่ยวกับ PFA สามารถอ่านเพิ่มเติมได้ที่ [9], [10]

นิยาม 2.4 : Probabilistic Finite Automata (PFA) คือ แมทชีน $M = (\Sigma, Q, \delta, I, F, P)$ โดย

- Σ เป็นเซตของสัญลักษณ์ที่เป็นอินพุต (Alphabet)
- Q คือ เซตของสเตท
- $\delta : Q \times \Sigma \to P(Q)$ คือ ทรานสิชั่นฟังก์ชั่น (Transition Function)
- $I : Q \to \Re^+$ เป็นความน่าจะเป็นของสเตทเริ่มต้น (Initial-state Probability)
- $P : Q \to \Re^+$ เป็นความน่าจะเป็นของทรานสิชั่น (Transition Probability)
- $F : Q \to \Re^+$ เป็นความน่าจะเป็นของสเตทสุดท้าย (Final-state Probability)

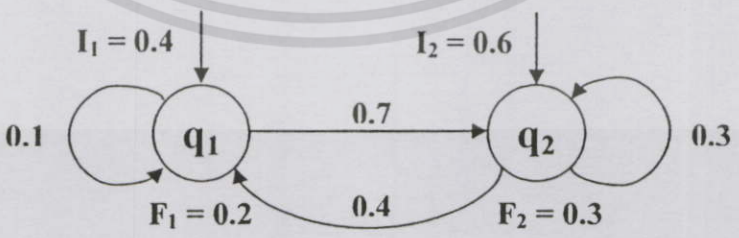
ซึ่ง I , P และ F เป็นฟังก์ชั่น โดยที่

$$\sum_{q \in Q} I(q) = 1$$

และ

$$\forall q \in Q, F(q) + \sum_{a \in \Sigma, q' \in Q} P(q, a, q') = 1$$

รูปที่ 2.5 จะแสดงตัวอย่างของ PFA จากรูปจะเห็นว่าทรานสิชั่นจากสเตทหนึ่งไปอีกสเตทหนึ่งจะมีค่าเป็นความน่าจะเป็น ซึ่งความน่าจะเป็นของทุกๆทรานสิชั่นที่ออกจากสเตทนั้นรวมกันแล้วจะได้เท่ากับ 1 และทุกๆสเตทจะมีค่าความน่าจะเป็นของสเตทเริ่มต้น ในที่นี้คือ ที่สเตท q_1 จะมีโอกาสในการเป็นสเตทเริ่มต้น $I_1 = 0.4$ ส่วน q_2 จะมีโอกาสในการเป็นสเตทเริ่มต้น $I_2 = 0.6$ นอกจากนั้น ในแต่ละสเตทยังมีค่าความน่าจะเป็นในการเป็นสเตทสุดท้าย โดยที่ q_1 และ q_2 มีโอกาสเป็นสเตทสุดท้าย $F_1 = 0.2$ และ $F_2 = 0.3$ ตามลำดับ และค่าความน่าจะเป็นที่สเตทใดๆ ของทุกๆทรานสิชั่น รวมกับค่าความน่าจะเป็นในการเป็นสเตทสุดท้ายรวมกันแล้วให้ได้เท่ากับ 1 ทั้งนี้ ลักษณะการนิยาม PFA อาจจะแตกต่างกันไปบ้าง แต่หลักสำคัญคือ ทรานสิชั่นจะมีค่าเป็นความน่าเป็นแทนที่จะเป็นอินพุต PFA ที่เป็นที่รู้จักกันดี คือ Hidden Marmov Model ซึ่งสามารถอ่านรายละเอียดของวิธีนี้ได้ในบทที่ 3 หัวข้อที่ 3.2



รูปที่ 2.5 แสดงตัวอย่างของ Probabilistic Finite Automata (PFA)

2.3 ภาษา (Language)

นิยาม 2.5 : กำหนดให้ Σ เป็น Alphabet และ $L \subseteq \Sigma^*$ แล้ว L เป็นภาษา (Language) บน Σ โดยภาษา (Language) คือ เซตของสตริงใดๆที่เป็นซับเซตของ Σ^*

ภาษาของสตริง สามารถแบ่งได้ตามลักษณะของแมทชีนที่ยอมรับภาษานั้นๆ ในที่นี้ จะขอกล่าวถึงเพียง 2 ภาษา คือ Regular Language และ Stochastic Language

2.3.1 Regular Language

นิยาม 2.6 : ภาษา (Language) จะเรียกว่า Regular Language ก็ต่อเมื่อ สามารถหา Finite Automaton มารู้จำ (Recognize) ภาษานั้นได้

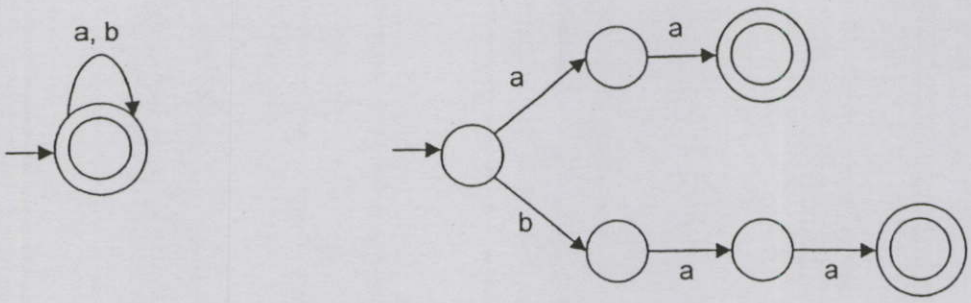
ภาษา Regular Language จะเป็นภาษาจำกัดหรือไม่จำกัดก็ได้ แต่ภาษาจำกัด (Finite Language) จะเป็น Regular Language เสมอ และจะเป็นภาษาที่ยอมรับโดย DFA หรือ NFA ใดๆ ซึ่งสามารถเขียนอธิบายรูปแบบของภาษาได้ด้วย Regular Expression ตัวอย่างเช่น $(0 \cup 1)^*$ คือ ภาษาที่ประกอบด้วยสตริงใดๆที่มี 0 และ 1 อย่างละกี่ตัวก็ได้ และเรียงลำดับอย่างไรก็ได้ เป็นต้น

2.3.2 Stochastic Language

นิยาม 2.7 : ภาษา (Language) จะเรียกว่า Stochastic Language ก็ต่อเมื่อ สามารถหา Probabilistic Automaton มารู้จำ (Recognize) ภาษานั้นได้ โดย Regular Language จะเป็นซับเซตของ Stochastic Language

2.4 ลักษณะทั่วไป (Generalization)

แมทชีนจะมีลักษณะทั่วไป หรือ *Generalized* ก็ต่อเมื่อสามารถยอมรับสตริงที่มีรูปแบบคล้ายคลึงกับสตริงตัวอย่างได้ โดยแมทชีนจะปฏิเสธสตริงอื่นๆที่อยู่นอกคลาสได้ด้วย ซึ่งถือว่าเป็นแมทชีนในอุดมคติ แมทชีนที่มีลักษณะ *Over Generalized* คือ แมทชีนที่รับสตริงมากเกินไป รวมทั้งสตริงที่ไม่ได้อยู่ในคลาสเดียวกัน ถ้าแมทชีนไม่สามารถยอมรับสตริงที่ควรจะอยู่ในคลาสนั้น ๆ แมทชีนนั้นมีลักษณะ *Under Generalized*



(a) Over Generalized **(b) Under Generalized**

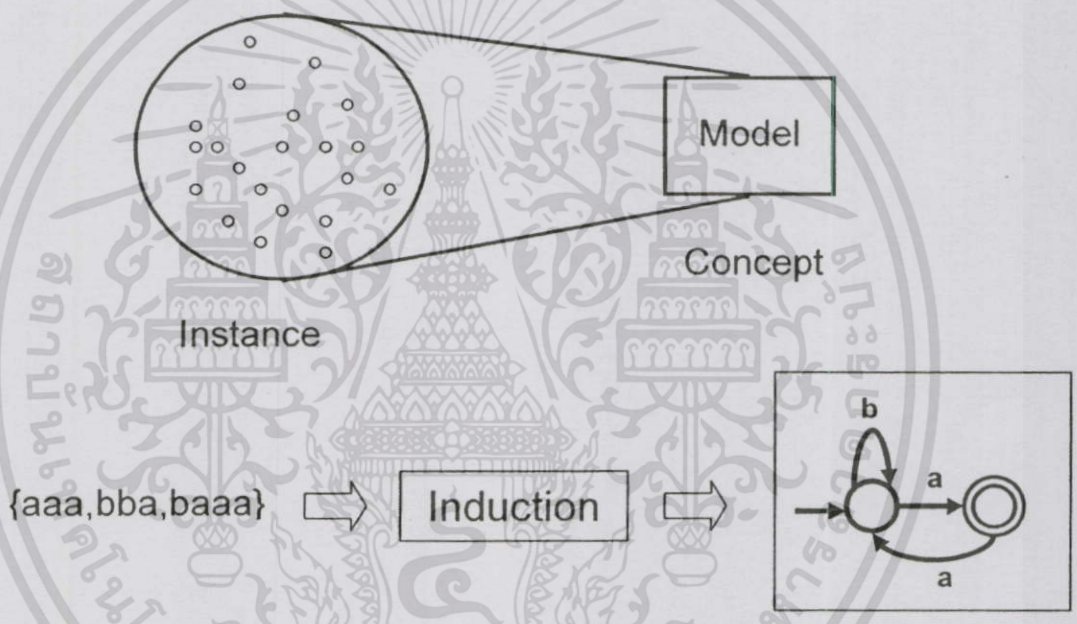
รูปที่ 2.6 แสดงลักษณะทั่วไปที่ผิดปกติของแมทชีน

รูปที่ 2.6 จะแสดงลักษณะของแมทชีนที่ Over Generalized และ Under Generalized ซึ่งสร้างจากสตริงตัวอย่าง {aa, baa} ในรูป 2.6(a) แมทชีนจะยอมรับสตริงทุกตัวที่ประกอบด้วย a และ b ซึ่งทำให้ไม่สามารถแยกแยะความแตกต่าง หรือลักษณะเด่นของสตริงตัวอย่างนี้จากสตริงอื่นๆได้เลย จึงเรียกแมทชีนลักษณะนี้ว่า Over Generalized ส่วนรูป 2.6(b) แมทชีนจะสามารถยอมรับได้เฉพาะสตริงตัวอย่างที่กำหนดมาให้เท่านั้น แต่ไม่สามารถยอมรับสตริงอื่นๆที่ใกล้เคียงกันได้ จึงเรียกแมทชีนลักษณะนี้ว่า Under Generalized

การจะบอกว่าโมเดลมีลักษณะ Generalized หรือไม่นั้น ขึ้นอยู่กับลักษณะของสตริงต้นแบบด้วย ว่าเป็นสตริงจากปัญหาชนิดใด มีสตริงตัวอย่าง (Training String) แบบไหน และสตริงที่นำมาใช้ทดสอบ (Test String) เป็นแบบใด การออกแบบโมเดลให้มี Genearalization ที่ดี จะต้องพิจารณาจากปัญหาต้นแบบของสตริงเป็นสำคัญ อย่างเช่น ในวิทยานิพนธ์ฉบับนี้ จะพิจารณาออกแบบโมเดลจากสตริงที่ได้จากการเส้นขอบของตัวอักษร เพื่อใช้ในการรู้จำ จึงกำหนดรูปแบบสเตทภายในโมเดลเป็น 2 แบบ คือ Loop และ Loop Pair ตาม [1] เพื่อแทนทิศทางที่ตรงกับทิศของเชนโค้ด และทิศทางที่อยู่ระหว่างทิศหลักของเชนโค้ด และบังคับให้โมเดลไม่มีทรานสิชั่นย้อนกลับ เนื่องจากทรานสิชั่นย้อนกลับ อาจจะทำให้เกิด Over Generalization ขึ้นภายในโมเดลได้

บทที่ 3
งานวิจัยที่เกี่ยวข้อง

Angluin และ Smith [11] กล่าวว่า การอนุมาน (Induction) เป็นกระบวนการในการหาข้อสมมติฐานในรูปแบบของกฎทั่วไป (General Rule) จากกลุ่มตัวอย่าง หรือกล่าวอีกนัยหนึ่งก็คือ เราใช้ตัวอย่างเหตุการณ์ต่างๆที่เกิดขึ้น (Instance) มาสรุปเป็นหลักการ (Concept) หรือกฎ (Rule) เพื่อให้สามารถอธิบายสิ่งนั้นๆให้เข้าใจได้โดยง่าย รูปที่ 3.1 จะแสดงรูปแบบการสร้างแบบจำลอง (Model) ของหลักการ โดยการอนุมานจากตัวอย่าง



รูปที่ 3.1 แสดงการสร้างแบบจำลองจากกลุ่มตัวอย่าง

ในบทนี้ จะกล่าวถึงขั้นตอนวิธีต่างๆที่ใช้ในการอนุมานแบบจำลองจากกลุ่มตัวอย่าง ซึ่งในงานวิจัยนี้ จะสร้างแบบจำลองในรูปแบบของออโตมาตา ซึ่งสามารถแทนลำดับของสตริงตัวอย่างได้ง่าย ทั้งนี้ การสร้างแบบจำลองสามารถทำได้หลายวิธีแตกต่างกันไป อาจจะมีการใช้ค่าความน่าจะเป็นหรือคะแนนต่างๆมาช่วยในการตัดสินใจ ซึ่งได้มีการทำวิจัยกันอย่างกว้างขวาง ต่อจากนี้จะอธิบายงานวิจัยอื่นๆบางงานที่เกี่ยวข้องกับการสร้างออโตมาตาจากกลุ่มตัวอย่าง โดยแบ่งเป็น 2 ประเภท คือ Deterministic Finite Automata Learning (DFA Learning) และ Probablilistic Finite Automata Learning (PFA Learning) รวมทั้งอธิบายงานวิจัยต้นแบบที่วิทยานิพนธ์นี้จะใช้แนวคิดเดิมนำมาพัฒนาและต่อยอดต่อไป

3.1 ประเภทของอัลกอริทึมแบบต่างๆ

ในการสร้างสเตทแมทชีนหรือออโตมาตาจากกลุ่มสตริงที่กำหนดให้ สามารถจะทำการเรียนรู้ได้หลายแบบ แยกตามคุณลักษณะต่างๆกัน เช่น การให้ความรู้ก่อนการเรียน เพื่อให้สามารถสร้างแมทชีนได้เหมาะสมยิ่งขึ้น วิธีการใช้สตริงในการสร้างแมทชีน เป็นต้น สิ่งต่างๆเหล่านี้ ทำให้รูปแบบของอัลกอริทึมแต่ละประเภทแตกต่างกันไป ในที่นี้ จะขออธิบายอัลกอริทึมบางชนิดที่นำมาใช้ในการสร้างสเตทแมทชีนเพื่อการรู้จำสตริง

3.1.1 อัลกอริทึมแบบ Induction และ Deduction

Induction หรือ Inductive Reasoning เป็นกระบวนการในการหาเหตุผล โดยการนำเหตุการณ์หรือสิ่งต่างๆที่เป็นข้อเท็จจริงที่เกิดขึ้นมาสรุปรวมกัน เพื่อหาบทสรุปของสิ่งนั้นๆ เช่น ถ้าน้ำแข็งนี้เย็น จะสรุปว่า น้ำแข็งทุกก้อนจะเย็นเหมือนกันหมด เป็นต้น ในกรณีนี้ ก็คือ นำตัวอย่าง (Sample) ต่างๆมาเรียนรู้ (Learning) เพื่อสร้างเป็นโมเดลหรือแมทชีน ซึ่งสามารถอธิบายสิ่งนั้นๆได้ในภาพรวม

Deduction หรือ Deductive Reasoning เป็นกระบวนการนำข้อเท็จจริงที่เกิดขึ้นอยู่แล้ว มาขยายเพื่อให้เกิดความรู้ (Knowledge) ใหม่ๆต่อไป ตัวอย่างเช่น ถ้าฉลามเป็นปลา และปลาทุกตัวมีครีบ เราจะสรุปว่า ฉลามทุกตัวต้องมีครีบ เป็นต้น จะเห็นว่า วิธีนี้ จะต้องมีการอ้างอิงความรู้หรือข้อเท็จจริงเดิมที่มีอยู่แล้วด้วย

3.1.2 อัลกอริทึมแบบ Passive Learning และ Active Learning

การเรียนแบบ Passive (Passive Learning) เป็นกระบวนการเรียนรู้โดยถือว่า ผู้เรียนนั้นไม่มีความรู้ใดๆมาก่อนเลย ดังนั้น ผู้สอนจะต้องเป็นผู้ป้อนความรู้ต่างๆให้ทุกอย่าง และผู้เรียนจะทำการบันทึกและจดจำความรู้นั้นไว้ โดยไม่มีการปรับปรุงแก้ไขใดๆ

การเรียนแบบ Active (Active Learning) เป็นกระบวนการเรียนรู้โดยการให้ผู้เรียนสามารถดึงความรู้ใหม่ๆออกมาใช้ได้ โดยอาศัยความรู้เดิมที่มีอยู่มาต่อยอดออกไป ผู้สอนจะให้ความรู้เพียงบางส่วน และผู้เรียนต้องมีความรู้พื้นฐานอยู่แล้ว ซึ่งจะทำให้ผู้เรียนสามารถเรียนรู้ได้โดยได้รับการแนะแนวทางจากผู้สอน

3.1.3 อัลกอริทึมแบบ Supervised Learning และ Unsupervised Learning

Supervised Learning เป็นการเรียนรู้การสร้างแมทชีนจากข้อมูล (Training Data) ที่มีอยู่ ซึ่งข้อมูลที่ใช้จะประกอบด้วย อินพุตและผลลัพธ์ที่ต้องการ ถ้าผลลัพธ์นั้นเป็นฟังก์ชั่นของค่าต่อเนื่อง (Continuous Value) จะเรียกวิธีนี้ว่า Regression ถ้าผลลัพธ์เป็นลาเบลของคลาสใดๆ จะเรียกวิธีนี้ว่า Classification ซึ่งการเรียนรู้จะทำได้โดยการนำตัวอย่างที่เคยพบเห็นมาใช้ เป็นการให้ความรู้ก่อนว่า

ตัวอย่างลักษณะนี้ จะได้ผลลัพธ์เป็นแบบนี้ และนำมาใช้ทำนายผลลัพธ์ที่จะเกิดขึ้นกับอินพุตที่ไม่ทราบค่าว่าควรจะเป็นอะไรจากโมเดลที่ได้สร้างขึ้น

Unsupervised Learning เป็นวิธีการเรียนรู้เพื่อสร้างแมทชีนโดยไม่มีการกำหนดผลลัพธ์ที่จะได้เอาไว้ก่อน ข้อมูลที่ใช้ในการเรียน (Train) จะเป็นแบบสุ่ม ซึ่งเราไม่อาจทราบได้ว่าข้อมูลแต่ละตัวเป็นประเภทใด แต่จะแยกแยะข้อมูลโดยอาศัยความเหมือนของข้อมูล โดยข้อมูลประเภทเดียวกันควรที่จะอยู่ใกล้เคียงกัน วิธีนี้อาจเรียกอีกอย่างหนึ่งว่า Clustering

3.1.4 อัลกอริทึมแบบ Offline Recognition และ Online Recognition

การรู้จำตัวอักษรหรือลายมือเขียน (Character Recognition หรือ Handwritten Recognition) จะแบ่งเป็น 2 ลักษณะ คือ แบบ Offline และแบบ Online

การรู้จำแบบ Offline จะเป็นการนำรูปภาพของตัวอักษรที่ได้เขียนไว้แล้วมาหาคุณลักษณะ จึงไม่มีข้อมูลด้านเวลา เนื่องจากได้ข้อมูลมาจากภาพตัวอักษรที่เขียนเอาไว้แล้ว จากนั้น นำมาให้โปรแกรมประมวลผลเป็นตัวอักษรหรือข้อความที่ปรากฏบนคอมพิวเตอร์ได้ วิธีนี้จะใช้ข้อมูลที่มีลักษณะคงที่ (Static) ไม่มีการเปลี่ยนแปลงไปตามเวลา

การรู้จำแบบ Online จะใช้เซนเซอร์แบบพิเศษเพื่อให้สามารถแปลงข้อความได้โดยอัตโนมัติ ส่วนใหญ่จะเป็นการใช้ปากกา หรือ Stylus เขียนลงใน PDA หรือ Pocket PC โดยตัวอักษรที่เขียนจะถูกแปลงเป็นโค้ดเช่นเดียวกันแบบ Offline แต่เนื่องจากเรารับข้อมูลเข้ามาแบบเรียลไทม์ จึงสามารถนำเอาลำดับทิศทางการเคลื่อนไหวของการเขียน หรือข้อมูลที่ใช้เวลา เช่น ความเร็ว หรือลำดับพิกัดของการเขียน มาพิจารณาเพิ่มเติมได้ด้วย จึงมีข้อมูลที่ใช้ในการรู้จำมากยิ่งขึ้น

3.1.5 อัลกอริทึมแบบ Top-down และ Bottom-up

อัลกอริทึมแบบ Top-down เป็นวิธีการทำงานโดยเริ่มจากการสร้างโมเดลต้นแบบก่อน จากนั้นจะค่อยๆเพิ่มสเตทและทรานสิชั่นไปเรื่อยๆ วิธีนี้จะต้องทำการตัดสินใจว่า เส้นทางที่มีอยู่เดิมนั้นใช้ได้อยู่หรือไม่ และจะต้องแตก (Split) สเตทใหม่เมื่อใด เพื่อให้เกิดสเตทและทรานสิชั่นโดยรวมให้น้อยที่สุด ในขณะที่สามารถยอมรับสตริงตัวอย่างทั้งหมดได้

อัลกอริทึมแบบ Bottom-up เป็นวิธีการทำงานโดยเริ่มจากสร้างโมเดลย่อยๆจากสตริงตัวอย่างแต่ละตัว จากนั้น ทำการยุบรวม (Merge) สเตทที่เหมือนกันเข้าไว้ด้วยกัน วิธีนี้จะต้องทำการตัดสินใจว่า สเตทใดที่สามารถยุบรวมเข้ากับสเตทอื่นได้บ้าง และจะยุบอย่างไรเพื่อให้เหลือสเตทน้อยที่สุด

3.2 DFA Learning

การสร้าง DFA โดยการเรียนรู้จากกลุ่มตัวอย่างนั้น ได้มีการวิจัยกันมานาน ซึ่งจะเป็นการสร้าง DFA เพื่อรู้จำ Regular Language ที่สามารถแทนปัญหาทั่วๆไปที่เราพบในปัจจุบันได้ Gold [12] ได้พิสูจน์ว่า เราไม่สามารถหาวิธีที่มีประสิทธิภาพในการสร้าง DFA ที่มีขนาดเล็กที่สุด จากกลุ่มตัวอย่างที่เป็นบวกและลบได้ ดังนั้น จึงได้มีงานวิจัยในสาขานี้มากมายที่พยายามหาวิธีที่ดีในการสร้าง DFA เพื่อใช้เป็นตัวแทนของกลุ่มตัวอย่าง ในที่นี้ จะขอยกตัวอย่างวิธีการสร้าง DFA จากตัวอย่างที่เป็นบวกและลบบางวิธี คือ วิธี Trakhtenbrot – Barzdin วิธี Self-Adaptive Greedy Estimate และวิธี Evidence-Driven State Merging เพื่อให้เห็นภาพวิธีการสร้าง DFA ในเบื้องต้น

3.2.1 วิธีแบบ Trakhtenbrot – Barzdin (TB)

Trakhtenbrot และ Barzdin [2] ได้เสนอขั้นตอนวิธีในการสร้าง DFA โดยใช้ข้อมูลตัวอย่าง ซึ่งเป็นวิธีที่เรียบง่าย ไม่ซับซ้อน และใช้เวลาน้อย ภายใน polynomial time วิธี TB นั้น จะเริ่มจากการสร้าง PTA (Prefix Tree Acceptor) ของข้อมูลตัวอย่างทั้งหมดก่อน ซึ่งแต่ละโหนดต้องมีการลาเบลว่า โหนดใดที่ยอมรับ (Accept) และโหนดใดที่ปฏิเสธ (Reject) จากนั้น จะทำการรวมโหนดที่มีซับทรี (Sub-tree) เหมือนกัน เพื่อให้กลายเป็น DFA ที่มีขนาดเล็กที่สุด ขั้นตอนวิธีที่จะอธิบายต่อไปนี้ อ้างอิงจากวิธี TB ที่อธิบายไว้ใน [13] ซึ่งมีรายละเอียดเป็นดังนี้

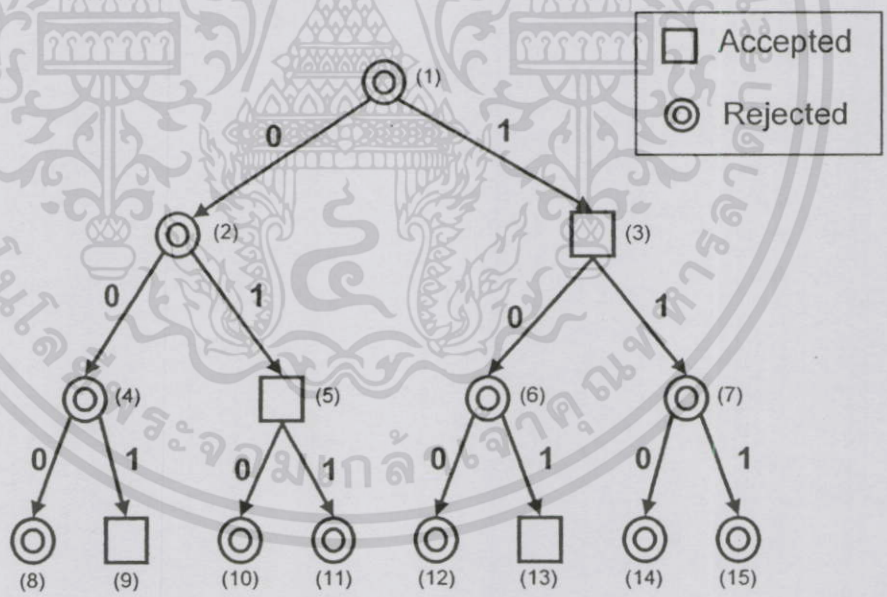
1. กำหนดให้ Root ของทรีเป็นโหนดเฉพาะ (Unique Node) และเพิ่ม Root เข้าไปในลิสต์ของโหนดเฉพาะ โดยโหนดเฉพาะ คือ โหนดที่ไม่เหมือนกับโหนดที่ผ่านมาแล้ว หรือกล่าวได้ว่า เป็นโหนดที่ไม่สามารถไปรวมกับโหนดใดได้อีก (แต่โหนดอื่นสามารถยุบมารวมกับโหนดนั้นได้) และเมื่อ Root เป็นโหนดแรก และไม่ได้ผ่านโหนดใดมาก่อน จึงกำหนดให้เป็นโหนดเฉพาะ
2. เยี่ยมโหนดอื่นๆที่เป็นโหนดลูกต่อๆมาตามแนวกว้าง (Bredth-first Order) เปรียบเทียบซับทรีของโหนดนั้น กับซับทรีของแต่ละโหนดในลิสต์
3. ถ้าโหนดนั้นไม่เหมือนกับโหนดในลิสต์ กล่าวคือ ซับทรีของทั้งสองโหนดมีลาเบลขัดแย้งกัน ให้ถือว่าโหนดนั้นเป็นโหนดเฉพาะ และเพิ่มโหนดเข้าไปในลิสต์
4. แต่ถ้าโหนดนั้นเหมือนกับโหนดใดโหนดหนึ่งในลิสต์ คือ มีลาเบลตรงกัน และสามารถยุบรวมโหนดกันได้ ให้ยุบทั้งสองโหนดเข้าด้วยกัน โดยย้ายทรานสิชั่นที่ชี้ไปยังโหนดนั้นให้เชื่อมต่อกับโหนดในลิสต์ที่เหมือนกันแทน และตัดโหนดรวมทั้งซับทรีของโหนดนั้นทิ้งไป

รูปที่ 3.2 จะแสดงอัลกอริทึมของวิธี TB ในรูปแบบ Pseudo Code [13]

```
Put root node in list of unique node L
do
    for each node (λ) that not in L
        Compare sub-tree of λ with sub-tree of each node in list L
        if λ differs from each node in L then
            Add λ to L
        else
            Disconnect λ from APTA
            Redirect all transitions to λ to point to the node in list L
        endif
until (all nodes in L)
```

รูปที่ 3.2 แสดงอัลกอริทึมของวิธี Trakhtenbrot – Barzdin [13]

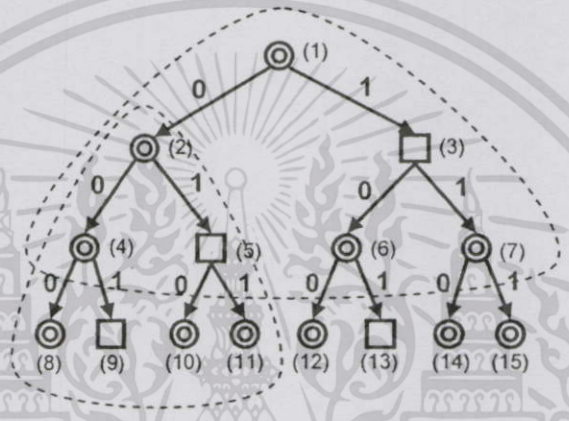
เพื่อให้สามารถเข้าใจได้ง่าย จะขอยกตัวอย่างการสร้าง DFA ด้วยวิธีนี้ โดยมี PTA ของข้อมูลตัวอย่างดังรูปที่ 3.3 อันประกอบด้วยข้อมูลที่เป็นบวก S_+ และข้อมูลที่เป็นลบ S_- ทุกๆโหนดจะต้องมีการลาเบลว่าสตริงนั้น Accept หรือ Reject ในที่นี้ $S_+ = \{001, 01, 1, 101\}$ และ $S_- = \{\lambda, 0, 00, 000, 010, 011, 10, 100, 11, 110, 111\}$



รูปที่ 3.3 แสดง Prefix Tree Acceptor ของข้อมูลตัวอย่างของวิธี Trakhtenbrot – Barzdin

การเข้าถึงแต่ละโหนด จะเป็นไปตามแนวกว้าง หรือ Breadth-first Order โดยเริ่มจาก Root เป็นอันดับแรก คือ ไล่จากโหนดที่ (1) ไปจนถึงโหนดที่ (15) ตามลำดับ และทำการเปรียบเทียบแต่ละโหนดว่ามีลักษณะเหมือนกันหรือไม่ กล่าวคือ มีซับทรีเหมือนกัน และโหนดทุกโหนดไม่มีลาเบลที่ขัดแย้งกัน ก็จะทำการยุบโหนดนั้นมารวมกัน

การทำงานจะเริ่มจากโหนดที่ 1 เป็นโหนดเฉพาะ (เมื่อโหนดนั้นๆไม่สามารถรวมกับโหนดใดๆที่ผ่านมาแล้วได้) เนื่องจากโหนดที่ 1 เป็นโหนดราก (Root Node) เป็นโหนดแรกที่เข้าเยี่ยมในทรี และไม่ได้ผ่านโหนดใดมาก่อน จึงกำหนดให้อยู่ในลิสต์โหนดเฉพาะ ต่อมา จะเริ่มเปรียบเทียบทุกๆ โหนดต่อมาตามแบบ Breadth-first Order ว่าซับทรีของโหนดต่อๆมานั้นเหมือนกับซับทรีของโหนดที่อยู่ในลิสต์หรือไม่ ตัวอย่างเช่น จะเปรียบเทียบซับทรีของโหนดที่ 2 กับซับทรีของโหนดในลิสต์ (ในที่นี้คือ โหนด 1) ว่ามีลาเบลตรงกันหรือไม่ ถ้าลาเบลตรงกัน ก็จะย้ายทรานสิชั่นให้ชี้เข้าที่โหนด 1 และตัดโหนด 2 รวมทั้งซับทรีของมันทิ้งไป รูปที่ 3.4 จะแสดงซับทรีที่เหมือนกันของโหนดที่ 1 และ 2

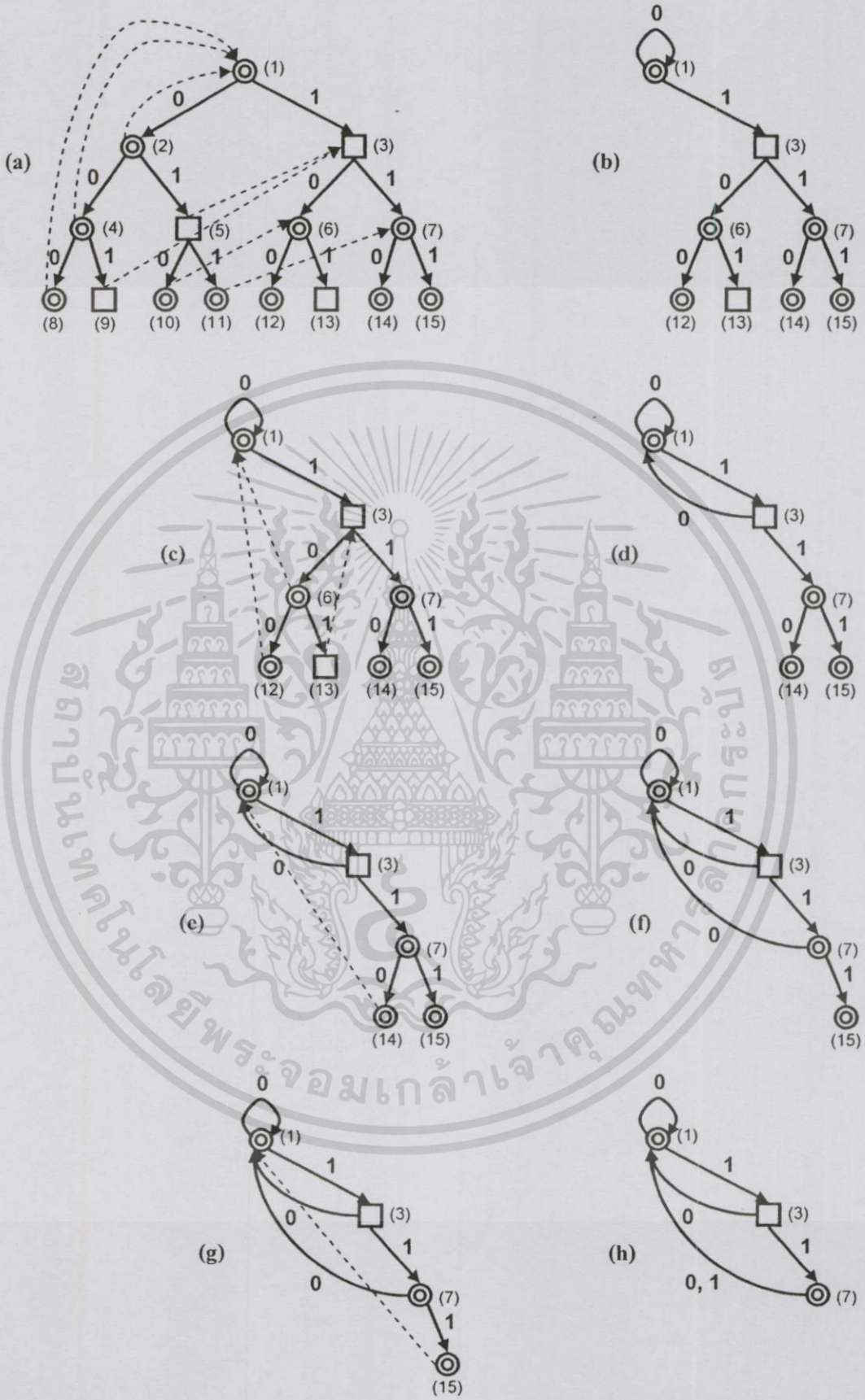


รูปที่ 3.4 แสดงซับทรีของโหนด 1 และ 2 ที่เหมือนกันในวิธี Trakhtenbrot – Barzdin

จากรูปที่ 3.4 เมื่อซับทรีทั้งสองเหมือนกัน จึงพิจารณายุบโหนด โดยจะต้องตรวจสอบว่า เมื่อยุบโหนดไปแล้ว ลาเบลจะต้องไม่ขัดแย้งกัน เมื่อโหนด 2 จะเข้าไปรวมกับโหนด 1 แล้ว ทรานสิชั่น 0 จากโหนด 1 มาโหนด 2 จะย้ายไปชี้ที่โหนด 1 แทน ดังนั้น จะทำให้เกิดลูปเข้าที่โหนด 1 สำหรับอินพุตเท่ากับ 0 ซึ่งจะทำให้ซ้ำกับโหนดที่ 4 และ 8 ด้วย สำหรับการ Reject สตริง {0, 00, 000} และทำให้โหนดที่ 5 และ 9 ซ้ำกับโหนดที่ 3 ซึ่งต้อง Accept สตริง {1, 01, 001} รวมทั้งโหนดที่ 10 ซ้ำกับโหนดที่ 6 และโหนดที่ 11 ซ้ำกับโหนดที่ 7 ในรูปแบบเดียวกัน จึงสามารถยุบรวมโหนดกันได้

ในรูปที่ 3.5(a) ลูกศรเส้นประจะแสดงลาเบลที่ตรงกันของซับทรีที่โหนด 1 และโหนด 2 อย่างเช่น โหนดที่ 8, 4 และ 2 สามารถรวมเข้ากับโหนดที่ 1 ได้ โหนดที่ 5 และ 9 สามารถรวมกับโหนดที่ 3 ได้ โหนดที่ 10 และ 11 สามารถรวมกับโหนดที่ 6 และ 7 ได้ ตามลำดับ เมื่อรวมโหนด 1 และ 2 แล้ว จะได้ดังรูปที่ 3.5(b)

ต่อมา พิจารณาที่โหนด 3 ว่าสามารถรวมกับโหนดใดๆในลิสต์ได้หรือไม่ เริ่มจากโหนดที่ 1 ปรากฏว่า มีลาเบลขัดแย้งกัน ไม่สามารถรวมโหนด 1 และ 3 ได้ และไม่มีโหนดอื่นๆในลิสต์ที่โหนด 3 จะรวมได้อีก (เนื่องจากมีโหนดในลิสต์เพียงโหนดเดียว คือ โหนด 1) จึงกำหนดให้โหนด 3 เป็นโหนดเฉพาะ



รูปที่ 3.5 แสดงตัวอย่างการสร้าง DFA โดยวิธี Trakhtenbrot – Barzdin

เมื่อพิจารณาที่โหนดที่ 6 ดังรูปที่ 3.5(c) ว่าสามารถรวมกับโหนดใดในลิสต์ได้หรือไม่ พบว่าซับทรีของโหนด 6 และ 1 ไม่มีลาเบลขัดแย้งกัน (โหนด 12 รวมกับโหนด 1 ได้ และโหนด 13 รวมกับโหนด 3 ได้) จึงย้ายทรานสิชั่นที่ชี้เข้าโหนด 6 ให้ไปชี้เข้าที่โหนด 1 ดังรูปที่ 3.5(d) ทำเช่นนี้ซ้ำไปเรื่อยๆ จนกว่าทุกโหนดจะกลายเป็นโหนดเฉพาะ ซึ่งหมายความว่า ไม่มีโหนดใดสามารถรวมกันได้อีก จนได้เป็น DFA สุดท้าย ดังในรูปที่ 3.5(h)

3.2.2 วิธีแบบ Self-Adaptive Greedy Estimate (SAGE)

ในปี 1997 นั้น Lang และ Pearlmutter [4] ได้จัดการแข่งขัน Abbadingo One DFA Learning Competition ขึ้น เพื่อสนับสนุนให้มีการพัฒนากลวิธีการสร้าง DFA จากกลุ่มตัวอย่างทั้งสตริงที่เป็นบวกและลบ ซึ่งลักษณะข้อมูลจะมีความหนาแน่นน้อย และยากต่อการสร้างโมเดล นำมาสร้าง DFA ตามขนาดที่กำหนด เพื่อทดสอบกับสตริงที่ไม่ทราบค่า โดยให้มีค่าความถูกต้องมากเกินกว่า 99% และผู้ชนะการแข่งขันครั้งนี้ คือ Juillé และ Price จากกลุ่มตัวอย่างต่างๆที่มีความแตกต่างกัน

ขั้นตอนวิธีของ Juillé หรือวิธี Self-Adaptive Greedy Estimate (SAGE) [3] จะเป็นการสร้าง DFA โดยเริ่มจากการสร้าง PTA ของอินพุตสตริงก่อน จากนั้น ทำการรวมสเตทเข้าไว้ด้วยกันให้กลายเป็น DFA เช่นเดียวกับวิธี TB แต่ได้ใช้เทคนิคการสุ่ม (Sampling) มาช่วยให้การรวมสเตทมีความถูกต้องมากขึ้น และลักษณะข้อมูลของวิธี SAGE จะแตกต่างกับวิธี TB กล่าวคือ ในวิธี SAGE นั้น ข้อมูลไม่จำเป็นต้องมีการลาเบลทุกโหนด ว่าโหนดนั้น Accept หรือ Reject อาจจะมีบางโหนดที่ไม่ได้ระบุค่า (Unknown) ดังนั้น โหนดที่ไม่ได้ระบุค่า อาจจะถูกยุบรวมกับโหนดที่ถูกลาเบลว่าเป็น Accept หรือ Reject ก็ได้ (แต่โหนดที่เป็น Accept ไม่สามารถรวมกับโหนดที่เป็น Reject ได้) ขั้นตอนวิธีจะเป็นดังนี้

1. เริ่มจากกำหนดให้ Root ของทรีเป็นสเตทเริ่มต้นของ DFA
2. สุ่มเลือกสเตทใดๆที่ยังไม่มีทรานสิชั่นออกจากสเตทนั้น แล้วสร้างทรานสิชั่น '0' และ '1' ออกจากสเตทที่เลือก โดยทรานสิชั่นที่สร้างขึ้นอาจจะเชื่อมต่อไปยังสเตทที่มีอยู่ หรือสเตทใหม่ก็ได้
3. ถ้าทรานสิชั่นที่สร้างขึ้นได้เชื่อมไปยังสเตทที่มีอยู่แล้ว แสดงว่าโหนดได้ถูกรวมกันแล้ว (Merge) และจะมีการเปลี่ยนลาเบลของแต่ละโหนดตามการรวมกันนั้น
4. ย้อนกลับไปทำข้อ 4 ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะสร้าง DFA ได้เสร็จสมบูรณ์

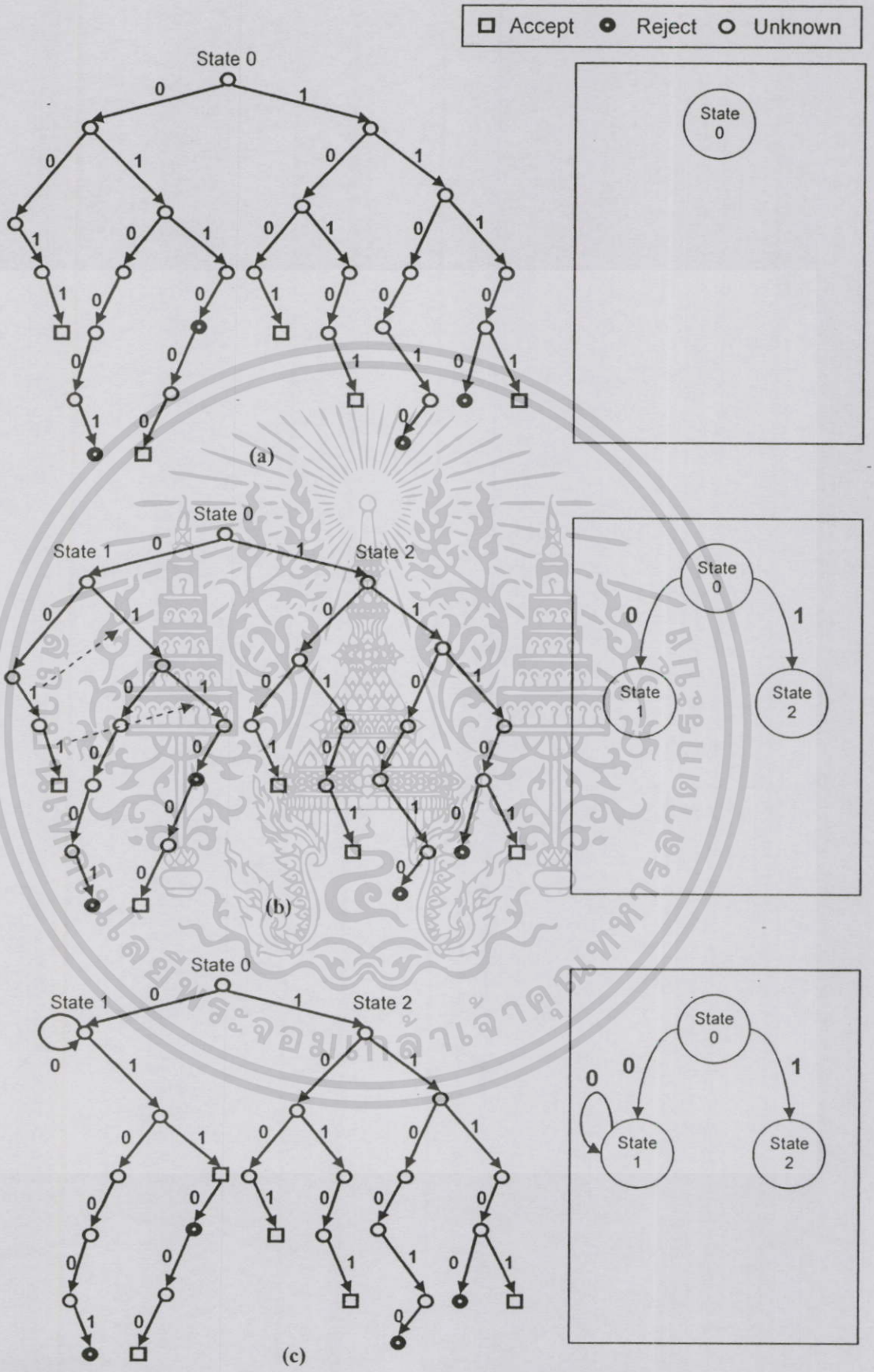
รูปที่ 3.6 จะแสดงอัลกอริทึมของวิธี SAGE ในรูปแบบ Pseudo Code [3]

```
Begin with first state map to root of tree
Put that state in unprocessed state list L

do
    Pick randomly a state S from L

    Compute the set T0 of valid transition on '0' from state S
    Pick randomly a transition t0 from T0
    if t0 goes to an existing state then
        Merge corresponding nodes in tree
    else
        Create new state, map to corresponding node and add to L
    endif

    Compute the set T1 of valid transition on '1' from state S
    Pick randomly a transition t1 from T1
    if t1 goes to an existing state then
        Merge corresponding nodes in tree
    else
        Create new state, map to corresponding node and add to L
    endif

until (L is empty)
```

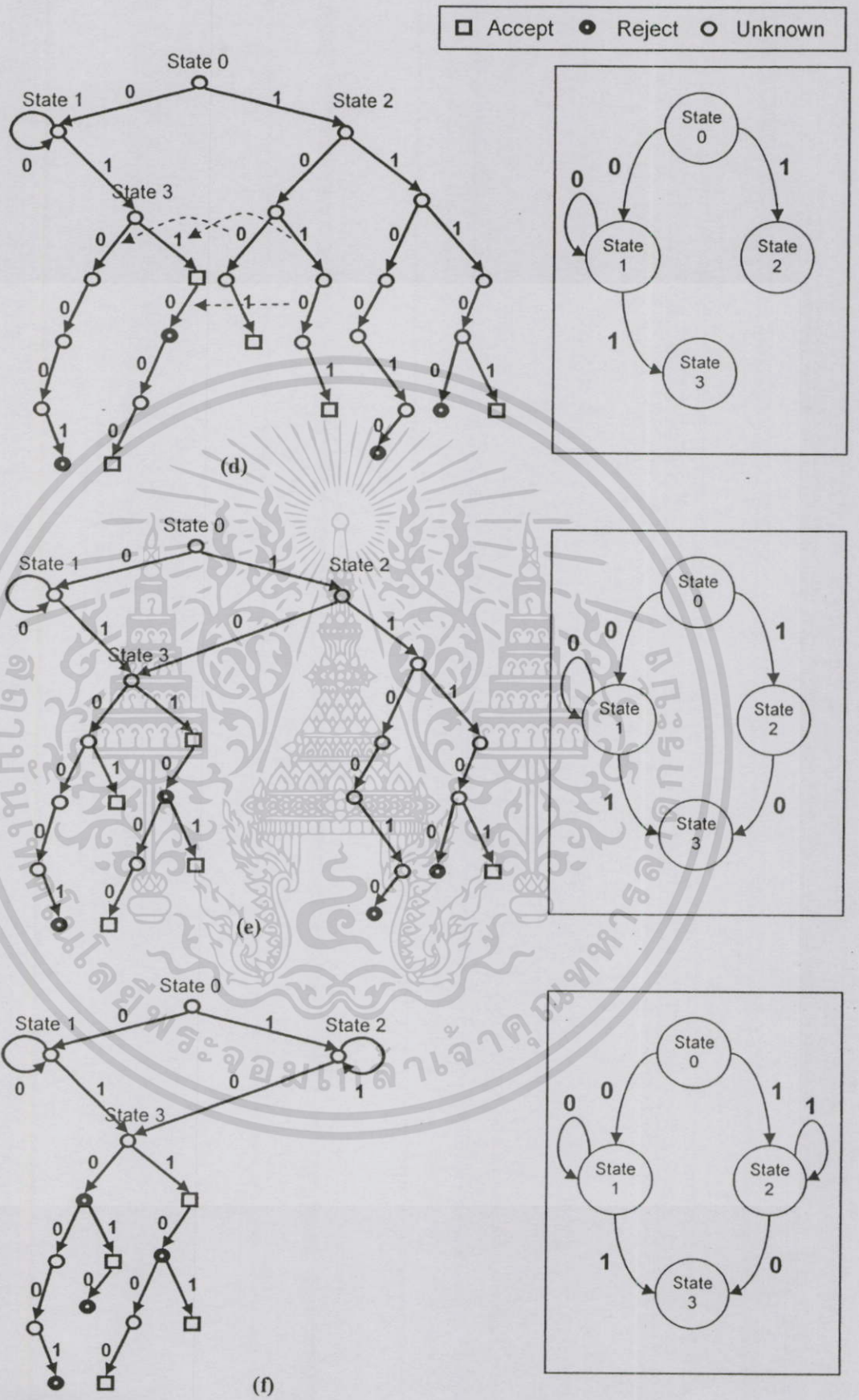
รูปที่ 3.6 แสดงอัลกอริทึมของวิธี Self-Adaptive Greedy Estimate [3]

รูปที่ 3.7 เป็นตัวอย่างการสร้าง DFA โดยใช้วิธี SAGE ซึ่งข้อมูลสตริงตัวอย่างจะประกอบไปด้วย S_+ = {0011, 011000, 1001, 10101, 11101} และ S_- = {010001, 0110, 110010, 11100} โหนดอื่นๆที่ไม่ได้ลาเบลจะสามารถรวมกับโหนดใดๆก็ได้ โดยในที่นี้ โหนดที่ไม่ได้ระบุค่า จะแทนด้วยสัญลักษณ์วงกลม หลักการรวมสเตทก็เช่นเดียวกับวิธี TB คือ จะรวมสเตทที่ลาเบลไม่ขัดแย้งกันเข้าไว้ด้วยกัน เพียงแต่ลำดับสเตทที่จะพิจารณาจะไม่ได้เรียงตามแนวกว้างทุกครั้งดังเช่นวิธี TB แต่จะเป็นการสุ่มเลือกสเตทที่เป็นไปได้แทน

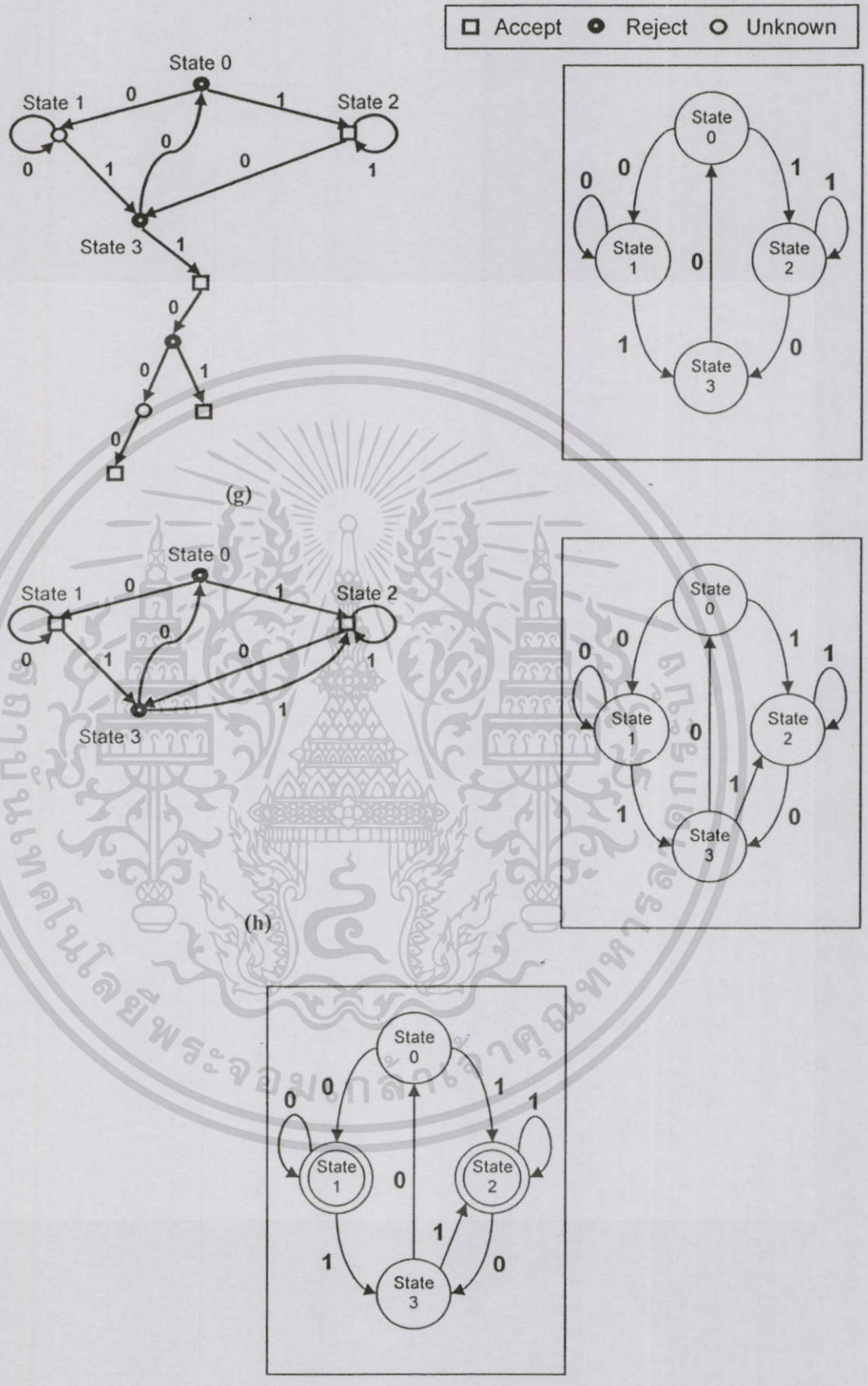
จะเห็นว่า ในวิธี TB นั้น DFA ที่ได้จะสามารถรับสตริงทุกตัวใน S_+ ได้ และปฏิเสธสตริงทุกตัวใน S_- ได้ โดยจะยอมรับหรือปฏิเสธสตริงอื่นๆที่ยังไม่ทราบค่า เฉพาะสตริงที่ยาวกว่าสตริงตัวอย่างเท่านั้น (เนื่องจากทุกๆโหนดของทรีจะต้องถูกลาเบล ซึ่งรวมถึงสตริงที่สั้นไปแล้ว) ในขณะที่ข้อมูลของ SAGE มีบางโหนดที่ไม่ทราบค่า ซึ่งสามารถถูกแทนที่ด้วยโหนดที่ Accept หรือ Reject ก็ได้ DFA ที่ได้ จึงอาจจะยอมรับหรือปฏิเสธสตริงอื่นที่สั้นกว่า หรือยาวกว่าสตริงตัวอย่างก็ได้



รูปที่ 3.7 แสดงตัวอย่างการสร้าง DFA โดยวิธี Self-Adaptive Greedy Estimate



รูปที่ 3.7 (ต่อ)



รูปที่ 3.7 (ต่อ)

จากรูปที่ 3.7 ภาพด้านซ้ายจะเป็น APTA หรือทรีที่สร้างขึ้นจากสตริงตัวอย่าง ส่วนด้านขวา คือ DFA ที่ได้จากการสร้างโมเดลด้วยวิธี SAGE ลูกศรเส้นประจะแสดงถึงทรานสิชั่นที่ตรงกันของโหนดที่สามารถยุบเข้าด้วยกันได้ โดยเริ่มจาก Root จะกำหนดให้เป็นสเตทที่ 0 แล้วไล่ดูที่โหนดต่อไปสำหรับทรานสิชั่นที่มีอินพุตเท่ากับ 0 และ 1 กำหนดให้เป็นสเตทที่ 1 และ 2 ตามลำดับ จากนั้นดูว่ามีโหนดที่สามารถยุบรวมกันได้โดยลาเบลไม่ขัดแย้งหรือไม่ และพยายามยุบโหนดรวมกันให้เหลือเป็น DFA ที่มีขนาดเล็กที่สุด โดยสุดท้ายจะได้เป็น DFA ดังรูปที่ 3.7(i) ซึ่งกำหนดให้สเตทที่ Accept กลายเป็น Final State ของ DFA นั่นเอง

3.2.3 วิธีแบบ Evidence-Driven State Merging (EDSM)

วิธี Evidence-Driven State Merging (EDSM) [4] จะใช้หลักการรวมสเตทเช่นเดียวกับสองวิธีข้างต้น แต่ตามปกติแล้ว ลำดับการเลือกรวมสเตทใดๆจะทำตามแนวกว้าง (Breadth-first Order) เนื่องจากทำให้สามารถตัดซับทรีขนาดใหญ่ทิ้งได้รวดเร็วกว่า แต่การกำหนดลำดับเช่นนี้ ทำให้เกิดข้อจำกัดในการรวมสเตท และส่งผลกระทบต่อการรวมสเตทในภายหลัง ซึ่งอาจทำให้เกิดความผิดพลาดได้ ดังนั้น จึงได้คิดวิธีการคำนวณหาค่า Evidence และจะทำการรวมสเตทที่มีค่า Evidence มากที่สุด ซึ่ง Price ได้นำวิธีนี้มาพัฒนา เพื่อประยุกต์ให้เข้ากับโจทย์ในการแข่งขันมากขึ้น

Price [4] ผู้ชนะในการแข่งขัน Abbadingo อีกรายหนึ่ง ได้นำวิธี EDSM เดิมมาปรับปรุง ซึ่งเรียกวิธีใหม่นี้ว่า Blue-fringe Algorithm หรือ Red-Blue Algorithm ซึ่งยังคงใช้หลักการ EDSM เดิม แต่ข้อกำหนดเพิ่มขึ้น กล่าวคือ กำหนดให้โหนดสีแดงเป็นโหนดที่ไม่สามารถรวมกับโหนดอื่นๆได้ (Unmergeable) และให้โหนดลูกของโหนดสีแดงทุกๆโหนดเป็นโหนดสีน้ำเงิน การรวมโหนด จะทำได้เฉพาะการรวมโหนดสีแดงและสีน้ำเงินเท่านั้น ขั้นตอนวิธี Blue-fringe จะเป็นดังนี้

1. เริ่มจากกำหนดให้ Root ของทรีเป็นโหนดสีแดง แทนด้วยสัญลักษณะ R และให้โหนดลูกของโหนดสีแดง เป็นโหนดสีน้ำเงิน แทนด้วยสัญลักษณ์ B
2. คำนวณค่าคะแนนการรวมโหนด (Merging Score) ของโหนดสีแดงและสีน้ำเงิน โดยคะแนนจะได้จากการนับจำนวนโหนดที่ถูกลาเบลแล้วภายในซับทรี ลบด้วย 1 ถ้าในซับทรีของโหนดนั้น มีโหนดมีขัดแย้งกัน (เช่น โหนดหนึ่ง Accept และอีกโหนด Reject เมื่อโหนดทั้งสองอยู่ในตำแหน่งเดียวกัน) ให้ค่าคะแนนเป็น $-\infty$
3. ถ้าโหนดนั้นไม่สามารถรวมได้ ให้เปลี่ยนโหนดสีน้ำเงินให้เป็นโหนดสีแดง
4. ถ้าโหนดสามารถรวมกันได้ ทำการรวมโหนดสีแดงและสีน้ำเงินที่มีค่าคะแนนมากที่สุด

ตัวอย่างการรวมสเตทด้วยวิธี Blue-fringe จะแสดงในรูปที่ 3.8 ซึ่งจะใช้ตัวอย่างข้อมูลอันเดียวกับตัวอย่างในรูปที่ 3.7 โดยกำหนดให้โหนดที่มีสัญลักษณ์ (R) คือโหนดสีแดง และโหนดที่มีสัญลักษณ์ (B) คือโหนดสีน้ำเงิน การรวมสเตทจะรวมโหนดสีแดงและโหนดสีน้ำเงินเข้าไว้ด้วยกัน จนกว่าทุกๆโหนดในทรีจะเป็นสีแดงทั้งหมด

รูปที่ 3.8 แสดงตัวอย่างการสร้าง DFA โดยวิธี Evidence-Driven State Merging

จากรูปที่ 3.8(a) จะเริ่มด้วยการกำหนด Root ให้เป็นโหนดสีแดง จากนั้น จะกำหนดโหนดลูกของโหนดสีแดงให้เป็นโหนดสีน้ำเงิน และพยายามรวมโหนดสีแดงกับโหนดสีน้ำเงินเข้าไว้ด้วยกัน รูปที่ 3.8(b) จะเห็นว่าซับทรีของโหนดสีน้ำเงินฝั่งขวามือสามารถยุบรวมกับโหนดสีแดงได้ (ในที่นี้คือ Root ของทรี) ส่วนโหนดน้ำเงินทางฝั่งซ้ายมือ เมื่อไม่สามารถยุบรวมกับโหนดสีแดงใดๆได้ ก็จะเปลี่ยนโหนดน้ำเงิน ให้กลายเป็นโหนดแดง แล้วกำหนดโหนดลูกของมันให้กลายเป็นสีน้ำเงิน และพยายามรวมโหนดน้ำเงินกับแดงอีกครั้ง โดยจะพยายามรวมโหนดสีน้ำเงินที่มาก่อนเข้ากับโหนดสีแดงตามลำดับก่อนหลัง (โหนดสีน้ำเงินที่เพิ่งถูกกำหนดใหม่ ซึ่งเป็นโหนดลูกของโหนดสีน้ำเงินเดิมที่ถูกเปลี่ยนเป็นสีแดง จะทำทีหลัง) เมื่อพบว่าซับทรีของโหนดน้ำเงินด้านขวามือสามารถยุบรวมกับโหนดสีแดงได้ จึงได้ตามรูปที่ 3.8(c) ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะสามารถเปลี่ยนโหนดทุกโหนดในทรีให้กลายเป็นโหนดสีแดงได้ทั้งหมด หรือหมายความว่า ทุกโหนดนั้น ไม่สามารถรวมกับโหนดอื่นๆได้อีกแล้ว และได้เป็น DFA ตามรูปที่ 3.8(f) นั่นเอง

3.3 PFA Learning

การสร้าง PFA หรืออโตมาที่ใช้ความน่าจะเป็นมาประกอบนั้น วิธีที่ได้ความนิยมและเป็นที่รู้จักกันดี คือ Hidden Markov Model หรือ HMM ซึ่งได้พัฒนามาจากแนวคิดพื้นฐานของ Markov Chain ในหัวข้อนี้ จะกล่าวถึงนิยามของ Markov Chain [14] และวิธี HMM [6] ซึ่งจะใช้เป็นตัวเปรียบเทียบประสิทธิภาพการทำงานกับวิธีที่จะใช้ในวิทยานิพนธ์นี้

3.3.1 นิยามของ Markov Chain

Markov Chain จะประกอบไปด้วย เซตของสเตท $S = \{s_1, s_2, ..., s_N\}$ ซึ่งกระบวนการจะเริ่มโดยการกระโดดจากสเตทหนึ่งไปอีกสเตทหนึ่ง การย้ายสเตทนี้ จะเรียกว่า เสตป (Step) เมื่ออยู่ที่สเตท s_i จะย้ายไปที่สเตท s_j ด้วยค่าความน่าจะเป็น p_{ij} ซึ่งค่าความน่าจะเป็นนี้จะไม่ขึ้นอยู่กับเส้นทางที่ผ่านมาก่อนหน้าจะมาอยู่ ณ สเตทปัจจุบัน

ค่าความน่าจะเป็น p_{ij} จะเรียกว่า Transition Probabilities ซึ่งสามารถวนอยู่ที่สเตทเดิมได้ ด้วยค่าความน่าจะเป็น p_{ii} และจะมีการกำหนดค่าความน่าจะเป็นเริ่มต้นที่สเตทเริ่มต้นด้วย

3.3.2 วิธีแบบ Hidden Markov Model (HMM)

จาก Markov Chain เดิม ซึ่งเป็นเซตของสเตทที่มีการข้ามไปยังแต่ละสเตทด้วยค่าความน่าจะเป็น ก็ได้สร้างโมเดลใหม่โดยอิงจากแนวคิดนี้ และมีพารามิเตอร์อื่นๆเพิ่มเข้ามา เรียกว่า Hidden Markov Model หรือ HMM ซึ่งจะมีพารามิเตอร์ต่างๆประกอบด้วย

นิยาม 3.1 : Hidden Markov Model จะมีส่วนประกอบดังนี้

- N คือ จำนวนสเตทของโมเดล โดย สเตทคือ $S = \{s_1, s_2, ..., s_N\}$ และสเตทที่เวลา t เรียกว่า q_t
- M คือ จำนวนสัญลักษณ์ (Symbol) ที่ปรากฏต่อหนึ่งสเตท โดย สัญลักษณ์คือ $V = \{v_1, v_2, ..., v_N\}$
- $A = \{a_{ij}\}$ คือ ค่าความน่าจะเป็นในการเปลี่ยนสเตท หรือ Transition Probability โดย

$$a_{ij} = P[q_{t+1} = s_j \mid q_t = s_i], 1 \le i, j \le N$$

- $B = \{b_j(k)\}$ คือ ค่าความน่าจะเป็นในการเกิดสัญลักษณ์ในสเตทที่ j หรือ Symbol Probability โดย

$$b_j(k) = P[v_k(t) \mid q_t = s_j], 1 \le j \le N, 1 \le k \le M$$

- $\pi = \{\pi_i\}$ คือ ค่าความน่าจะเป็นที่จะเป็นสเตทเริ่มต้น โดย

$$\pi_i = P[q_1 = s_i], 1 \le i \le N$$

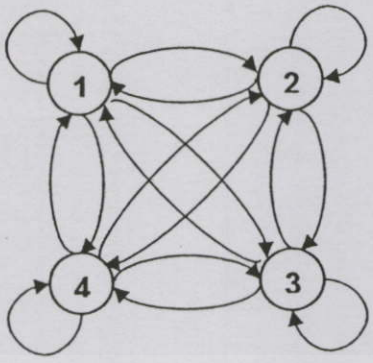
HMM จะต้องกำหนดค่าพารามิเตอร์ N, M, A, B และ π เสียก่อน จากนั้นจะสร้างโมเดลโดยใช้ลำดับของสัญลักษณ์ (Symbol) แทนด้วย

$$O = O_1 O_2 ... O_T$$

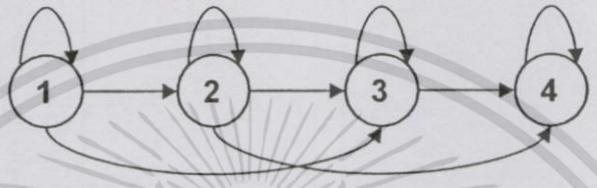
โดย แต่ละค่าที่ปรากฏเป็นสัญลักษณ์ O_t จะต้องเป็นหนึ่งในสัญลักษณ์จากเซต V และ T เป็นจำนวนค่าสัญลักษณ์ที่ปรากฏในลำดับ ซึ่งจะมีขั้นตอนการสร้างโมเดล ดังนี้

1. เลือกสเตทเริ่มต้น $q_1 = s_i$ โดยใช้ค่าความน่าจะเป็นในการเริ่มต้นสเตท π
2. กำหนดให้ $t = 1$
3. เลือก $O_t = v_k$ โดยใช้ค่าความน่าจะเป็นในการเกิดสัญลักษณ์ในสเตท s_i หรือ $b_i(k)$
4. ย้ายไปยังสเตทใหม่ $q_{t+1} = s_j$ โดยใช้ค่าความน่าจะเป็นในการเปลี่ยนสเตทของสเตท s_i หรือ a_{ij}
5. กำหนดให้ $t = t + 1$ แล้วย้อนกลับไปทำขั้นตอนที่ 3 จนกว่า $t \ge T$

โครงสร้างของ HMM (HMM Topology) สามารถเปลี่ยนแปลงไปได้ตามการกำหนดทรานสิชั่นที่ใช้ในการเชื่อมต่อจากสเตทหนึ่งไปอีกสเตทหนึ่ง เดิมแต่ละสเตทจะสามารถเชื่อมต่อถึงกันได้หมด เรียกว่า Fully Connected Topology หรือ Ergodic Topology แต่เราสามารถพิจารณาซับเซตของโมเดลนี้ในรูปแบบอื่นๆได้ รูปที่ 3.9 จะแสดงตัวอย่างโครงสร้างของ HMM ในรูปแบบ Fully Connected Topology และแบบ Left-right Topology ที่มีจำนวนสเตทเท่ากับ 4 สเตท



(a) Fully Connected Topology



(b) Left-right Topology

รูปที่ 3.9 แสดงตัวอย่างโครงสร้างแบบต่างๆของ Hidden Markov Model

โครงสร้างแบบ Fully Connected Topology จะมีรูปแบบที่สเตททุกสเตทสามารถเชื่อมต่อถึงกันได้หมด จากตัวอย่างในรูปที่ สำหรับโมเดลที่มี 4 สเตท จะสามารถเขียนเมทริกซ์ A ได้ดังนี้

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

โครงสร้างแบบ Left-right Topology จะมีรูปแบบที่สเตทจะย้ายสเตทจากซ้ายไปขวาเท่านั้น ไม่มีการย้อนกลับ ดังนั้น จะได้ว่า

$$a_{ij} = 0, j < i$$

และค่าความน่าเป็นในการเริ่มต้นสเตทจะเป็น

$$\pi_i = \begin{cases} 0, i \neq 1 \\ 1, i = 1 \end{cases}$$

เนื่องจาก สเตทจะเริ่มที่สเตท 1 และจบที่สเตท N และโดยปกติแล้วในโมเดลแบบ Left-right นี้มักจะมีข้อจำกัดให้สเตทไม่สามารถกระโดดข้ามไปได้ไกลเกินกว่าที่กำหนด ทั้งนี้เพื่อไม่ให้มีการเปลี่ยนแปลงดัชนีของสเตทมากนัก ข้อจำกัดนั้นคือ

$$a_{ij} = 0, j > i + \Delta$$

จากรูปที่ 3.9(b) ค่าของ $\Delta = 2$ หรือกระโดดข้ามสเตทไปได้ไม่เกิน 2 สเตท และจะมีทรานสิชั่นเมทริกซ์เป็นดังนี้

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

และสเตทสุดท้ายจะมีสัมประสิทธิ์การย้ายสเตทเป็น

$$a_{NN} = 1 \text{ และ } a_{Ni} = 0, i < N$$

3.4 Context – free Grammar Induction

การสร้างโมเดลในรูปแบบ Context-free Grammar จากกลุ่มตัวอย่าง มีได้หลายวิธีด้วยกัน เช่น วิธีของ Stolcke [15] หรือวิธีของ Chen [16] เป็นต้น ในที่นี้ จะขออธิบายถึงการสร้าง Context-free Grammar จากกลุ่มตัวอย่างที่มีรูปแบบเฉพาะ ของปองเกษม พลสันติกุล [1] ซึ่งจะนำมาใช้อ้างอิงในงานวิจัยต่อไป

งานวิจัยของปองเกษม [1] จะเป็นงานวิจัยที่ทำการรู้จำลายมือเขียนภาษาไทย โดยการสร้าง Context-free Grammar จากตัวอย่างเชนโค้ดของตัวอักษรนั้นๆ การสร้างแกรมม่าจะพิจารณาตามลักษณะของเชนโค้ดหรือสตริงตัวอย่าง ซึ่งจะมี 3 รูปแบบ ดังต่อไปนี้

1) Loop Policy

ตัวอย่างลำดับ : 4, 44, 444, 444444…

Regular Expression : 4*

Context-free Grammar : S1 → 4 S1

2) Loop Pair Policy

ตัวอย่างลำดับ : 434, 44333444, 4334434, 4443343434334…

Regular Expression : (4*3*)*

Context-free Grammar : S1 → 4 S1 | 3 S2

S2 → 3 S1 | 4 S2

3) Trap Loop Pair Policy

ตัวอย่างลำดับ : 5656323, 5653323, 55656332323…

Regular Expression : (5*6*)* (3*2*)*

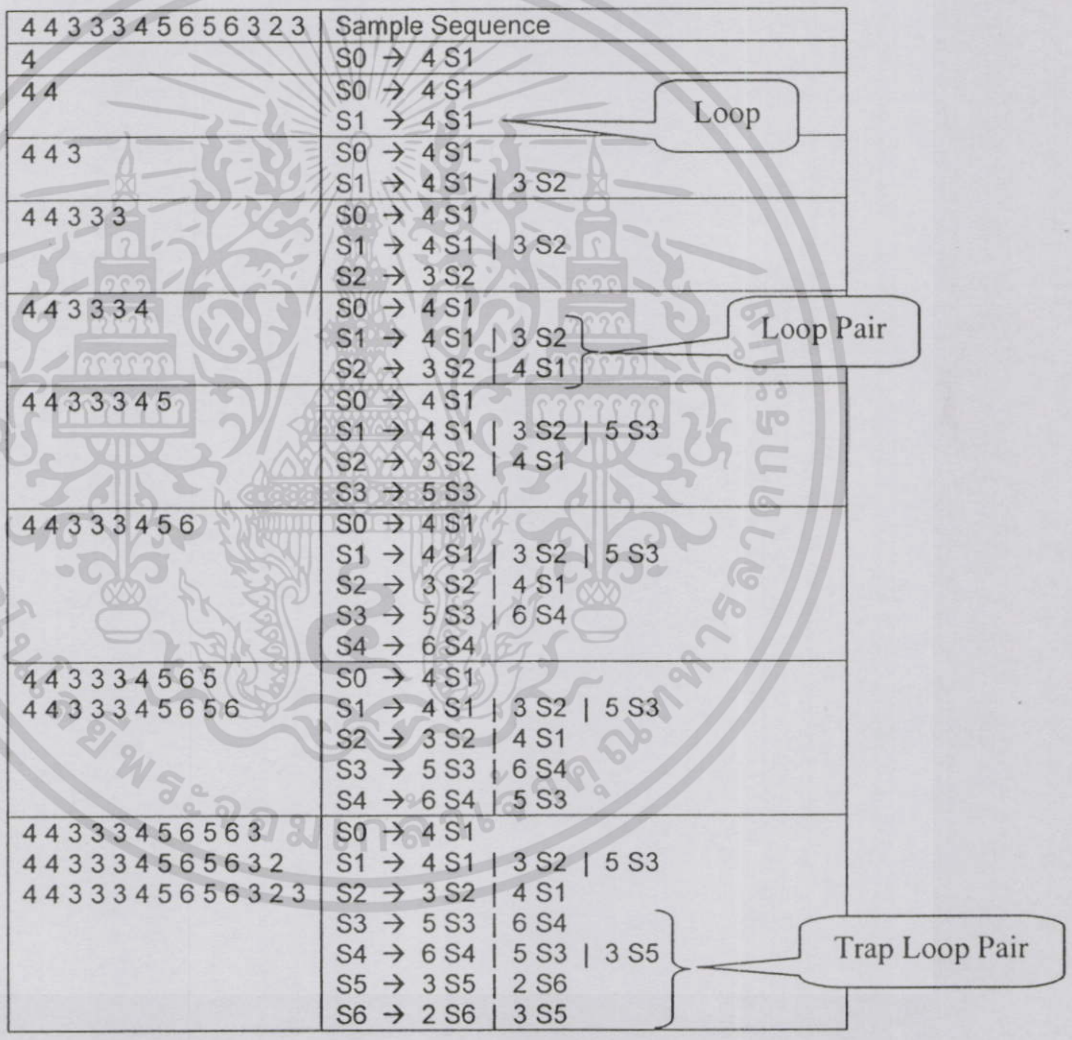
Context-free Grammar : S3 → 5 S3 | 6 S4 | 3 S5

S4 → 6 S4 | 5 S3 | 2 S5

S5 → 3 S5 | 2 S6

S6 → 2 S6 | 3 S5

วิธีการสร้างแกรมม่า จะทำนำเชนโค้ดที่เป็นสตริงตัวอย่างมาสร้างแกรมม่า โดยเลือกจากรูปแบบทั้งสาม เพื่อให้ได้แกรมม่าต้นแบบ รูปที่ 3.10 จะแสดงตัวอย่างการแกรมม่าเริ่มต้นในรูปแบบทั้ง 3 รูปแบบ จากสตริง "4433345656323" จากนั้น จะนำสตริงอื่นๆมา parse เข้าไปในแกรมม่า และปรับปรุงให้แกรมม่าสามารถยอมรับสตริงใหม่ได้ด้วย โดยการเพิ่มกฎเข้าไปใหม่ และพยายามใช้กฎเดิมที่มีอยู่ให้มากที่สุด ซึ่งจะทำการทดลองเพิ่มกฎในทุกๆรูปแบบที่เป็นไปได้ แล้วนำมาเลือกแกรมม่าที่มีค่า cost ต่ำที่สุด และ cost ในการเลือกแกรมม่า ก็คือ การเพิ่มตัวแปร (Non-terminal) ภายในแกรมม่า ถ้าเพิ่มตัวแปรที่มีอยู่แล้วในแกรมม่า ก็จะมี cost น้อยกว่าการเพิ่มตัวแปรใหม่เข้าไป ทำเช่นนี้ไปสำหรับทุกๆสตริงตัวอย่าง



รูปที่ 3.10 แสดงตัวอย่างการสร้างแกรมม่าเริ่มต้นจากสตริงตัวอย่าง

จากงานวิจัยนี้ มีแนวคิดที่น่าสนใจ กล่าวคือ สามารถกำหนดรูปแบบของกฎได้เอง เพื่อให้เหมาะสมกับรูปแบบของปัญหา จึงได้นำวิธีการนี้มาใช้ โดยการโมเดลใหม่ ให้อยู่ในรูป DFA

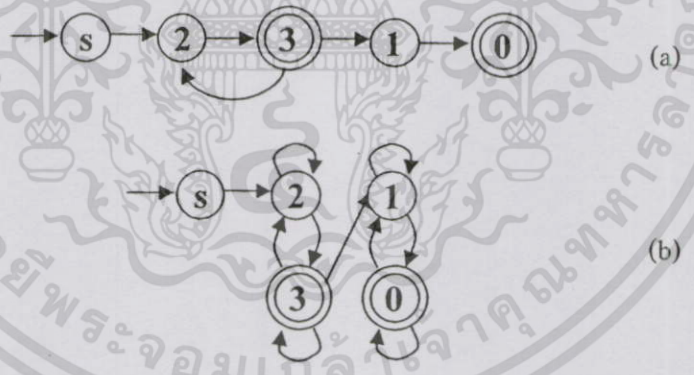
แทนที่จะเป็น Context-free Grammar แต่งานวิจัยนี้ยังไม่สามารถแยกแยะสตริงของแต่ละคลาสได้ เพราะสร้างโมเดลจากสตริงที่เป็นบวกเท่านั้น จึงนำแนวความคิดในการใช้สตริงที่เป็นลบ มาช่วยให้สามารถรู้จำได้ดียิ่งขึ้น รวมทั้ง นำข้อมูลตำแหน่งและความยาว และ การกำหนดลำดับของสเตทมาช่วยในการสร้างโมเดล ซึ่งจะกล่าวถึงหลักการของวิธีใหม่ที่จะนำเสนอในบทต่อไป

บทที่ 4
การสร้างสเตทแมทชีนแบบอัตโนมัติเพื่อการรู้จำสตริง

ปัญหาในการสร้างโมเดลเพื่อรู้จำสตริงใดๆนั้น คือ จะทำอย่างไรให้โมเดลที่สร้างขึ้นมีลักษณะทั่วไปมากพอที่จะยอมรับสตริงที่มีรูปแบบคล้ายคลึงกันได้ แต่จะต้องเจาะจงมากพอที่จะไม่ยอมรับสตริงจากคลาสอื่นๆด้วย งานวิจัยในวิทยานิพนธ์ฉบับนี้จะเป็นวิธีการสร้างสเตทแมทชีนแบบอัตโนมัติ ซึ่งได้จากสตริงตัวอย่างทั้งสตริงที่เป็นบวก และสตริงที่เป็นลบ

จุดเด่นในงานวิจัยนี้ คือ เป็นวิธีการสร้างโมเดลที่สามารถควบคุมรูปแบบของโมเดล ให้มีโครงสร้างเป็นไปตามต้องการได้ ซึ่งใช้โครงสร้างสเตทตามรูปแบบใน [1] นอกจากนั้น ยังนำตัวอย่างสตริงทั้งสตริงที่เป็นบวก และสตริงที่เป็นลบ มาใช้สร้างโมเดล ซึ่งจะช่วยเพิ่มประสิทธิภาพในการรู้จำได้อีกด้วย

การสร้างสเตทแมทชีนโดยทั่วไปนั้น มักจะใช้วิธีการยุบทรีที่สร้างจากสตริงตัวอย่างทั้งหมด ให้กลายเป็น DFA ที่เล็กที่สุด โดยที่ไม่ให้ขัดต่อตัวอย่างที่กำหนดไว้ เช่น วิธี Trakhtenbrot – Barzdin [2], SAGE [3], EDSM [4] เป็นต้น ซึ่งสามารถอ่านรายละเอียดของขั้นตอนวิธีเหล่านี้ได้ในบทที่ 3 แต่วิธีการเหล่านี้ อาจจะไม่เหมาะกับการรู้จำสตริงบางประเภทที่มีรูปแบบเฉพาะ



รูปที่ 4.1 แสดงโมเดลเปรียบเทียบระหว่างวิธีอื่นๆและวิธีในงานวิจัยนี้

ในงานวิจัยนี้ จะสามารถกำหนดรูปแบบในการสร้างโมเดลให้มีลักษณะตามต้องการได้ โดยการนำสตริงที่ได้จากตัวอักษรภาษาไทยมาเป็นต้นแบบ ซึ่งจะแบ่งได้เป็น 2 รูปแบบด้วยกัน คือ Loop และ Loop Pair ตามงานวิจัยของปองเกษม [1] ซึ่งเป็นงานวิจัยต้นแบบ ซึ่งได้พิจารณารูปแบบจากลักษณะเฉพาะของสตริงตัวอย่างโดยตรง และสรุปเป็นรูปแบบของสเตทตามต้องการ ดังนั้น โมเดลที่ได้จึงเหมาะสมกับสตริงตัวอย่างได้มากกว่าโมเดลทั่วไป รูปที่ 4.1 จะแสดงความแตกต่างของ

โมเดลจากงานวิจัยนี้และวิธีอื่น ๆ ซึ่งสร้างมาจากสตริง "2310" และ "2323" รูปที่ 4.1(a) เป็นโมเดลที่ได้จากวิธีอื่นๆ ส่วนภาพที่ 4.1(b) เป็นโมเดลที่ได้จากวิธีในงานวิจัยนี้ ถ้านำโมเดลมารู้จำสตริงใหม่ "23310" โมเดลแรกจะปฏิเสธสตริง ในขณะที่โมเดลที่ 2 จะสามารถรับสตริงได้ จะเห็นว่าอินพุตที่เกินมาเพียงตัวเดียวนั้น อาจจะมีผลกระทบให้โมเดลนั้นยอมรับหรือปฏิเสธสตริงได้ ทั้งที่ตามปกติแล้ว เราจะไม่สนใจว่าอินพุตมีจำนวนที่แน่นอนกี่ตัว เพียงแค่อินพุตนั้นมีตำแหน่งอยู่ภายในช่วงที่กำหนดก็เพียงพอแล้ว

นอกจากนั้น ขั้นตอนวิธีนี้ยังได้นำข้อมูลตำแหน่งและความยาวของอินพุตมาใช้ เพื่อช่วยในการตัดสินใจ เพื่อเลือกว่าสเตทใดที่เหมาะสมในการเพิ่มทรานสิชั่นเข้าไปในโมเดล และยังนำมาเป็นข้อมูลในการรู้จำตัวอักษรในภายหลังอีกด้วย

แต่งานวิจัยเดิม จะทำการสร้างโมเดลขึ้นมาจากสตริงของคลาสเป้าหมายหรือเฉพาะสตริงที่เป็นบวกเท่านั้น แม้ว่าจะสามารถยอมรับสตริงในคลาสนั้นๆได้ แต่อาจจะไม่สามารถปฏิเสธสตริงจากคลาสอื่นได้ ในงานวิจัยนี้ จึงใช้สตริงจากคลาสอื่นมาเป็นข้อมูลในการสร้างสเตทกับดัก (Trap State) ซึ่งจะช่วยป้องกันไม่ให้โมเดลยอมรับสตริงอื่นได้นั่นเอง

ในบทนี้ จะอธิบายขั้นตอนวิธีที่ได้พัฒนาขึ้นใหม่อย่างละเอียด จากวิธีต้นแบบเดิมของปองเกษมที่ใช้ Context-free grammar แต่จะนำมาโมเดลใหม่ในรูปแบบของ DFA และเพิ่มในส่วนของ Negative Training รวมทั้งใช้ข้อมูลตำแหน่งและความยาวสตริงในการสร้างแบบจำลองและการรู้จำ

4.1 ภาพรวมของระบบการทำงาน

งานวิจัยนี้จะทำการรู้จำสตริงที่ได้จากตัวอักษรภาษาไทยโดยการสร้างแมทชีนขึ้นมาเพื่อรองรับสตริงที่ใช้แทนอักษรแต่ละตัว ก่อนที่จะสร้างแมทชีนนั้น จะต้องทำการแปลงรูปภาพของตัวอักษรที่ได้สแกนเข้ามาให้เป็นสตริงที่แมทชีนยอมรับได้เสียก่อน ในที่นี้สตริงจะประกอบด้วยลำดับของตัวเลข 0-7 ซึ่งแทนทิศทางทั้ง 8 ทิศในเชนโค้ด (Chain Code) หลังจากได้เชนโค้ดเป็นสตริงแล้ว ก็จะสร้างแมทชีนสำหรับตัวอักษรแต่ละตัว โดยนำตัวอย่างสตริงของแต่ละตัวอักษรมาสร้างเป็นแมทชีนทีละ 1 สตริง ทำเช่นนี้สำหรับทุกๆตัวอักษร ดังนั้น เราจะได้แมทชีน 1 แมทชีนต่อ 1 ตัวอักษร รูปที่ 4.2 จะแสดงภาพรวมของระบบการทำงานในการสร้างแมทชีนเพื่อรู้จำตัวอักษร

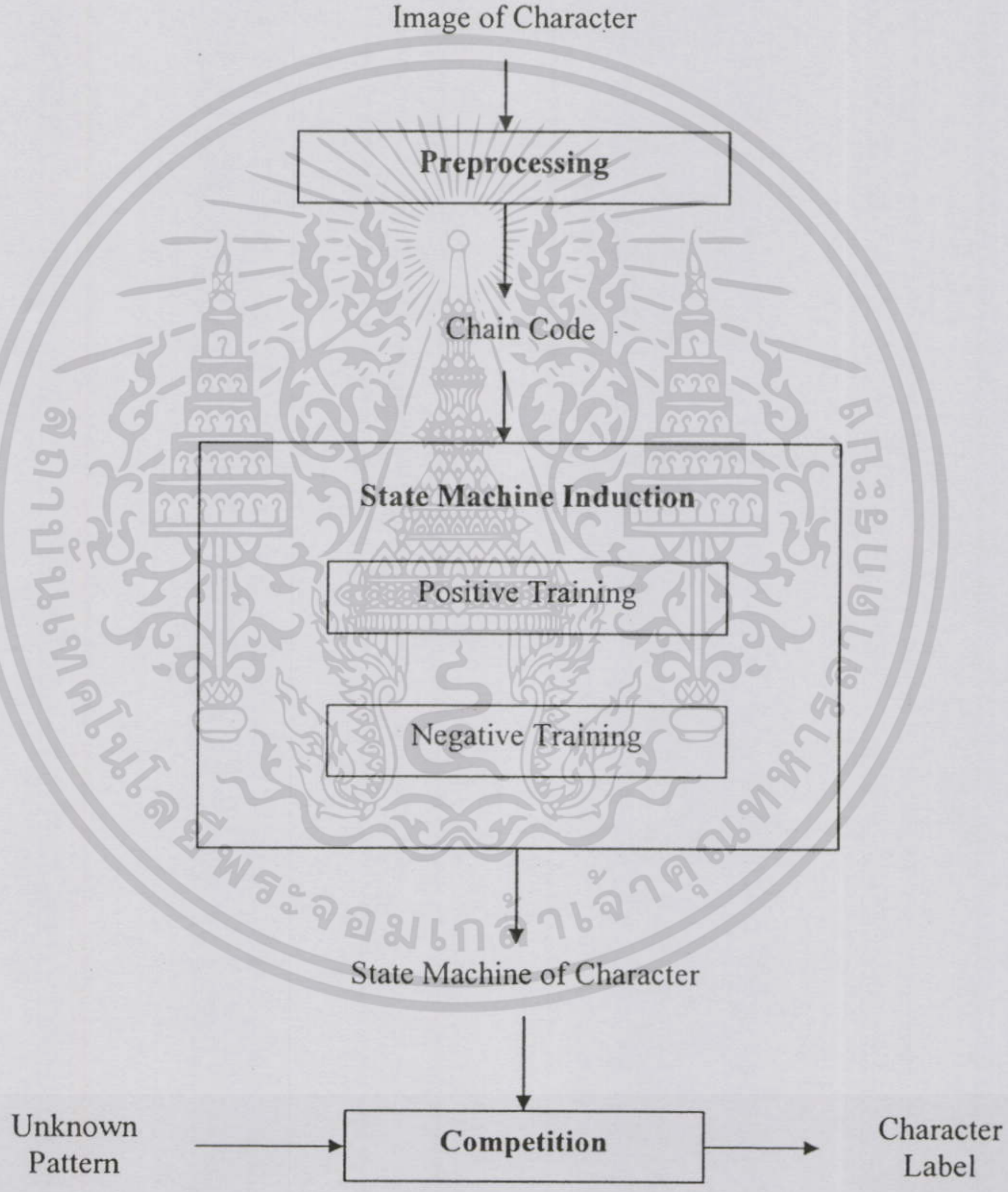
ขั้นตอนการรู้จำตัวอักษรโดยใช้สเตทแมทชีน เป็นดังนี้

1. *Preprocessing phase* จะแปลงรูปภาพให้เป็นสตริงที่ประกอบด้วยตัวเลข 0-7 ใช้แทนทิศทั้ง 8 เพื่อเป็นสตริงตัวอย่างต้นแบบสำหรับการสร้างแมทชีนของแต่ละตัวอักษร
2. *State Machine Induction phase* จะทำการสร้างแมทชีนทีละตัวอักษร จากตัวอย่างสตริงของตัวอักษรนั้นๆนำมาสร้างเป็นสเตทแมทชีนที่สามารถยอมรับตัวอักษรนั้นได้ และนำสตริงตัวอย่างของตัวอักษรอื่นๆมาใช้ปรับแมทชีนให้ปฏิเสธอักษรตัวอื่นๆที่ไม่เกี่ยวข้อง แบ่งเป็น 2 ขั้นตอน ดังนี้

ก. *Positive Training* จะนำสตริงตัวอย่างของตัวอักษรเป้าหมายมาสร้างเป็นสเตทแมทชีนที่สามารถ accept ตัวอักษรนั้นได้

ข. *Negative Training* จะนำสตริงอื่นๆที่ไม่ใช่อักษรเป้าหมายมาสร้าง Trap State เพื่อให้แมทชีนนั้น reject ตัวอักษรอื่นๆที่ไม่ใช่ตัวอักษรเป้าหมาย

3. *Competition phase* จะนำสตริงที่ไม่ทราบว่าเป็นตัวอักษรใด มาผ่านเข้าไปในแต่ละแมทชีนที่สร้างขึ้น เพื่อค้นหาว่าสตริงนั้นเป็นสตริงของตัวอักษรใด



รูปที่ 4.2 แสดงภาพรวมของระบบในการสร้างแมทชีนสำหรับรู้จำตัวอักษร

สเตทแมทชีนที่ใช้ในงานวิจัยนี้จะเป็น Deterministic Finite Automata หรือ DFA ซึ่งจะสร้างขึ้นมาจากตัวอย่างสตริงที่อยู่ในคลาสนั้นๆ โดยจะมี 1 DFA ต่อ 1 คลาส จากนั้นจะนำสตริงที่ไม่ทราบว่าเป็นของคลาสใดมาทดสอบกับทุกๆ DFA เพื่อค้นหาคลาสที่สตริงนั้นควรจะเป็น

4.2 ตัวอย่างข้อมูลที่ใช้

ข้อมูลที่นำมาใช้ในงานวิจัยนี้ จะเป็นสตริง (String) ซึ่งจะประกอบด้วยสัญลักษณ์อินพุตเรียงต่อๆกัน สามารถดูนิยามและลักษณะทั่วไปของสตริงได้ในบทที่ 2

4.2.1 ลักษณะของสตริงตัวอย่าง

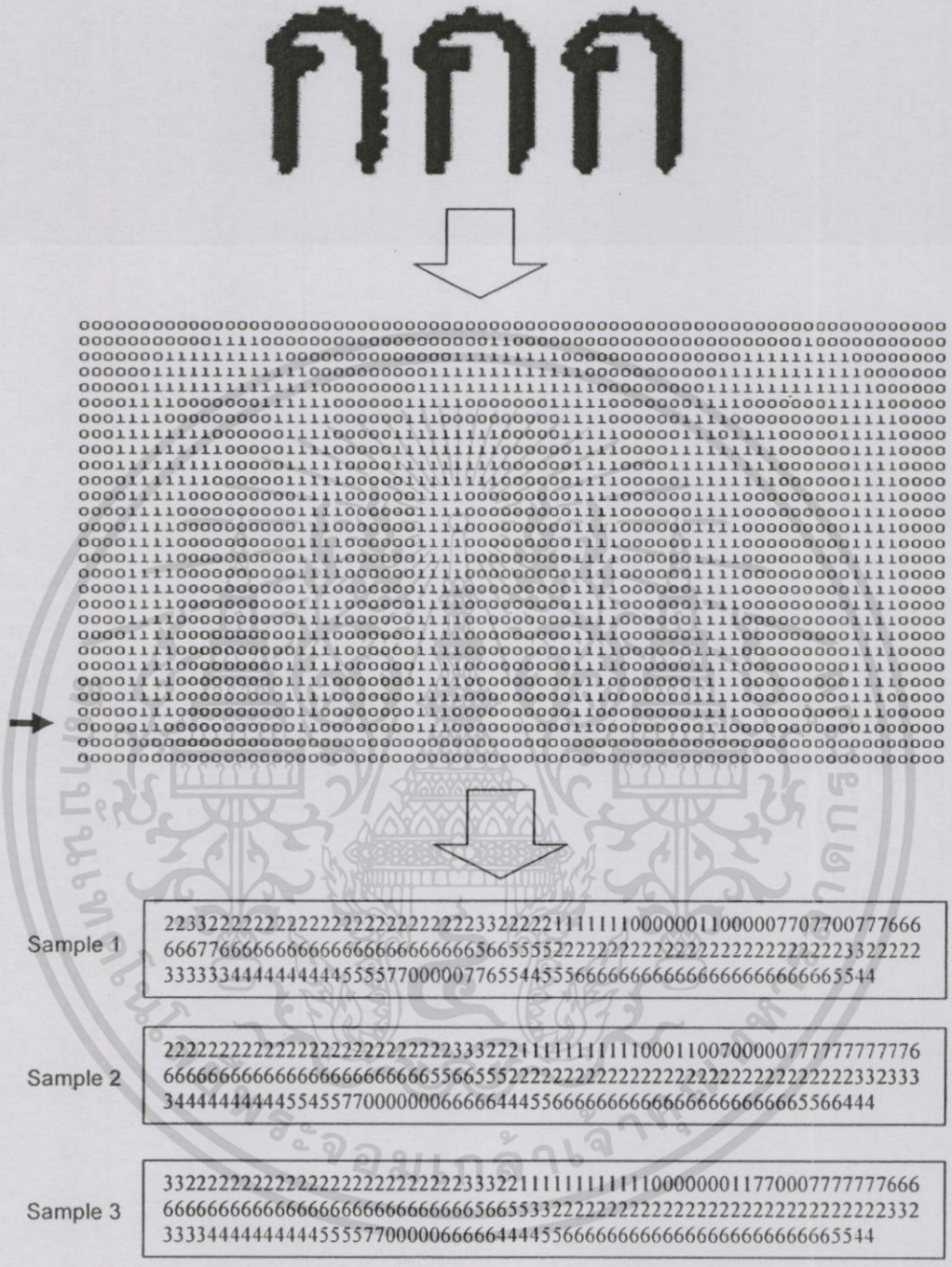
สตริงที่นำมาใช้เป็นตัวอย่างในการสร้างแมทชีน จะเป็นสตริงที่ได้จากตัวอักษรภาษาไทย โดยมีลักษณะเป็นเชนโค้ด (Chain Code) มี 8 ทิศทาง ซึ่งจะได้เป็นสัญลักษณ์อินพุต 0 – 7 แทนแต่ละทิศทางที่กำหนด สามารถดูวิธีการหาเชนโค้ดของสตริงตัวอย่างได้ในบทที่ 2

สตริงจะได้จากตัวอักษร ซึ่งสตริงที่เป็นตัวอักษรเดียวกัน จะถือว่าอยู่ในคลาสเดียวกัน จะใช้ทั้งสตริงที่เป็นบวก (Positive Sample) และสตริงที่เป็นลบ (Negative Sample) โดยจะเรียกสตริงใดๆว่าสตริงที่บวก ก็ต่อเมื่อ สตริงนั้นอยู่ในคลาสเดียวกันสตริงที่นำมาใช้สร้างแมทชีน หรือเป็นสตริงที่อยู่ในคลาสเป้าหมาย และจะเรียกสตริงอื่นๆว่า สตริงที่เป็นลบ ตัวอย่างเช่น เราจะสร้างแมทชีนสำหรับอักษร “ก” ดังนั้น สตริงทุกตัวที่ได้จากอักษร “ก” จะถือว่าเป็น สตริงที่เป็นบวกสำหรับแมทชีนนี้ และสตริงที่ได้จากตัวอักษรอื่นๆ “ข” ... “ฮ” รวมทั้งสระและวรรณยุกต์ต่างๆ จะถือว่าเป็นสตริงที่เป็นลบทั้งหมด เป็นต้น ตัวอย่างสตริงที่ใช้ในการทดสอบสามารถดูได้ใน ภาคผนวก ก.

4.2.2 การเตรียมข้อมูล

การเตรียมข้อมูลให้เป็นสตริงเพื่อนำมาใช้ในการสร้างแมทชีนนั้น จะทำการแสกนรูปภาพตัวอักษรตัวพิมพ์ภาษาไทยเป็นไฟล์ Bitmap ก่อน จากนั้น จะทำการหาเส้นขอบของตัวอักษร แล้วลากตามเส้นขอบของตัวอักษรจนมาบรรจบกับอินพุตตัวแรกอีกครั้ง เมื่อได้สตริงมาแล้ว ซึ่งอาจจะทำการปรับความยาวสตริงให้มีขนาดเท่ากันหรือไม่ก็ได้ ก็นำสตริงที่ได้มาเขียนรวมเป็น Text ไว้ในไฟล์เดียวกัน เพื่อนำมาเป็นต้นแบบในการสร้างแมทชีนต่อไป รูปที่ 4.3 จะแสดงตัวอย่างการแปลงข้อมูลรูปภาพให้เป็นสตริง

เมื่อได้ข้อมูลสตริงตัวอย่างทั้งหมดแล้ว จะนำมาแบ่งเป็น 2 ส่วน คือ Training Set สำหรับการทดลองในวิทยานิพนธ์นี้จะมีคลาสจะประมาณ 200 ตัวอักษร และที่เหลือจัดให้เป็น Test Set ทั้งหมด เหตุที่ให้จำนวน Training Set มีจำนวนประมาณ 200 ตัวอักษร เนื่องจากการทดสอบโดยใช้วิธี HMM นั้นใช้เวลามาก ซึ่งการใช้สตริงตัวอย่างจำนวนมาก ก็จะยิ่งเสียเวลาในการ Training นานมากขึ้นไปด้วย



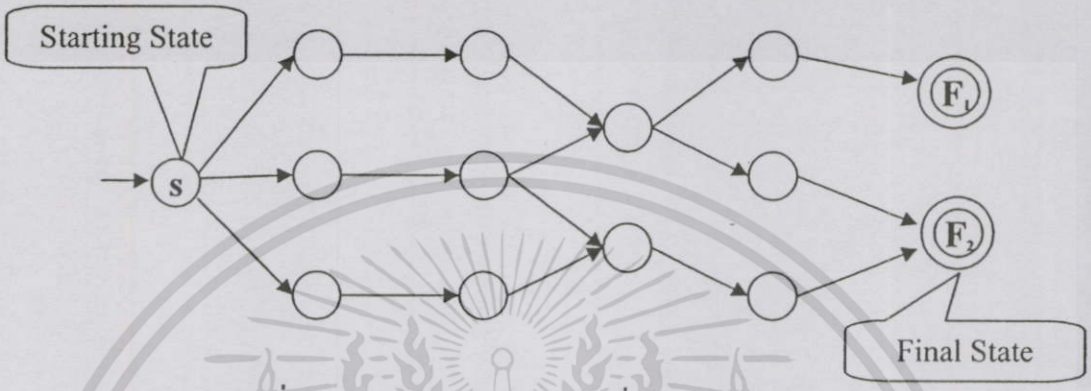
รูปที่ 4.3 แสดงตัวอย่างการแปลงข้อมูลรูปภาพให้เป็นสตริง

4.3 รูปแบบโครงสร้างของสเตท

เนื่องจากเราทราบว่าสตริงตัวอย่างนี้เป็นสตริงของตัวอักษรภาษาไทย จึงได้มีการปรับแมทชีนให้ยืดหยุ่นมากขึ้น และเหมาะสมกับลักษณะของตัวอักษร ลักษณะโครงสร้างของสเตทในแมทชีนมีดังนี้

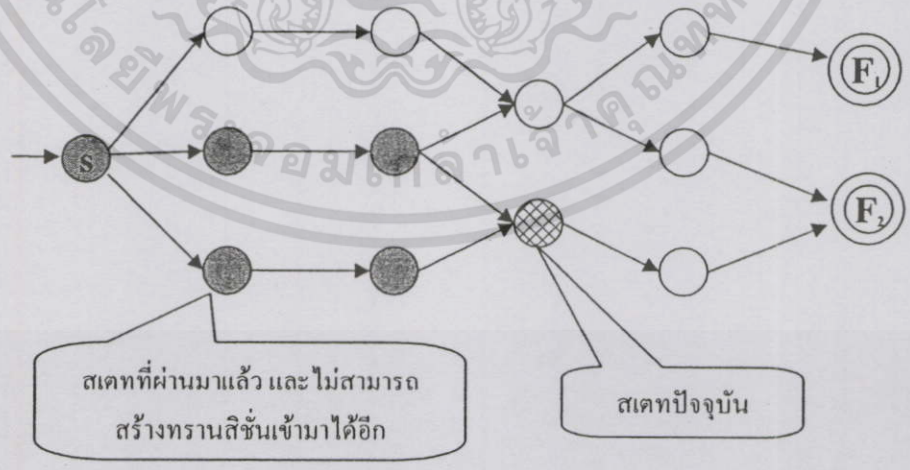
4.3.1 ลักษณะทั่วไปของสเตท

ทุกๆแมทชีนจะต้องเริ่มด้วยสเตทเริ่มต้น (Starting State) เพียงสเตทเดียว แทนด้วยสเตทที่มีสัญลักษณ์ S ภายในเสตท และมีลูกศรชี้เข้าที่สเตทนั้น ส่วนสเตทสุดท้าย (Final State) อาจจะมีกี่สเตทก็ได้ แทนด้วยสเตทที่มีวงกลมซ้อนกันสองวง ดังที่แสดงในรูปที่ 4.4



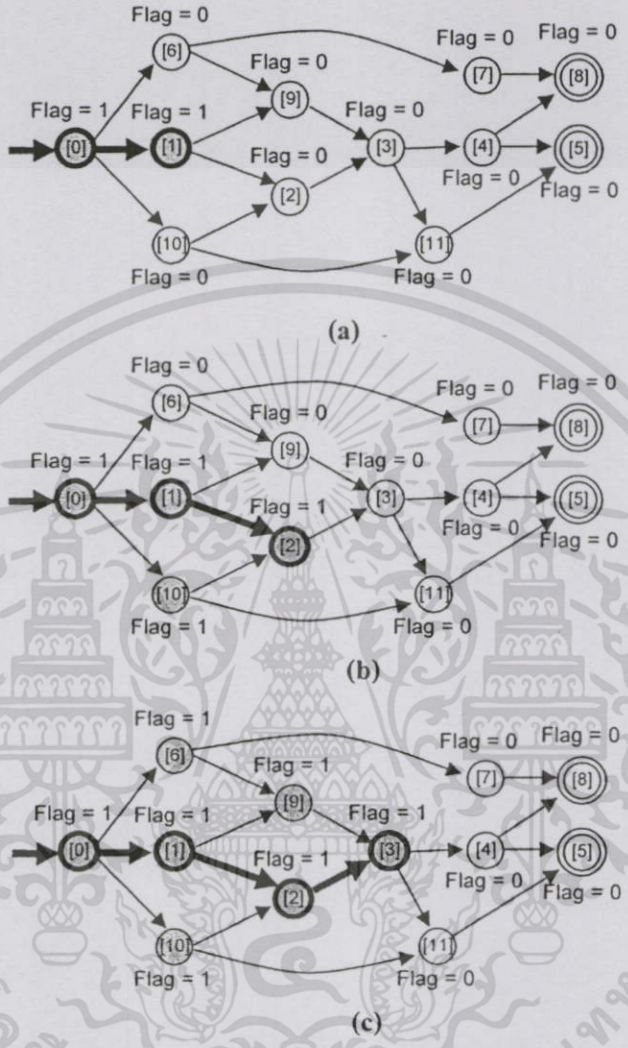
รูปที่ 4.4 แสดงลักษณะของสเตทเริ่มต้นและสเตทสุดท้าย

แมทชีนที่ได้จะไม่มีลูปย้อนกลับ (ยกเว้น Loop Pair ที่เป็นคู่สเตท ซึ่งจะได้อธิบายต่อไป) การจะเพิ่มทรานสิชั่นหรือสเตทใดๆ จะต้องทำกับสเตทข้างหน้าที่ยังไม่ได้ผ่านมาแล้ว โดยเราจะสามารถทราบได้ว่า สเตทนั้นผ่านมาแล้วหรือไม่ โดยให้สเตทใดๆก็ตาม ที่เป็นโหนดที่อยู่เหนือขึ้นไปจากสเตทปัจจุบัน (Ancestor) จะถูกกำหนดเป็นสเตทที่ไม่สามารถสร้างทรานสิชั่นเข้าไปได้ จนกว่าจะหมดอินพุตทุกตัวของสตริงนั้น และไปเริ่มที่สเตทเริ่มต้นใหม่อีกครั้งหนึ่ง รูปที่ 4.5 จะแสดงว่าสเตทใดที่พิจารณาว่าไม่สามารถสร้างทรานสิชั่นเข้าไปได้



รูปที่ 4.5 แสดงแมทชีนที่ไม่มีลูปย้อนกลับ

การตรวจสอบว่าสเตทใดเป็นสเตทที่ผ่านมาแล้วหรือไม่นั้น สเตทที่ผ่านมาแล้ว คือ โหนดที่เป็นบรรพบุรุษ (Ancestor) ของโหนดปัจจุบันไล่ย้อนกลับขึ้นไปจนถึงโหนดรากแบบ Backtracking ตัวอย่างการตรวจสอบโหนดที่ผ่านมาแล้ว แสดงดังรูปที่ 4.6



รูปที่ 4.6 แสดงการตรวจสอบสเตทที่ไม่สามารถเพิ่มทรานสิชั่นได้

จากรูปที่ 4.6 ตัวเลขภายในโหนดแสดงลำดับการเกิดของโหนดตั้งแต่ [0] – [11] ค่า Flag ที่แต่ละโหนด เป็นการแสดงว่า เป็นโหนดที่เพิ่มทรานสิชั่นได้หรือไม่ โดย Flag = 0 แสดงว่าเป็นโหนดที่ไม่ได้อยู่ก่อนหน้า และสามารถเติมทรานสิชั่นเข้าไปได้ ส่วน Flag = 1 แสดงว่าเป็นโหนดที่เป็นบรรพบุรุษของโหนดปัจจุบัน ในตัวอย่างนี้ เส้นทางที่ผ่าน จะเริ่มต้นที่โหนด 0 ไปยังโหนด 1, 2 และ 3 ตามลำดับ (แสดงเส้นทางที่ผ่านด้วยลูกศรและโหนดที่หนากว่าปกติ) ก่อนเริ่มสตริงใหม่ ก็จะกำหนดค่า Flag ทั้งหมดให้เท่ากับ 0 หมายถึง ยังไม่มีสเตทใดที่ผ่านมาก่อนเลย ในรูปที่ 4.6(a) เมื่อไปถึงโหนดที่ 1 ก็จะกำหนดค่า Flag = 1 ที่โหนดตัวเอง และพ่อแม่ของมัน ในที่นี้คือ โหนด 0 ต่อมาเมื่อไปถึงโหนดที่ 2 จะได้โหนดที่เป็นบรรพบุรุษ คือ โหนด 0, 1 และ 10 รวมถึงโหนดตัวเองด้วย

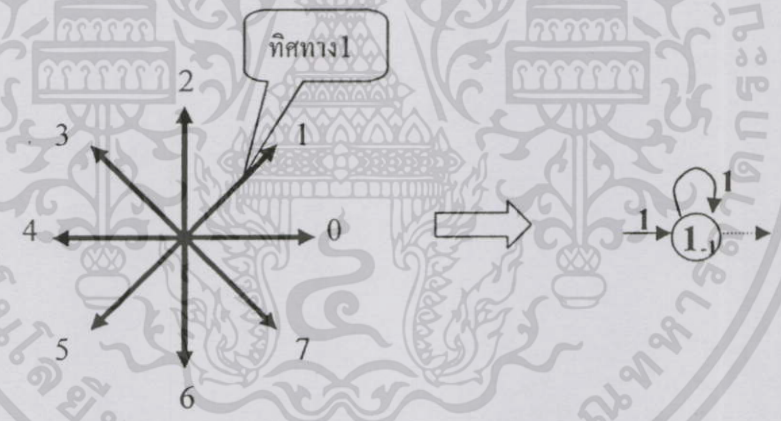
ดังรูปที่ 4.6(b) การกำหนดค่า Flag จะใช้วิธีการไล่จากโหนดปัจจุบันย้อนกลับขึ้นไปจนกว่าจะพบโหนดที่ถูกกำหนดให้ Flag = 1 จึงหยุด ซึ่งเมื่อพบโหนดที่ Flag = 1 แล้ว แสดงว่า โหนดก่อนหน้านั้น (โหนดบรรพบุรุษของมัน) ย่อมถูกเซตให้ Flag = 1 ทั้งหมดแล้ว ทำเช่นนี้ไปเรื่อยๆจนกว่าเข้าเยี่ยมโหนดจนถึงสเตทสุดท้าย และเมื่อเริ่มสตริงใหม่ ก็จะล้างค่า Flag ทั้งหมดให้เท่ากับ 0 ดังเดิม

นอกจากนี้ เรายังสามารถนำข้อมูลตำแหน่งของอินพุตในสตริง และตำแหน่งของสเตทมาเปรียบเทียบ เพื่อค้นหาทรานสิชั่นที่เหมาะสม และมีตำแหน่งใกล้เคียงกันได้อีกทางหนึ่งด้วย ซึ่งจะกล่าวถึงรายละเอียดต่อไป

4.3.2 โครงสร้าง Loop และ Loop Pair

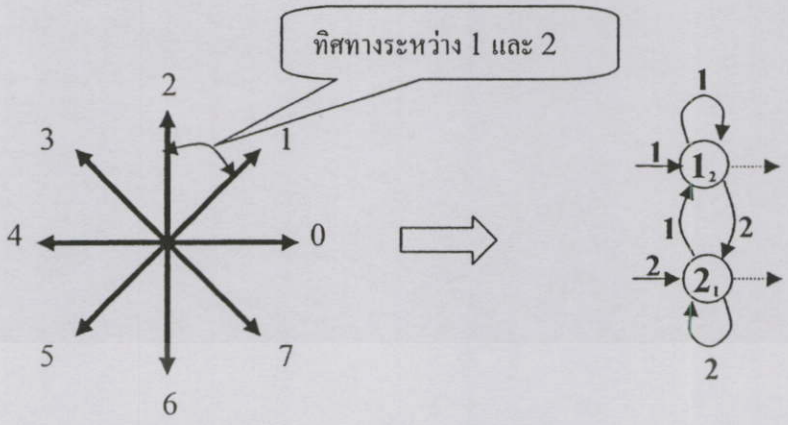
จากลักษณะของสตริงที่ได้จากทิศทางของเส้นขอบ จึงสามารถวิเคราะห์ได้เป็น 2 กรณี คือ อินพุตที่เกิดจากทิศทางที่กำหนดใน 8 ทิศนั้น และอินพุตที่เกิดในทิศทางที่อยู่ระหว่างทิศที่กำหนด ดังนั้น จึงกำหนดรูปแบบสเตทที่จะสร้างขึ้น ไว้เป็น 2 รูปแบบ เรียกว่า Loop และ Loop Pair [16] มีลักษณะดังนี้

1) สร้างลูป (Loop) สำหรับทุกๆสเตท เมื่ออินพุตเป็นทิศทางเดียวกันซ้ำๆ เช่น 1, 11, 111, 1111, 1111, ... หรือ 1*



รูปที่ 4.7 แสดงการสร้างลูปสำหรับอินพุตของทิศทางเดียวกัน

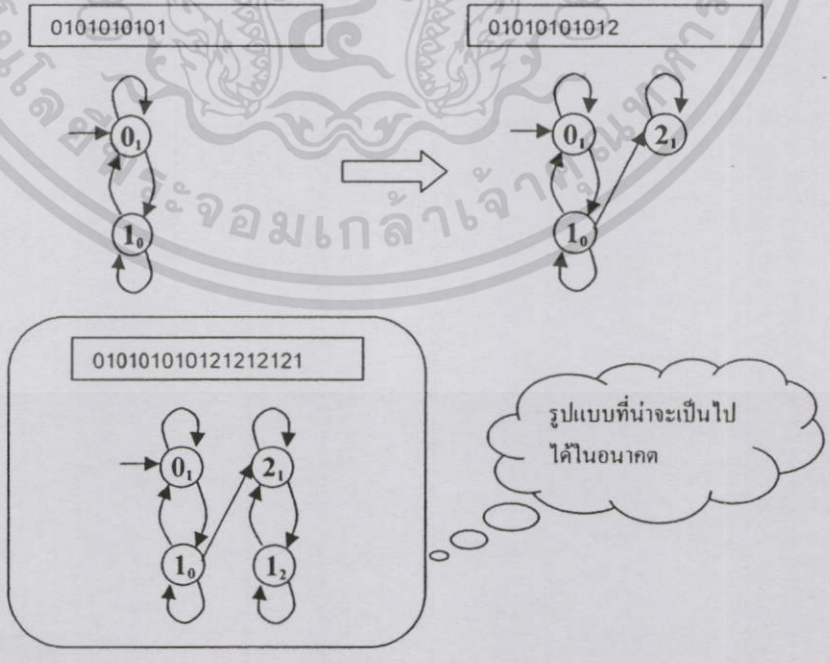
2) สร้างคู่สเตท (Loop Pair) สำหรับอินพุตที่มีทิศทางอยู่ติดกัน ใช้แทนทิศทางที่อยู่ระหว่างทั้ง 2 ทิศนี้ เช่น 12, 121, 1121, 122212112, 111221212211, ... หรือ (1*2*)*



รูปที่ 4.8 แสดงการสร้างคู่สเตทสำหรับอินพุตที่มีทิศทางอยู่ติดกัน

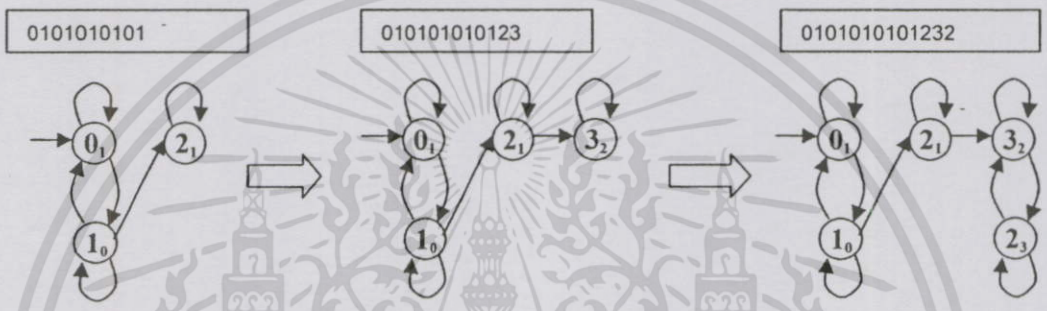
4.3.3 การกำหนดคู่สเตทล่วงหน้า

นอกจากการกำหนดรูปแบบของสเตทในแมทชีนแล้ว จะมีการกำหนดคู่สเตทล่วงหน้า ก่อนที่จะมีการสร้างสเตทจริงได้ ถ้าอินพุตตัวถัดไปกับตัวก่อนหน้ามีทิศอยู่ติดกัน เช่น ลำดับสตริงเป็น 0 1 2 จะเห็นว่าอินพุต 2 ตัวแรกจะทำให้เป็นสเตทคู่ก่อน เมื่อพบ 2 จะสร้างสเตทใหม่ แม้ว่าจะยังสร้างคู่สเตทไม่ได้ แต่จะกำหนดไว้ก่อนว่าคู่สเตทของมันจะต้องเป็นสเตทที่อินพุตเป็น 1 เท่านั้น เผื่อลำดับของสตริงในอนาคตจะเป็น 01010121212... เป็นต้น ในที่นี้ ตัวเลขที่ห้อยอยู่ภายในสเตทจะแทนสัญลักษณ์ของคู่สเตทล่วงหน้า ซึ่งค่านี้ จะต้องเป็นทิศทางที่อยู่เดียวกันกับสเตทนั้นๆเท่านั้น เช่น สเตทที่แทนอินพุตเป็น 1 ก็จะมีคู่สเตทเป็นอินพุตเท่ากับ 0 หรือ 2 ได้เท่านั้น และถ้าคู่สเตทมีค่าเป็น -1 แสดงว่า ไม่มีคู่สเตทที่เหมาะสม และสามารถกำหนดคู่สเตทเป็นอะไรก็ได้



รูปที่ 4.9 แสดงการกำหนดคู่สเตทล่วงหน้า

จากรูปที่ 4.9 ในกรณีที่อินพุตตัวต่อมาเป็น 3 (สมมติให้สตริงตัวอย่างเป็น 01010101012323...) ก็จะทำการสร้างสเตทใหม่สำหรับอินพุตเท่ากับ 3 ขึ้นมา และเนื่องจากสเตทก่อนหน้านั้นมีค่าอินพุตเท่ากับ 2 ซึ่งมีทิศทางอยู่ติดกันกับ 3 จึงกำหนดค่าคู่สเตทให้เป็น 2 แต่เนื่องจากสเตทก่อนหน้านั้นมีคู่สเตทเป็น 1 อยู่แล้ว ดังนั้น จึงไม่สามารถสร้างเป็นคู่สเตท 3 และ 2 ได้ และต้องต่อสเตทใหม่ออกไป ดังรูปที่ 4.10 จะเห็นว่าสเตทใหม่จะถูกกำหนดคู่สเตทไว้ก่อนว่ามีค่าเท่ากับ 2 แต่ไม่ได้เป็นคู่สเตทกับสเตทก่อนหน้านี้ เพราะสเตทก่อนหน้าได้ถูกกำหนดค่าคู่สเตทเป็นตัวอื่นไปแล้ว เป็นการจองตำแหน่งเอาไว้ในกรณีที่ในอนาคตมีอินพุตเท่ากับ 1 มาแทรกกลางนั่นเอง และเมื่ออินพุตตัวต่อไปเป็น 2 จึงสร้างเป็นคู่สเตท 3 และ 2 เพราะมีค่าคู่สเตทตรงกัน

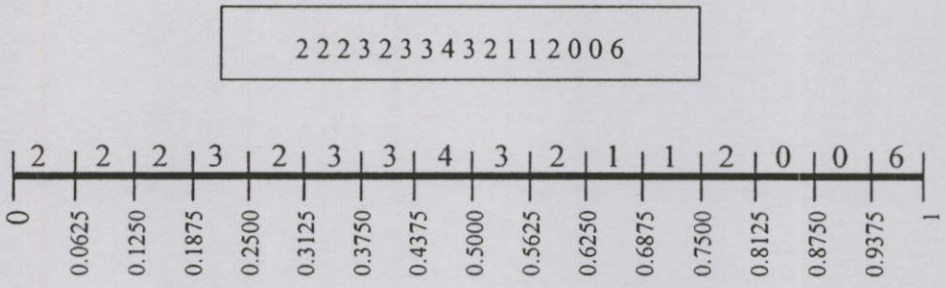


รูปที่ 4.10 แสดงการกำหนดคู่สเตทล่วงหน้าสำหรับอินพุตตัวถัดไป

การกำหนดคู่สเตทในลักษณะนี้จะส่งผลดีในการเพิ่มสเตทหรือทรานสิชั่นในภายหลัง ไม่ให้เกิดความสับสน และสามารถเติมได้ง่าย เนื่องจากปัญหาในการเพิ่มสเตทหรือทรานสิชั่นในงานวิจัยก่อนหน้านี้ การสร้างคู่สเตททำไปโดยมิได้คำนึงถึงการเกิดสเตทในลักษณะเส้นโค้ง ที่มีลำดับทิศทางค่อยๆเปลี่ยนไปทีละน้อย หรือมีทิศติดกันไปเรื่อยๆ เช่น 01012123234345 จะเห็นว่า จะมีลักษณะเป็นคู่สเตทติดต่อกัน แต่ในสตริงตัวอย่างนั้น อาจจะมีอินพุตที่กระโดดข้ามไปบ้าง มิได้เรียงเช่นนี้ เมื่อมาเพิ่มสเตทหรือทรานสิชั่นในภายหลัง ทำให้มีการเติมสเตทมากเกินจำเป็น และไม่สามารถนำสเตทที่มีอยู่เดิมมาใช้ซ้ำอีกครั้งได้ เพราะเกิดเส้นทางใหม่ไปแล้ว ดังนั้น การกำหนดลำดับเอาไว้ก่อน จึงช่วยแก้ปัญหาความสับสนในช่วงเส้นโค้งของตัวอักษรได้

4.3.4 ข้อมูลตำแหน่งและความยาว

จะมีการกำหนดตำแหน่งที่เป็นไปได้ภายในสตริงของแต่ละสเตท ข้อมูลตำแหน่งนั้นจะคิดเป็นอัตราส่วนจากความยาวสตริงทั้งหมด เพื่อให้สามารถใช้ได้กับสตริงที่มีขนาดความยาวแตกต่างกัน ข้อมูลตำแหน่งนี้จะทำให้ทราบว่าอินพุตที่เข้ามานั้นอยู่ในสเตทที่ถูกต้องหรือไม่ และทำให้ทราบสเตทที่ควรจะไป ถ้าต้องการเพิ่มทรานสิชั่น

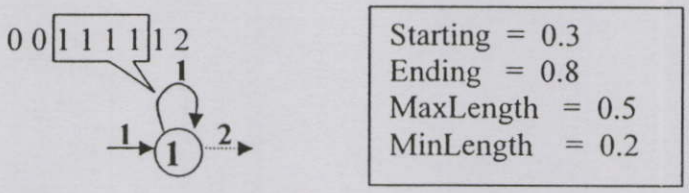


รูปที่ 4.11 แสดงตำแหน่งของสตริง

สำหรับในงานวิจัยนี้ จะเพิ่มข้อมูลในส่วนของตำแหน่งและความยาวอินพุตเข้าไป ซึ่งตำแหน่งทั้งหมดจะถูกแปลงให้อยู่ในช่วง [0, 1] ดังรูปที่ 4.11 ข้อมูลที่เพิ่มเข้ามาในแต่ละสเตท มีดังต่อไปนี้

- *Starting point* เป็นตำแหน่งน้อยที่สุดของอินพุตตัวแรกที่จะเข้ามายังสเตทนั้น
- *Ending point* เป็นตำแหน่งมากที่สุดของอินพุตตัวสุดท้าย ก่อนที่ย้ายไปยังสเตทถัดไป
- *MaxLength* เป็นช่วงความยาวของอินพุตที่มากที่สุด ที่มีอินพุตวนอยู่ภายในสเตทเดียวกัน ถ้าเป็นสเตทเดี่ยวไม่มีคู่สเตท ความยาวจะเป็นความยาวที่มากที่สุดที่วนอยู่ภายในสเตทนั้น ถ้าเป็นสเตทที่มีคู่สเตท ความยาวจะเป็นความยาวรวมของการวนอยู่ภายในคู่สเตท ดังนั้นในคู่สเตทจะมีค่าความยาวนี้เท่ากันทั้ง 2 สเตท ค่า MaxLength นี้ จะเป็นค่าความยาวที่เกิดขึ้น โดยจะมีค่าน้อยกว่าหรือเท่ากับ Ending Point – Starting Point เนื่องจาก ตำแหน่ง Starting Point และ Ending Point จะลดลง หรือเพิ่มขึ้นเรื่อยๆ เมื่อทำการ Training สตริงผ่านไป ซึ่งขึ้นอยู่กับตำแหน่งของอินพุตตัวแรก หรือตัวสุดท้ายที่เข้ามาที่สเตทนั้น
- *MinLength* เป็นช่วงความยาวของอินพุตที่น้อยที่สุด ที่มีอินพุตวนอยู่ภายในสเตทเดียวกัน ถ้าเป็นสเตทเดี่ยวไม่มีคู่สเตท ความยาวจะเป็นความยาวที่น้อยที่สุดที่วนอยู่ภายในสเตทนั้น ถ้าเป็นสเตทที่มีคู่สเตท ก็จะเป็นความยาวที่น้อยที่สุดที่เกิดขึ้นในการวนอยู่ภายในคู่สเตท

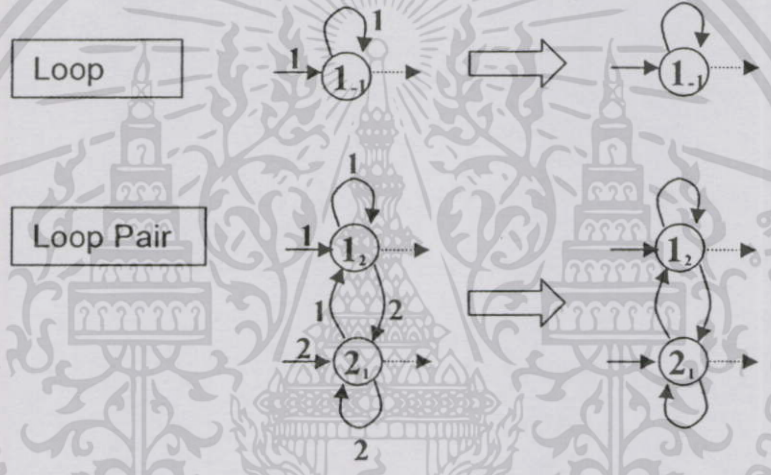
ตัวอย่างของสเตทที่อยู่ใน DFA แสดงในรูปที่ 4.12 หลังอินพุตตัวที่ 2 ที่มีค่าเป็น “0” อินพุตตัวถัดไปจะเป็น “1” ดังนั้น จะมีทรานสิชั่นจากสเตทก่อนหน้านี้ สำหรับอินพุตเท่ากับ “1” ไปยังสเตทใหม่ที่มีค่าเป็น “1” ถ้ามีอินพุต “1” ซ้ำอีก ก็จะวนซ้ำอยู่ที่สเตทเดิมไปจนกว่าจะมีอินพุตค่าอื่น หลังจากวนซ้ำที่สเตทเดิม 5 ครั้งแล้ว ก็จะเปลี่ยนไปยังสเตทต่อไป เมื่อมีอินพุตใหม่เท่ากับ “2”



รูปที่ 4.12 แสดงรูปแบบสเตทที่ใช้ในสเตทแมทชีน

เนื่องจากข้อมูลภายในแต่ละสเตทมีจำนวนมาก เมื่อแสดงภาพตัวอย่างการสร้างสเตทในขั้นตอนต่างๆอาจจะทำให้เกิดความสับสน จึงขอนิยามสัญลักษณ์ต่างๆที่ใช้ในการแสดงรูปสเตทแมทชีนดังต่อไปนี้

- สัญลักษณ์วงกลมแทนสเตทแต่ละสเตท ◯ และลูกศรแทนทรานสิชั่น ⟶
- สเตทเริ่มต้น (Starting State) คือ สเตทซึ่งมีสัญลักษณ์ S ภายใน ⟶(S)
- สเตทสุดท้าย (Final State จะมีวงกลมซ้อนกันสองวง ◎
- ตัวเลขภายในสเตท หมายถึงทิศทางของเสตทนั้นๆ และเลขห้อย หมายถึงคู่เสตท ถ้ามีเลขห้อยเป็น -1 หมายถึงไม่มีคู่สเตทสำหรับสเตทนั้น และเนื่องจากสเตทที่เป็นตัวแทนของทิศทางใด ทรานสิชั่นที่เข้ามาสเตทนั้น ก็ต้องมาจากอินพุตของทิศนั้นๆด้วย จึงขอละตัวเลขอินพุตบนลูกศรทรานสิชั่น ดังนั้น รูปแบบสเตทจะเป็นดังรูป



* เว้นแต่อินพุตสำหรับทรานสิชั่นนั้นจะเป็นอินพุตอื่น จะมีหมายเลขอินพุตกำกับ $\xrightarrow{4}$

- ตัวเลขในวงเล็บที่กำกับอยู่พร้อมสเตท เช่น (0, 0, 0) หมายถึง (Starting Point, Ending Point, MaxLength) ตามลำดับ ในที่นี้จะขอละข้อมูล MinLength เอาไว้ เนื่องจากวิธีการหาคล้ายคลึงกับ MaxLength

4.4 ขั้นตอนการสร้างสเตทแมทชีน

การสร้างแมทชีนจะแบ่งเป็น 2 ช่วง คือ สร้างแมทชีนจากสตริงของตัวอักษรเป้าหมาย (Positive Training) และการสร้างแมทชีนจากตัวอักษรอื่นๆ (Negative Training)

4.4.1 Positive Training

ในขั้นตอนนี้ จะเริ่มจากการสร้างแมทชีนเริ่มต้น จากสตริงตัวอย่างแรกสุดของตัวอักษรนั้นๆ ต่อจากนั้น นำสตริงตัวต่อๆไปมาปรับปรุงแมทชีน โดยการเพิ่มสเตทหรือทรานสิชั่นไปเรื่อยๆทีละ

สตริง จนกว่าจะครบทุกสตริงตัวอย่างที่มี ดังนั้น แมทชีนที่ได้จะยอมรับทุกๆสตริงตัวอย่างที่นำมาใช้สร้างแมทชีน อัลกอริทึมของขั้นตอนนี้จะแสดงในรูปที่ 4.13

```
//Positive Training
For each character class C
    Let M be the machine for class C
    Create starting state and add state in M

    For each pattern string P in class C
        Begin at the starting state
        For each input x in string P

            If there has transition for input x to the next state then
                Go to the next state
            Else
                Find the state represented for input x and has position in range
                If there exists the corresponding state then
                    Add transition from the current state to that state
                Else
                    Create new state s and add transition to new state
                    If input x is adjacent to the previous state then
                        If the previous state does not have its pair state OR
                        its pair state flag is input x then
                            Add transition to the previous state
                            Set the pair state flag for both state
                        Else
                            Set value of previous state as its pair state flag
                        End If
                    End If
                    Add new state in machine M
                End If
            End If
        End For //for all inputs in P
        Set the current state as final state
    End For //for all patterns in C
End For //for all character classes
```

รูปที่ 4.13 แสดงอัลกอริทึมของขั้นตอน Positive Training

เพื่อให้สามารถเข้าใจได้ง่าย จึงขออธิบายขั้นตอนวิธีการสร้างแมทชีนโดยใช้ตัวอย่างประกอบ กำหนดให้สตริงตัวอย่างเป็นดังนี้

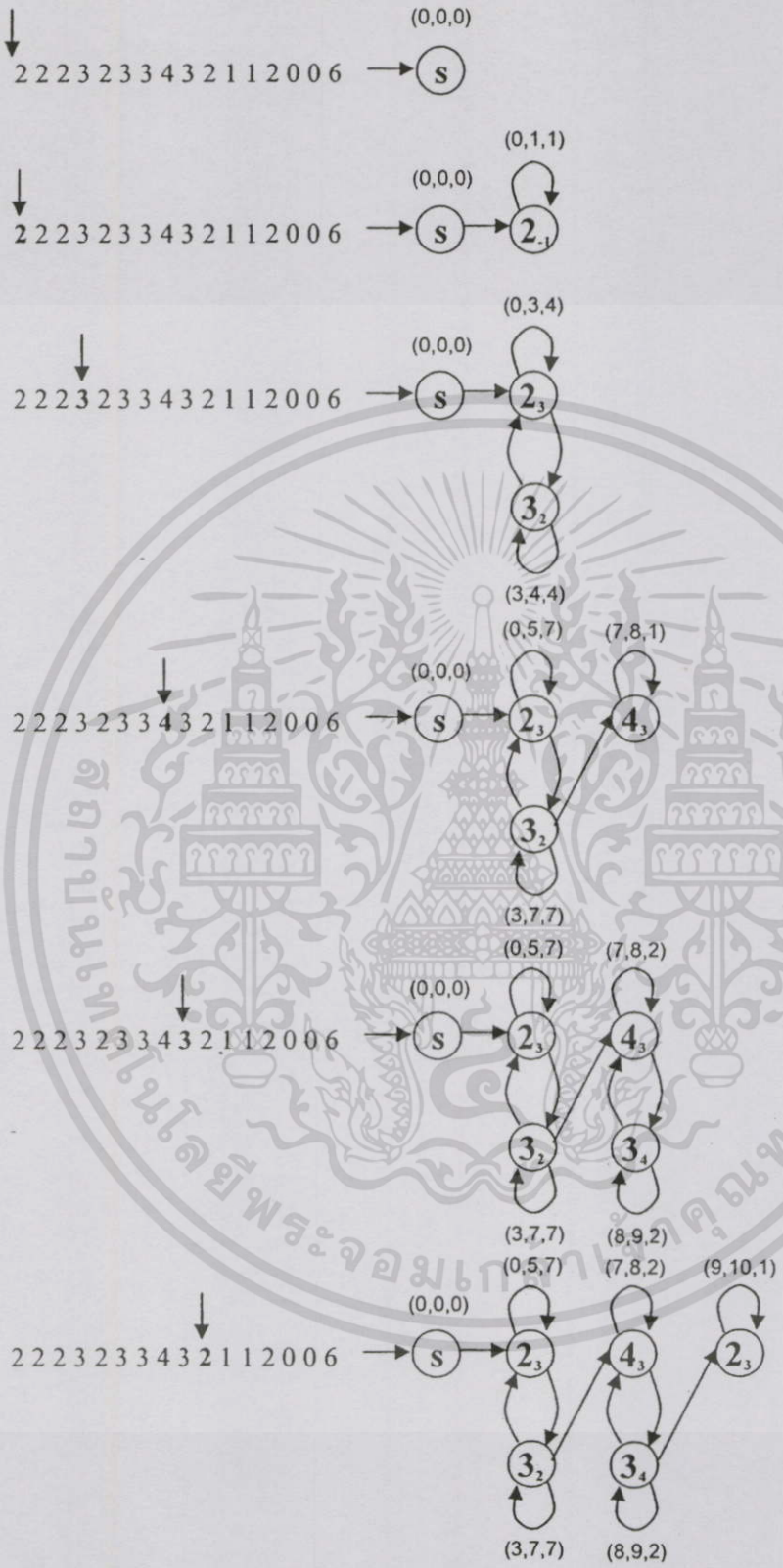
2	2	2	3	2	3	3	4	3	2	1	1	2	0	0	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ทั้งนี้ ในการทำงานจริง จะกำหนดให้สตริงมีความยาวเท่าใดก็ได้ไม่จำกัด และจะใช้สตริงมาสร้างแมทชีนเป็นจำนวนกี่สตริงก็ได้ โดยปกติแล้ว การกำหนดตำแหน่งของแต่ละอินพุตในสตริงจะมีค่าในช่วง [0, 1] แต่ในที่นี้ เพื่อให้ตัวเลขมีความกระชับ จะขอกำหนดตำแหน่ง เป็นค่า 0, 1, ..., 16 โดยตัวเลขในวงเล็บเหนือที่แต่ละสเตท จะแทนค่า Starting Point, Ending Poing และ Max Length ของสเตทนั้นๆ ค่าตำแหน่งเริ่มต้น (Starting Point) จะเป็นตำแหน่งที่อยู่ของอินพุตตัวปัจจุบัน ส่วนค่าตำแหน่งสิ้นสุด (Ending Point) จะเป็นตำแหน่งของอินพุตตัวถัดไป และความยาวมากที่สุด (Max Length) จะมีค่าอย่างน้อย 1 หรือเท่ากับอินพุตหนึ่งตัวเสมอ (เฉพาะตัวอย่างที่สาธิตในวิทยานิพนธ์ฉบับนี้เท่านั้น แต่ค่าจริงจะต้องเป็นเลขทศนิยม ตามสัดส่วนของความยาวสตริง) ส่วนค่า Min Length หรือค่าความยาวที่น้อยที่สุด ซึ่งจะมีการบันทึกไว้ในสเตทเช่นเดียวกับ Max Length จะขอละไว้สำหรับตัวอย่างนี้ แต่การเก็บค่าก็ทำลักษณะเดียวกับการเก็บค่า Max Length เพียงแต่ใช้ค่าความยาวน้อยที่สุดที่เกิดขึ้นในสเตทนั้นๆ หรือคู่สเตท ขั้นตอนในการสร้างแมทชีนจากสตริงแรกจะแสดงในรูปที่ 4.14 ซึ่งจะเป็นดังนี้

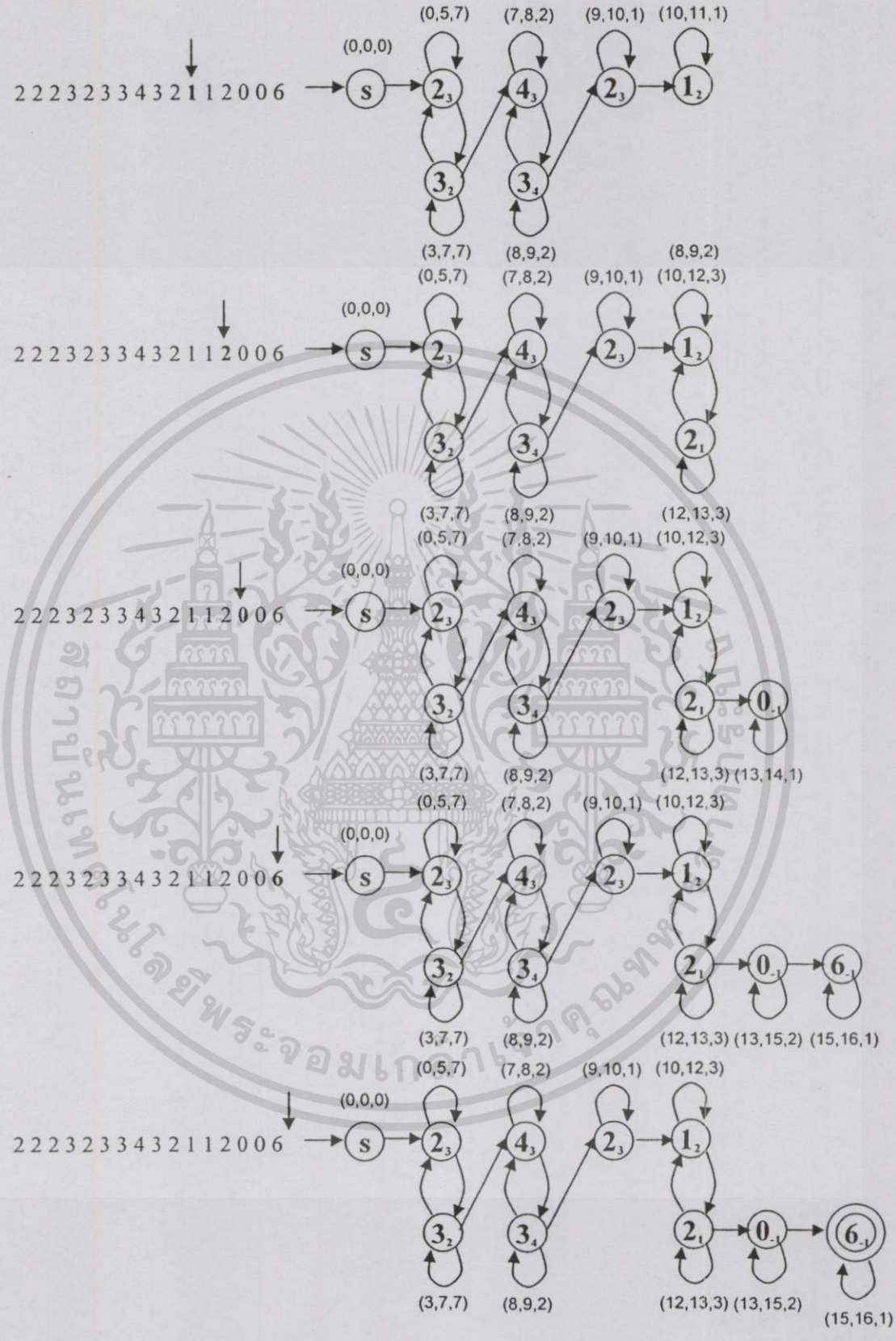
1. สร้างสเตทเริ่มต้น (Starting State) ซึ่งมีตำแหน่งเริ่มต้นและสิ้นสุด รวมทั้งความยาวเป็น 0 ทั้งหมด
2. ที่อินพุตตำแหน่งที่ 0 มีค่าเป็น 2 ให้สร้างสเตทใหม่ขึ้น ซึ่งจะแทนสถานะในการพบอินพุตเป็น 2 ซ้ำไปเรื่อยๆ (วนลูปเข้าหาตัวเองที่สเตทนี้) ค่าตำแหน่ง Ending Point และ MaxLength จะเพิ่มขึ้นเรื่อยๆเหมือนผ่านอินพุตแต่ละตัวไป จนกระทั่งพบอินพุตทิศทางอื่น
3. ที่อินพุตตำแหน่งที่ 3 มีค่าเป็น 3 ซึ่งเป็นทิศทางติดกับอินพุตตัวที่แล้ว (มีค่าอินพุตต่างกันอยู่ 1) และสเตทเดิมไม่ได้มีคู่สเตทอยู่ก่อน (คู่สเตทมีค่าเป็น -1) จะเพิ่มสเตทสำหรับอินพุตเท่ากับ 3 และให้เป็นสเตทคู่ (Loop Pair) ดังนั้น จะได้ว่า ที่สเตท 2 จะมีคู่สเตทเป็น 3 และสเตท 3 จะมีคู่สเตทเป็น 2 แทนลำดับของ Input ที่มี 2 และ 3 ค่าตำแหน่งเริ่มต้นของสเตทนี้คือ ตำแหน่งที่ 3 แต่ความยาวจะเท่ากับความยาวรวมของทั้งสองสเตทที่เป็นคู่กัน โดยสเตทเดิมมีความยาว 3 และมีอินพุตเพิ่มอีกตัวหนึ่งที่ตำแหน่งนี้ เป็นความยาวรวมเท่ากับ 4 โดยจะต้องไปปรับค่าความยาวของคู่สเตท (สเตทที่มีค่าเป็น 2) ด้วย
4. ที่อินพุตตำแหน่งที่ 7 มีค่าเป็น 4 ซึ่งมีทิศทางติดกับอินพุตตัวที่แล้ว คือ 3 แต่สเตทก่อนหน้านั้นมีคู่สเตทอยู่แล้ว จึงไม่สามารถทำเป็นคู่สเตทได้อีก จึงสร้างสเตทใหม่เพิ่มสำหรับอินพุตเท่ากับ 4 แล้วกำหนดให้คู่สเตทเป็น 3 ล่วงหน้าไว้ก่อน เผื่อจะเพิ่มคู่สเตทในภายหลัง
5. ที่อินพุตตำแหน่งที่ 8 มีค่าเป็น 3 จึงสร้างสเตทใหม่สำหรับอินพุตเท่ากับ 3 และทำให้เป็นคู่สเตทกับสเตทก่อนหน้า ที่ได้กำหนดคู่สเตทไว้แล้ว ค่าตำแหน่งเริ่มต้นเป็น 8 และความยาวเป็นความยาวรวมของคู่สเตท คือ 2
6. ที่อินพุตตำแหน่งที่ 9 มีค่าเป็น 2 ซึ่งมีทิศทางติดกับอินพุตตัวที่แล้ว คือ 3 แต่สเตทก่อนหน้านั้นมีการกำหนดคู่สเตทอยู่แล้ว จึงไม่สามารถทำเป็นคู่สเตทได้อีก จึงสร้างสเตทใหม่เพิ่ม

สำหรับอินพุตเท่ากับ 2 แล้วกำหนดให้คู่สเตทเป็น 3 ล่วงหน้าไว้ก่อน เผื่อจะเพิ่มคู่สเตทในภายหลัง

7. ที่อินพุตตำแหน่งที่ 10 มีค่าเป็น 1 ก็เช่นเดียวกัน แม้ว่าสเตทก่อนหน้าจะไม่มีคู่สเตทจริงๆ แต่กำหนดไว้ว่าคู่ของมันต้องเป็นสเตทที่มีอินพุตเท่ากับ 3 จึงไม่สามารถทำเป็นคู่สเตทได้อีก ดังนั้น จะเพิ่มสเตทใหม่สำหรับอินพุตเท่ากับ 1 และกำหนดให้คู่สเตทเป็น 2 ไว้ล่วงหน้า
8. ที่อินพุตตำแหน่งที่ 12 มีค่าเป็น 2 ให้สร้างสเตทเพื่อเป็นคู่สเตทกับสเตทที่มีอินพุตเท่ากับ 1 และปรับค่าความยาวเป็นความยาวรวมของคู่สเตท คือ 3
9. ที่อินพุตตำแหน่งที่ 13 มีค่าเป็น 0 ซึ่งทิศทางไม่ได้ติดกับอินพุตก่อนหน้า จึงสร้างสเตทใหม่และไม่มีคู่สเตท ให้คู่สเตทเป็น -1
10. ที่อินพุตตำแหน่งที่ 15 มีค่าเป็น 6 ก็เช่นกัน สร้างสเตทใหม่สำหรับอินพุตเท่ากับ 6 และไม่มีคู่สเตทสำหรับสเตทนี้
11. จบอินพุตทุกตัวของสตริงนี้ แล้วกำหนดให้สเตท ณ ตำแหน่งปัจจุบัน (ในที่นี้ คือ สเตทที่มีอินพุตเป็น 6) เป็นสเตทสุดท้าย (Final State)



รูปที่ 4.14 แสดงขั้นตอนการสร้างแมทชีนในช่วง Positive Training จากสตริงแรก



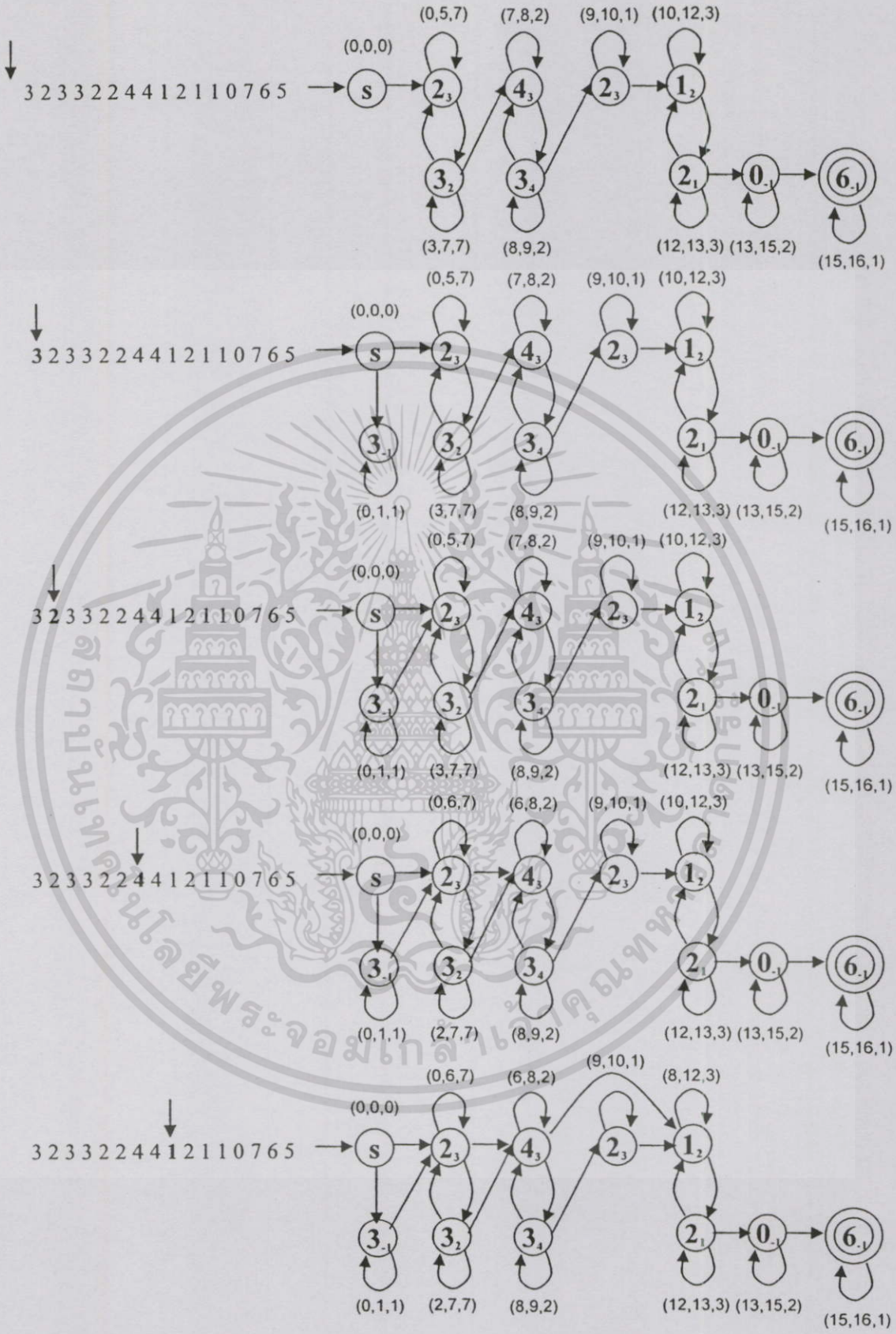
รูปที่ 4.14 (ต่อ)

หลังจากได้แมทชีนต้นแบบจากสตริงแรกมาแล้ว ก็จะนำสตริงตัวต่อๆไปมาทำการปรับปรุงแมทชีนเดิม ให้สามารถยอมรับสตริงนั้นได้ด้วย โดยการเติมสเตทใหม่ หรือเพิ่มทรานสิชั่นจากสเตทหนึ่งไปอีกสเตทหนึ่ง ซึ่งจะพยายามใช้แมทชีนเดิมให้มากที่สุด และเปลี่ยนแปลงแมทชีนให้น้อยที่สุดเท่าที่จะทำได้ รวมทั้งจะต้องมีการเปลี่ยนแปลงค่าตำแหน่งเริ่มต้น ตำแหน่งสิ้นสุด และความยาวอินพุตภายในแต่ละสเตท ถ้ามีค่าที่เปลี่ยนแปลงไปตามสตริงใหม่ที่รับเข้ามา ตัวอย่างเช่น ถ้าสตริงตัวที่ 2 มีลำดับอินพุตเป็นดังนี้

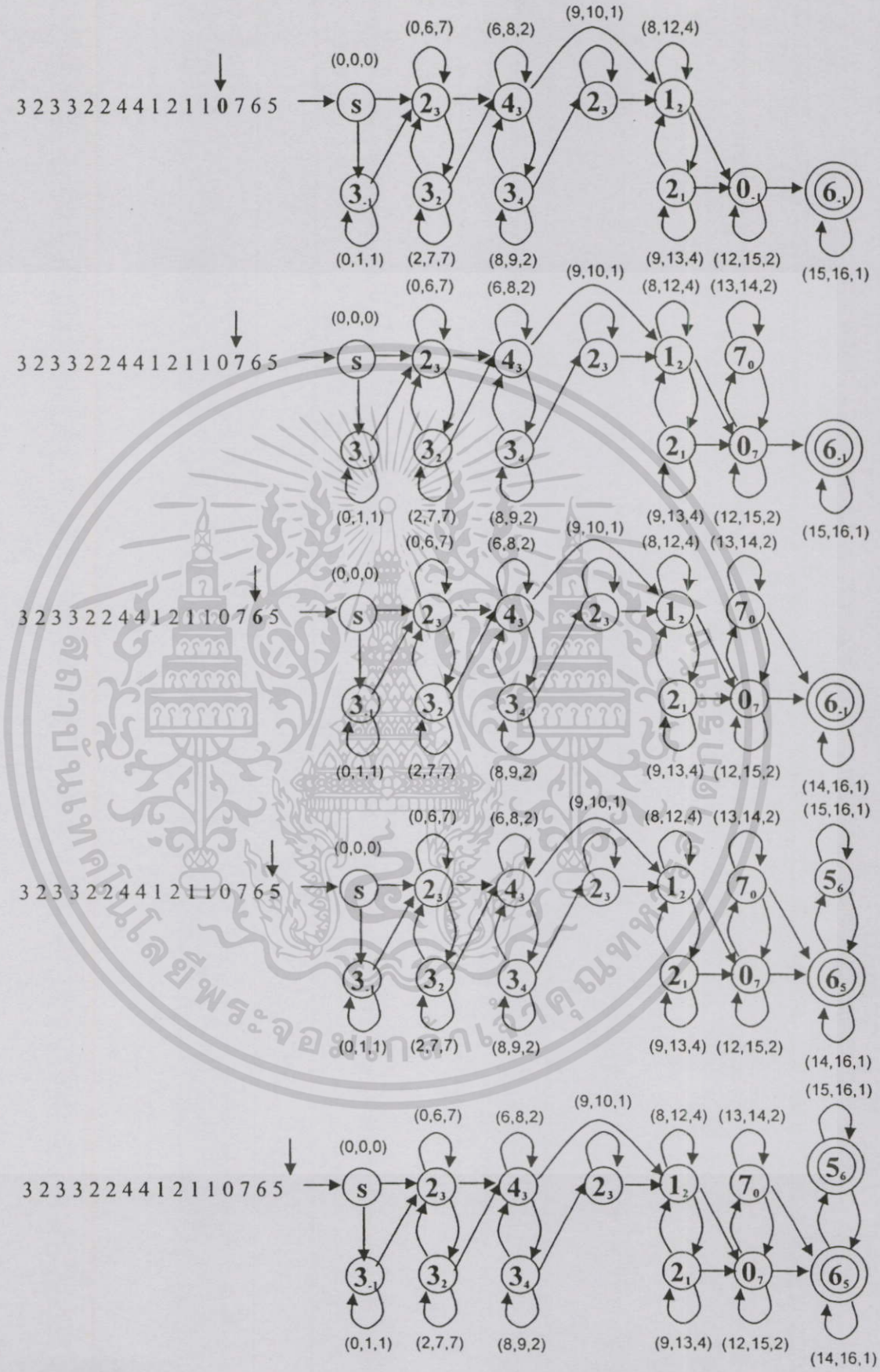
3	2	3	3	2	2	4	4	1	2	1	1	0	7	6	5
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

โดยจะมีลำดับการปรับปรุงแมทชีนเพื่อให้ยอมรับสตริงนี้ ดังรูปที่ 4.15 มีขั้นตอนดังต่อไปนี้

1. เริ่มการทำงานที่สเตทเริ่มต้น
2. ที่อินพุตตำแหน่งที่ 0 มีค่าเป็น 3 ซึ่งจากสเตทเริ่มต้นไม่มีทรานสิชั่นจะไปต่อได้ จึงค้นหาสเตทใกล้เคียงที่มีตำแหน่งในช่วงที่กำหนด เมื่อไม่พบสเตทที่มีตำแหน่งตรงกัน จึงตัดสินใจสร้างสเตทใหม่
3. ที่อินพุตตำแหน่งที่ 1 มีค่าเป็น 2 ซึ่งไม่มีทรานสิชั่นออกจากสเตทเดิม จึงค้นหาสเตทใกล้เคียง พบสเตทที่มีอินพุตเท่ากับ 2 และตำแหน่งอยู่ภายในช่วงเดียวกัน ในที่นี้ จะพบสเตทที่มีตำแหน่งเริ่มต้นเป็น 0 และตำแหน่งสิ้นสุดเป็น 5 ซึ่งตำแหน่งปัจจุบันก็อยู่ภายในช่วงนี้พอดี จึงทำการเพิ่มทรานสิชั่นจากสเตทเดิมที่มีอินพุตเท่ากับ 3 ไปยังสเตทนี้
4. สำหรับอินพุตถัดไป คือ 3 มีเส้นทางหรือทรานสิชั่นที่จะไปยังสเตทต่อไปได้ จึงไม่ต้องเปลี่ยนแปลงแมทชีน เช่นเดียวกับอินพุตที่เป็น 3 2 2 ตามลำดับ แต่จะมีการเปลี่ยนค่าตำแหน่งเริ่มต้น และสิ้นสุดไปตามตำแหน่งใหม่
5. เมื่ออินพุตเป็น 4 จึงเพิ่มทรานสิชั่นจากสเตทเดิมที่มีค่าเป็น 2 ไปยังสเตทที่มีอินพุตเท่ากับ 4
6. เมื่ออินพุตเป็น 1 จะค้นหาสเตทที่มีค่าเป็น 1 และมีตำแหน่งอยู่ที่ 8 จากนั้น เพิ่มทรานสิชั่นจากสเตทเดิมข้ามไปเช่นกัน
7. เมื่ออินพุตเป็น 0 จะเพิ่มทรานสิชั่นจากสเตทเดิมไปยังสเตทที่อินพุตเท่ากับ 0
8. เมื่ออินพุตเป็น 7 ไม่พบสเตทใกล้เคียง จึงสร้างสเตทใหม่ที่มีอินพุตเท่ากับ 7 ซึ่งมีทิศทางอยู่ติดกับอินพุตก่อนหน้า คือ 0 ดังนั้นจึงทำให้เป็นคู่สเตท
9. เมื่ออินพุตเป็น 6 จึงเพิ่มทรานสิชั่นจากสเตทเดิมไปยังสเตทที่อินพุตเท่ากับ 6
10. เมื่ออินพุตเป็น 5 ไม่พบสเตทใกล้เคียง จึงสร้างสเตทใหม่ที่มีอินพุตเป็น 5 ซึ่งมีทิศทางอยู่ติดกับอินพุตก่อนหน้า คือ 6 ดังนั้นจึงทำให้เป็นคู่สเตท
11. จบสตริงแล้วกำหนดให้สเตทปัจจุบันที่มีอินพุตเท่ากับ 5 เป็นสเตทสุดท้าย (Final State) ในขณะนี้ แมทชีนจะมีสเตทสุดท้าย 2 สเตท



รูปที่ 4.15 แสดงการสร้างสเตทแมทชีนในช่วง Positive Training ในสตริงถัดไป



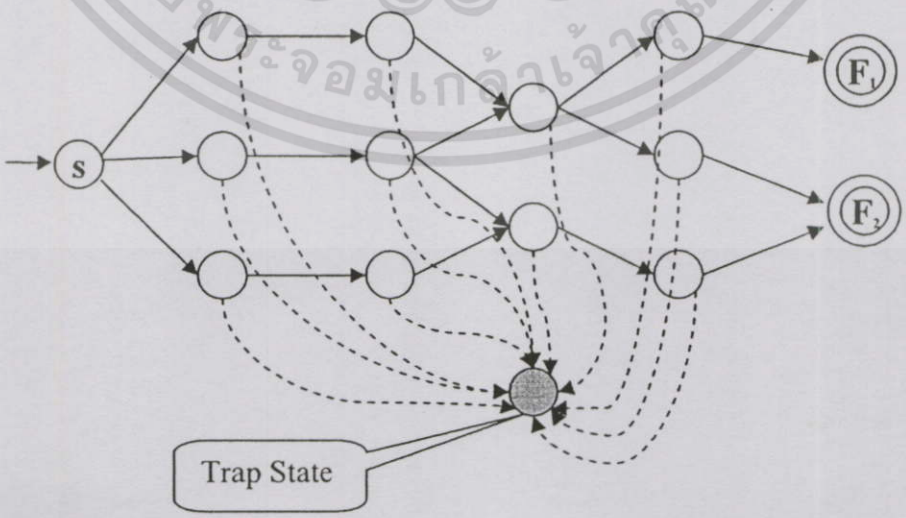
รูปที่ 4.15 (ต่อ)

สรุปขั้นตอนการปรับปรุงแมทชีนให้ยอมรับสตริงถัดๆไปด้วย จะทำดังต่อไปนี้

1. เริ่มจากสเตทเริ่มต้นทุกครั้งที่เริ่มสตริงใหม่
2. รับอินพุตเข้าไปทีละตัว ถ้ามีทรานสิชั่นไปยังสเตทต่อไปได้ ก็ให้ย้ายไปยังสเตทต่อไปไปเรื่อยๆ ในกรณีที่ไม่มีทรานสิชั่นจะไปต่อได้ จึงจะพิจารณาเพิ่มทรานสิชั่นไปยังสเตทอื่นที่อินพุตเป็นตัวเดียวกัน และอยู่ในช่วงตำแหน่งสตริงที่เป็นไปได้ของสเตทนั้น
3. ถ้าไม่มีสเตทที่จะเพิ่มทรานสิชั่นได้ เพราะตำแหน่งสตริงไม่อยู่ในช่วงที่เป็นไปได้ จะพิจารณาสเตทที่มีช่วงตำแหน่งใกล้เคียงกัน (ซึ่งไม่เกินค่าที่กำหนดไว้) และเลือกเพิ่มทรานสิชั่นไปยังสเตทที่ใกล้ที่สุด
4. ถ้าไม่สามารถเพิ่มทรานสิชั่นได้เลย จึงจะพิจารณาสร้างสเตทใหม่ โดยสเตทใหม่นี้ อาจจะเป็นคู่สเตทหรือไม่ก็ได้
5. เมื่อจบสตริง จะกำหนดให้สเตทปัจจุบันเป็นสเตทสุดท้าย (Final State) ซึ่งอาจจะมีหลายสเตทก็ได้
6. ย้อนกลับไปทำซ้ำข้อ 1 ใหม่อีกครั้ง สำหรับสตริงถัดไป ทำเช่นนี้ไปเรื่อยๆจนกว่าจะครบทุกสตริงที่มี

4.4.2 Negative Training

ในขั้นตอนนี้ จะนำสตริงของตัวอักษรอื่นๆมาทดสอบกับแมทชีนที่ได้สร้างขึ้นแล้วจากขั้นตอนแรก มาใช้ปรับแมทชีนให้ปฏิเสธ (Reject) สตริงของตัวอักษรอื่น โดยถ้าแมทชีนยอมรับ (Accept) สตริงใด ทั้งๆที่ไม่ควรจะยอมรับสตริงนั้น เพราะไม่ใช่สตริงของคลาสเป้าหมาย จะพิจารณาว่ามีอินพุตในช่วงใดที่ทำให้แมทชีนยอมรับสตริงนั้นได้ แล้วสร้างทรานสิชั่นไปยังสเตทใหม่ที่เป็นสเตทกับดัก (Trap State) เพื่อให้สตริงนั้นติดลูปที่สเตทกับดักจนไม่สามารถไปถึงสเตทสุดท้าย (Final State) ได้ ดังรูปที่ 4.16

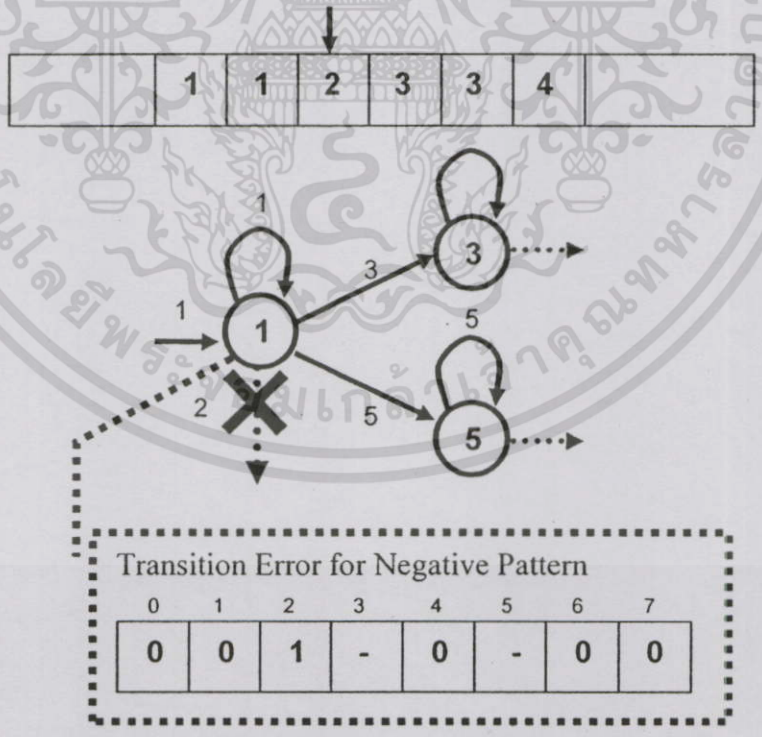


รูปที่ 4.16 แสดงการสร้างสเตทกับดักในแมทชีน

การจะสร้างทรานสิชั่นจากสเตทใดสเตทหนึ่งไปยังสเตทกับดักนั้น จะต้องเป็นทรานสิชั่นที่ไม่ส่งผลกระทบต่อสเตทเดิม กล่าวคืออินพุตใดที่มีทรานสิชั่นไปยังสเตทอื่นอยู่แล้ว จะไม่สามารถใช้อินพุตนั้นมาสร้างทรานสิชั่นไปยังสเตทกับดักได้อีก ทั้งนี้เพื่อให้แมทชีนเดิมไม่กระทบต่อสตริงที่ได้ยอมรับไปก่อนแล้ว

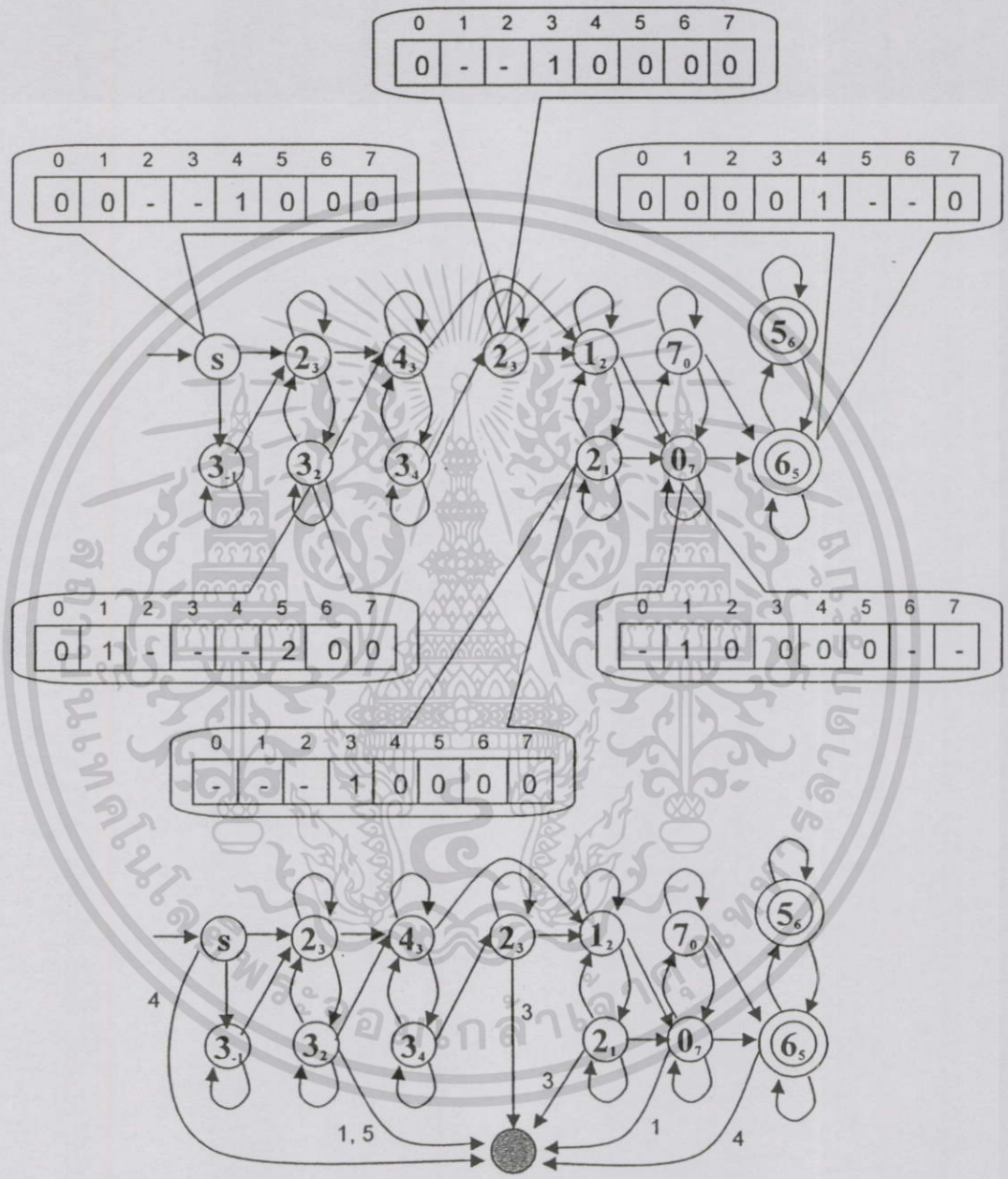
ดังนั้น การจะตัดสินใจสร้างทรานสิชั่นไปยังสเตทกับดัก จะทำได้โดยนำสตริงอื่นที่เป็นลบหรือสตริงของตัวอักษรอื่นๆ มาผ่านเข้าไปในแมทชีนที่สร้างไว้แล้ว ถ้าหากสตริงนั้นไม่ไปจบที่สเตทสุดท้าย (Final State) ของแมทชีน ก็แสดงว่า แมทชีนนั้นปฏิเสธสตริง จึงไม่จำเป็นต้องมีการปรับเปลี่ยนแมทชีนอะไรเพิ่มเติม แต่ถ้าหากสตริงไปจบที่สเตทสุดท้าย หรือแมทชีนสามารถยอมรับสตริงที่เป็นลบได้ ดังนั้น เราจึงต้องสร้างทรานสิชั่นเพิ่ม เพื่อดักสตริงไม่ให้ไปจนถึงสเตทสุดท้าย โดยการค้นหาอินพุตที่ทำให้เกิด Transition Error และนำมาใช้เชื่อมทรานสิชั่นจากสเตทใดสเตทหนึ่งไปยังสเตทกับดักต่อไป

Transition Error จะเกิดขึ้นก็ต่อเมื่อ ไม่มีทรานสิชั่นสำหรับอินพุตปัจจุบัน ณ สเตทที่อยู่ขณะนั้น เช่น สเตทมีทรานสิชั่นออกไปสำหรับอินพุตเท่ากับ 3 และ 5 แต่สตริงตัวถัดไปมีอินพุตเท่ากับ 2 ดังนั้น จึงเกิด Transition Error สำหรับสเตทนี้ที่อินพุตเท่ากับ 2 และจะบันทึกค่าเอาไว้ในตารางความผิดพลาดสำหรับสเตทนั้นๆ เพื่อใช้เป็นข้อมูลในการสร้างสเตทกับดัก รูปที่ 4.17 จะแสดงตัวอย่างการเกิด Transition Error และการบันทึกค่าความผิดพลาดลงในตาราง



รูปที่ 4.17 แสดงการเกิด Transition Error

รูปที่ 4.18 จะแสดงการเกิด Transition Error ของสตริงตัวอย่างทั้ง 2 และการสร้างทรานสิชั่นไปยังสเตทกับดัก จากแมทชีนในรูปที่ 4.15 เมื่อกำหนดให้มีสตริงที่เป็นลบ คือ 2232355432310646 และ 4323123541232016 ซึ่งจะได้เป็นแมทชีนสุดท้าย หลังจากผ่านขั้นตอน Negative Training แล้ว



รูปที่ 4.18 แสดงการสร้างทรานสิชั่นไปยังสเตทกับดักในช่วง Negative Training

สรุปขั้นตอนการเก็บข้อมูลของสตริงที่เป็นลบ และการสร้างสเตทกับดัก ในขั้นตอน Negative Training ได้ดังอัลกอริทึมในรูปที่ 4.19

```
//Negative Training
For each machine M of character class C
    Let C' be the class of any character EXCEPT class C

    For each pattern string P' in class C'
        Begin at the starting state
        Pass string P' into machine M
        If string P' can reach the final state then
            Pass string P' again and collect transition error in each state
        End If
    End For //for all patterns in C'

    Create trap state
    Add transition from each state that has error to the trap state
End For //for all character classes
```

รูปที่ 4.19 แสดงอัลกอริทึมของขั้นตอน Negative Training

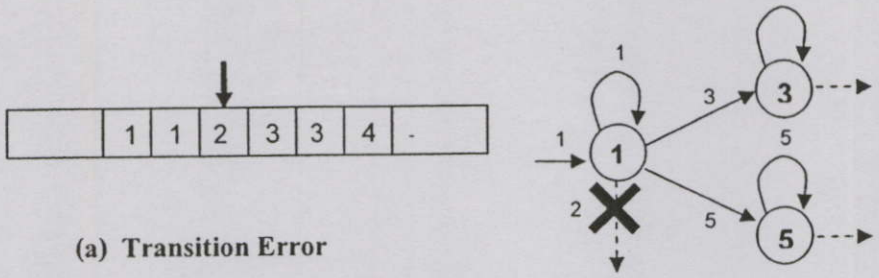
4.5 ขั้นตอนการรู้จำ

หลังจากขั้นตอนการสร้างแมทชีน (Training) สำหรับทุกๆตัวอักษรทั้งหมดแล้ว ก็จะนำสตริงที่ไม่ทราบค่ามาทดสอบกับแมทชีนทุกตัว เพื่อหาตัวอักษรที่ถูกต้อง โดยการทดสอบจะเริ่มไล่จากสเตทเริ่มต้น แล้วอ่านอินพุตเข้าไปทีละตัว ถ้ามีทรานสิชั่นสำหรับอินพุตตัวนั้น ก็จะย้ายไปสเตทใหม่ตามทรานสิชั่นที่มี ทำเช่นนี้ไปเรื่อย ๆ จนครบทั้งสตริง ถ้าสามารถไปจบที่สเตทสุดท้าย (Final State) ได้ แสดงว่า DFA ยอมรับสตริงนั้น จากนั้นนำไปทดสอบกับ DFA ของทุกคลาส

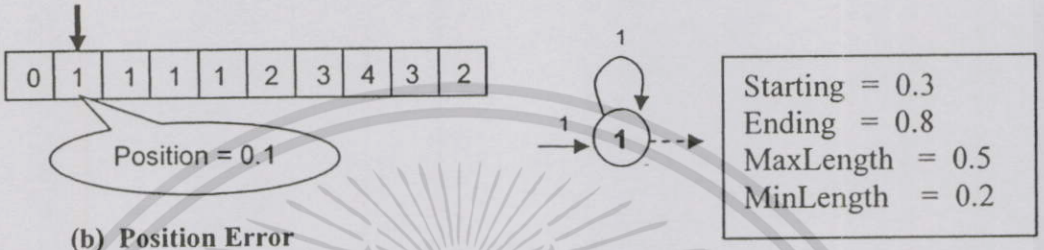
ในการทดสอบสตริงหนึ่ง ๆ นั้น อาจจะมี DFA หลายตัวที่สามารถยอมรับสตริงเดียวกันได้ ดังนั้น การเลือกว่าควรจะเป็น DFA ตัวใด จะใช้ข้อมูลความผิดพลาดที่เกิดขึ้นระหว่างการทดสอบมาประกอบการพิจารณา ความผิดพลาดมี 3 รูปแบบ ดังนี้

- *Transition Error* จำนวนอินพุตทั้งหมดในสเตทใด ๆซึ่งไม่มีทรานสิชั่นสำหรับอินพุตนั้น ๆ
- *Position Error* จำนวนอินพุตทั้งหมด ซึ่งตำแหน่งของอินพุตนั้น ไม่ได้อยู่ภายในช่วงตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายของแต่ละสเตท
- *Length Error* จำนวนอินพุตทั้งหมด ซึ่งมีความยาวเกินกว่าความยาวสูงสุด (MaxLength) หรือมีความยาวต่ำกว่าความยาวต่ำสุด (MinLength) ที่กำหนดในแต่ละสเตท

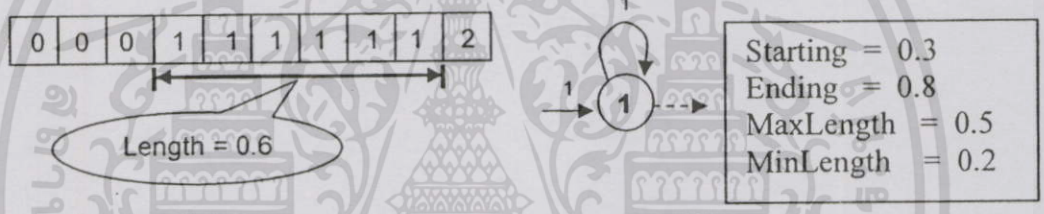
รูปที่ 4.20 จะแสดงการเกิดความผิดพลาดทั้ง 3 แบบ ซึ่ง Transition Error ก็คือ ลักษณะความผิดพลาดแบบเดียวกับที่ใช้เป็นข้อมูลในการสร้างสเตทกับดักในขั้นตอน Negative Training นั่นเอง



(a) Transition Error



(b) Position Error



(c) Length Error

รูปที่ 4.20 แสดงลักษณะความผิดพลาดที่เกิดขึ้นในขั้นตอนการรู้จำ

ถ้าไม่มีทรานสิชั่นสำหรับอินพุตตัวใด หรือเกิด Noise ขึ้นภายในสตริง ก็จะคงอยู่ที่สเตทเดิมไปจนกว่าจะมีทรานสิชั่นออกจากสเตทนั้น ตามอินพุตที่เข้ามาใหม่ โดยระหว่างนั้น จะนับเป็น Transition Error การเลือก DFA ที่เหมาะสม จะเลือกจาก DFA ที่ยอมรับสตริงนั้น และมีความผิดพลาดรวมน้อยที่สุด สรุปขั้นตอนการรู้จำได้ดังอัลกอริทึมในรูปที่ 4.21

```
//Competition Phase
Let P be the unknown pattern string
For each machine M of character class C
    Begin at the starting state
    //Pass string P into machine M and collect all the errors
    For each input x in string P
        If there has transition for input x to the next state then
            If position > ending OR position < starting then
                Increase position error
            End If
            If length > maxLength OR length < minLength then
                Increase length error
            End If
            Go to the next state
        Else
            Increase transition error
        End If
    End For //for all inputs in P
    Summarize all the errors and check if it reaches the final state
End For //for all character classes
//The winner is the machine that can be reached the final state and has minimum
errors
```

รูปที่ 4.21 แสดงอัลกอริทึมของขั้นตอนการรู้จำ

4.6 การสร้างแบบจำลองแบบ Multimodal

จากวิธีการที่ได้อธิบายไปข้างต้นนั้น จะเป็นการสร้างแมทชีนสำหรับแต่ละคลาส โดยหนึ่งคลาส หรือตัวอักษรแบบหนึ่งๆนั้น ก็จะมีหนึ่งแมทชีน แต่เมื่อพิจารณาสตริงตัวอย่างภายในคลาสแล้ว มีความเป็นไปได้ว่า ตัวอักษรหนึ่งๆนั้น สามารถมีลักษณะได้หลายรูปแบบ ซึ่งเมื่อทำการสร้างแมทชีนตัวแทนของคลาสขึ้นมา ก็จะต้องรวมสตริงที่มีลักษณะแตกต่างกันมากๆเข้าไว้ด้วยกัน ทำให้แมทชีนที่สร้างขึ้น อาจจะมีรูปแบบผิดปกติได้ จึงเกิดแนวความคิดที่จะแบ่งกลุ่มสตริงที่มีความคล้ายคลึงกันภายในแต่ละคลาส และสร้างแมทชีนสำหรับแต่ละกลุ่ม ทำให้หนึ่งคลาสของตัวอักษร มีแมทชีนได้หลายตัว หรือเรียกว่า Multimodal นั่นเอง

เนื่องจากสตริงภายในแต่ละคลาสนั้น อยู่รวมกันโดยที่เราไม่สามารถทราบได้ว่า สตริงใดมีลักษณะเหมือนหรือแตกต่างกับสตริงอื่น การจะจัดสตริงที่เหมือนกันให้อยู่ในกลุ่มเดียวกันได้นั้น จะทำการเปรียบเทียบโดยอาศัยการข้อมูลในการสร้างแมทชีนเป็นหลัก เช่น การนับจำนวนสเตทหรือทรานสิชั่นที่เพิ่มขึ้น หากเรารวมสตริงเข้าไปในกลุ่มที่มีอยู่แล้ว การแบ่งกลุ่มโดยใช้วิธีนี้ เป็น

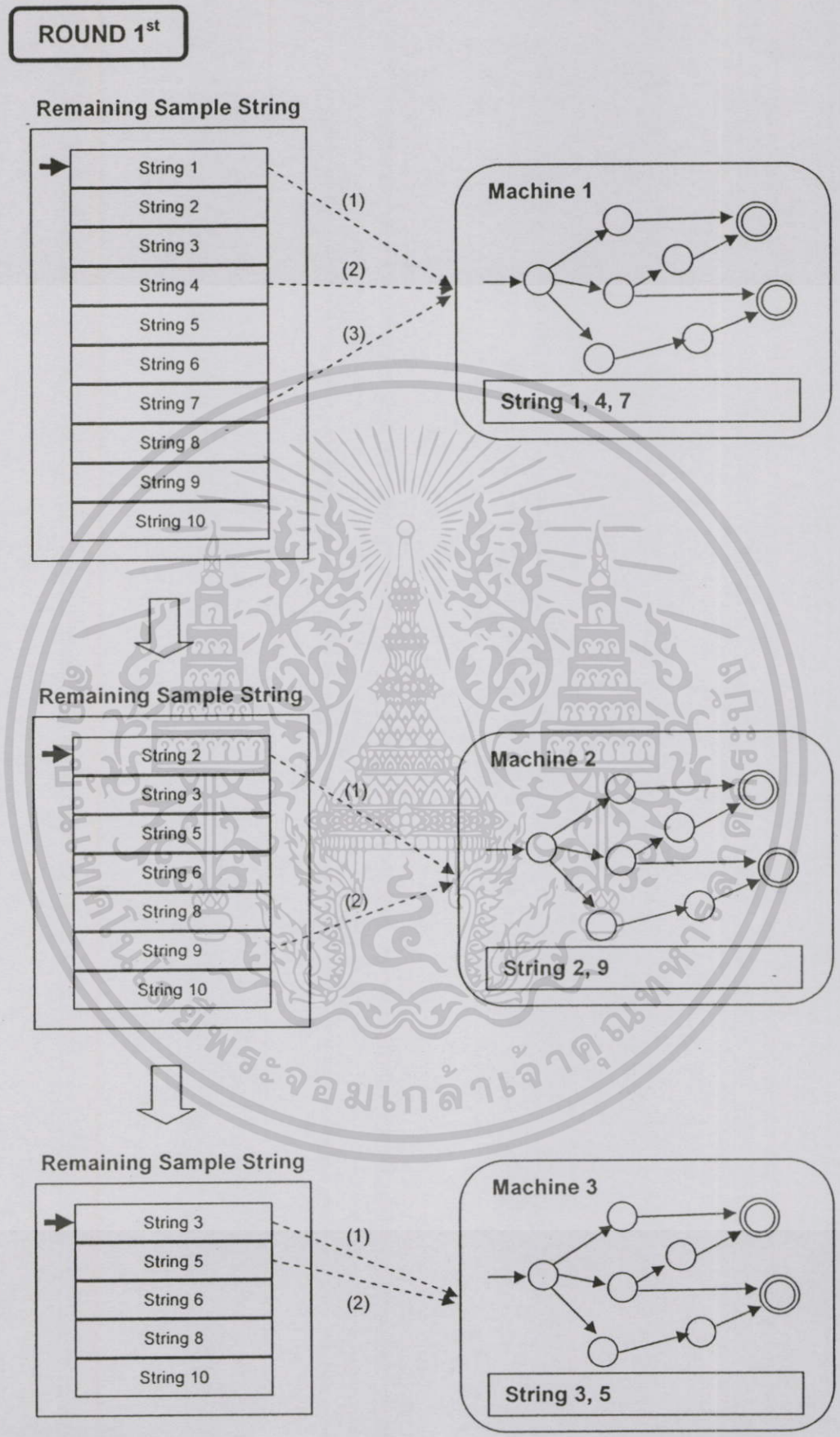
การจัดกลุ่มโดยที่ไม่มีการกำหนดไว้ก่อนว่าจะแบ่งอย่างไร จึงอาจถือเป็นการ Clustering ข้อมูลได้ด้วย

หลักการการแบ่งกลุ่มพร้อมๆกับสร้างแมทชีนนั้น ทำได้โดยนำสตริงตัวอย่างมาสร้างเป็นแมทชีนก่อนตามวิธีแบบเดียวกับที่อธิบายไปข้างต้น จากนั้น นำสตริงตัวต่อไปมาทดลองรวมเข้ากับแมทชีนเดิม ถ้าสตริงนั้นสามารถรวมเข้าไปในแมทชีนได้โดยที่ไม่มีการสร้างสเตทใหม่ หรือสร้างสเตทใหม่ในจำนวนและเงื่อนไขที่กำหนด ก็จะถือว่าเป็นสตริงในกลุ่มเดียวกัน เพราะสตริงใหม่ไม่ได้ทำให้โครงสร้างของแมทชีนเปลี่ยนไป จึงอนุมานว่าสตริงมีลักษณะคล้ายคลึงกัน

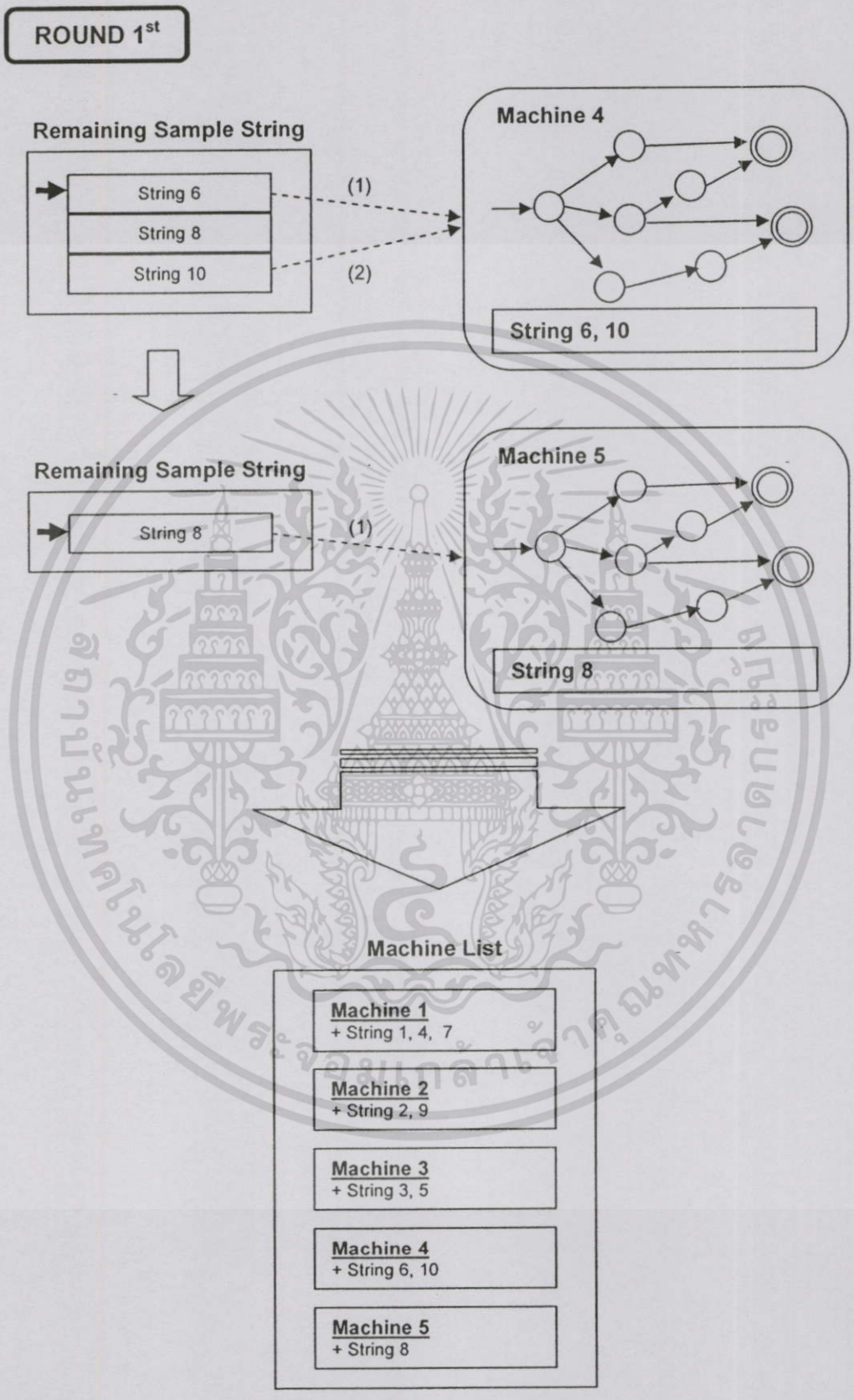
การรวมสตริงจะต้องทำซ้ำๆกัน จนกว่าจะมีจำนวนกลุ่ม (Cluster) ที่เหมาะสม โดยจะแบ่งแบบละเอียดก่อน นั่นคือ ตั้งเงื่อนไขการรวมให้แยกเป็นกลุ่มย่อยๆ แล้วค่อยรวมกลุ่มย่อยๆนั้นเข้ามารวมกันภายหลัง จะวนรวมกลุ่มจากสตริงแรกไปสตริงสุดท้าย และย้อนจากสตริงสุดท้ายกลับขึ้นไปสตริงแรกอีกครั้ง ในแต่ละรอบของการทำซ้ำ ก็จะมีการผ่อนปรนเงื่อนไขให้ลดลง เพื่อให้กลุ่มสามารถรวมกันได้มากขึ้น และไม่มีจำนวนกลุ่มมากเกินไป ซึ่งจะทำให้แมทชีนขาดลักษณะทั่วไป (Generalization) ได้ ขั้นตอนการสร้างโมเดลแบบ Multimodal แสดงในรูปที่ 4.22

```
//Multimodal
For each character class C
    Let MS be the list of machine in class C
    Create machine M from the first pattern using Positive Training
    Add M into machine list MS
    //from top to bottom
    For pattern string P_{i+1} to P_n in class C
        For each machine M in MS list
            Try to add string P into machine M
            If machine M is NOT changed then
                Merge P into the same group of M
            Else
                Create new machine M' from P and add M' into MS list
            End If
        End For //for all machine in MS
    End For //for all patterns in C except the previous patterns
    ... //repeat this step for all the conditions defined
End For //for all character classes
```

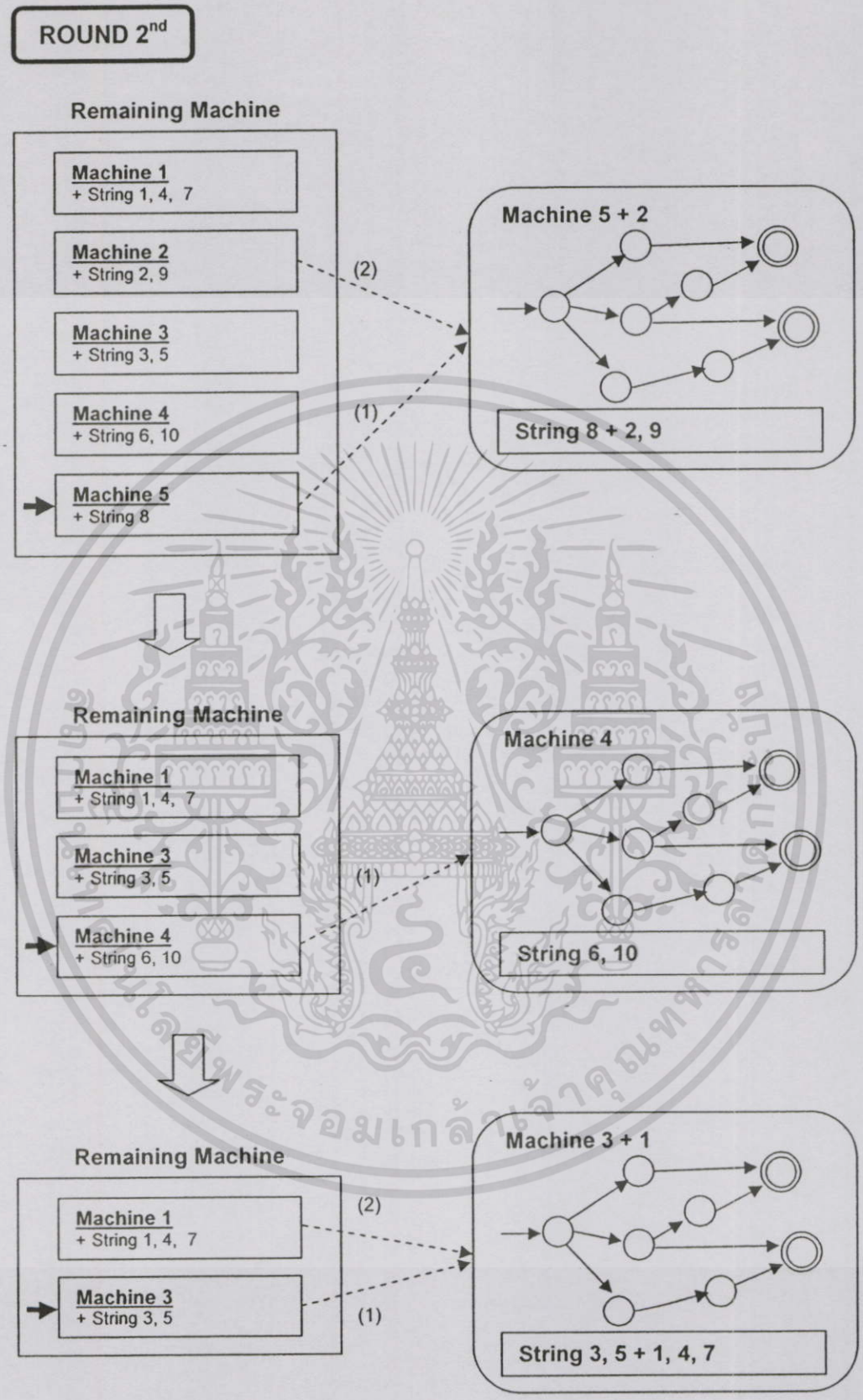
รูปที่ 4.22 แสดงอัลกอริทึมของวิธี Multimodal



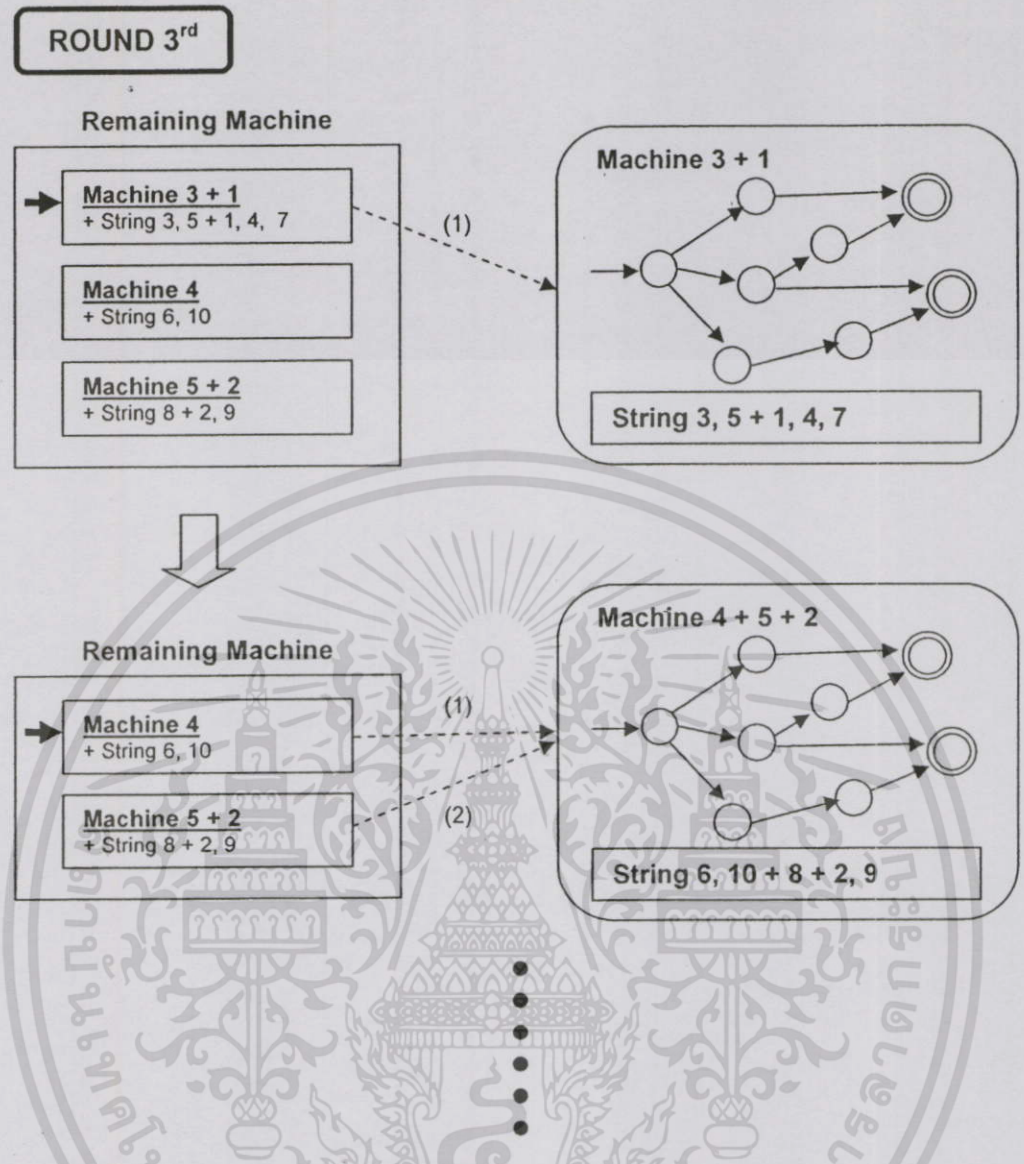
รูปที่ 4.23 แสดงตัวอย่างการสร้างแมทชีนจากวิธี Multimodal ในรอบแรก



รูปที่ 4.23 (ต่อ)



รูปที่ 4.24 แสดงตัวอย่างการยุบรวมแมทชีนจากวิธี Multimodal ในรอบที่ 2



รูปที่ 4.25 แสดงตัวอย่างการยุบรวมแมทชีนจากวิธี Multimodal ในรอบที่ 3

รูปที่ 4.23 – 4.25 จะแสดงตัวอย่างการสร้างแมทชีนในวิธี Multimodal ซึ่งจะเกิดจากการสร้างแมทชีนจากสตริงแต่ละกลุ่มย่อยๆก่อน จากนั้นจะยุบแมทชีนย่อยเข้าด้วยกัน ตามเงื่อนไขที่กำหนด การจะทราบว่าสตริงใด หรือแมทชีนใดอยู่กลุ่มเดียวกัน ทำได้โดยการนำสตริงนั้นๆ (ในกรณีที่เป็นรอบแรก) หรือสตริงที่ใช้สร้างแมทชีนนั้น (ในกรณีที่เป็นรอบถัดๆไป) มาทดสอบรวมกับแมทชีนตั้งต้น ถ้าสามารถรวมได้โดยแมทชีนนั้นไม่มีการเปลี่ยนแปลง (ในที่นี้ คือ การสร้างสเตทใหม่ ตามที่เงื่อนไขกำหนด) ก็แสดงว่า เป็นสตริงหรือแมทชีนในกลุ่มเดียวกัน สามารถรวมเข้ากันได้

ในรอบแรก จะเริ่มจากการสร้างแมทชีนจากสตริงตัวอย่าง ในรูปที่ 4.23 กำหนดให้มีสตริงตัวอย่างทั้งหมด 10 สตริง อยู่ภายในลิสต์ด้านซ้ายมือ ซึ่งก็คือ รายการสตริงที่เหลืออยู่ ซึ่งยังไม่ได้ถูก

นำไปสร้างเป็นแมทชีนใดๆ โดยจะเริ่มพิจารณาจากสตริงที่ 1 (ไล่จากสตริงแรกไปสตริงสุดท้าย) นำไปสร้างเป็นแมทชีนที่ 1 จากนั้น นำสตริงที่ 2 – 10 มาทดสอบรวมเข้ากับแมทชีนที่ 1 ตามลำดับ ว่าสามารถรวมได้หรือไม่ ผลปรากฏว่า สตริงที่ 4 และ 7 สามารถรวมเข้าไปในแมทชีนได้ โดยไม่มีการสร้างสเตทใหม่ จึงได้รวมเข้าเป็นแมทชีนที่ 1 ซึ่งสร้างจากสตริงที่ 1, 4 และ 7 ตามลำดับ รูปด้านขวามือ คือ แมทชีนที่ได้ ลูกศรเส้นประจะลากจากสตริงที่เป็นต้นกำเนิดมายังแมทชีน ตัวเลขบนเส้นประ คือ ลำดับการใช้สตริงมาสร้างแมทชีน ในรูป จะเริ่มจากสตริงที่ 1 เป็นลำดับแรก

ในขั้นต่อมา จะเหลือสตริงที่ยังไม่ได้นำมาสร้างเป็นแมทชีน คือ สตริง 2, 3, 5, 6, 8, 9, 10 เริ่มพิจารณาที่สตริงที่ 2 นำมาสร้างเป็นแมทชีนตั้งต้น แล้วนำสตริงอื่น คือ สตริง 3, 5, 6, 8, 9, 10 มารวมเข้ากับแมทชีนที่ 2 ตามลำดับ พบว่า สตริงที่ 9 สามารถรวมได้ จึงได้แมทชีนที่ 2 ซึ่งได้จากสตริงที่ 2 และ 9

ต่อมา เหลือสตริงที่ 3, 5, 6, 8, 10 เริ่มพิจารณาที่สตริงที่ 3 และนำสตริงอื่นมาทดสอบรวมเข้าในแมทชีนเช่นกัน จะได้แมทชีนที่ 3 ซึ่งได้จากสตริงที่ 3 และ 5 ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะหมดสตริงที่นำมาใช้ทดสอบ ซึ่งจะได้มาเป็นแมทชีนย่อยๆ ที่เป็นตัวแทนสตริงในแต่ละกลุ่ม ในรอบแรกนี้ จะได้มา 5 แมทชีน ดังรูป

ในรอบต่อๆมา จะเป็นการนำแมทชีนที่ได้จากรอบแรก มายุบรวมกันอีกครั้งตามเงื่อนไขที่กำหนด โดยรอบที่ 2 จะพิจารณายุบจากแมทชีนสุดท้าย ย้อนกลับมาแมทชีนแรก ในรูปที่ 4.24 ลิสต์ด้านซ้ายมือจะกลายเป็นแมทชีนที่ยังไม่ได้ประมวลผลในรอบนั้นๆ เริ่มพิจารณาจากแมทชีนสุดท้าย คือ แมทชีนที่ 5 แล้วนำแมทชีนที่ 4 – 1 มาทดสอบรวมเข้ากับแมทชีนที่ 5 ตามลำดับ (จากแมทชีนสุดท้าย ไปแมทชีนแรก) การทดสอบทำได้โดยนำสตริงที่ใช้สร้างแมทชีนนั้น มาทดสอบกับแมทชีนตั้งต้น ว่าสามารถรวมได้ทุกสตริง โดยที่ไม่สร้างสเตทใหม่หรือไม่ อย่างเช่น นำแมทชีนที่ 4 มาลองรวมเข้ากับแมทชีนที่ 5 ก็จะนำสตริงที่ 6, 10 มาทดสอบรวมกับแมทชีนที่ 5 เป็นต้น พบว่าแมทชีนที่ 2 สามารถรวมได้ จึงรวมได้เป็นแมทชีน 5 + 2 ซึ่งได้จากสตริงที่ 8, 2, 9

ต่อมา พิจารณาที่แมทชีนที่ 4 นำแมทชีนที่ 3 และ 1 มาทดสอบตามลำดับ พบว่า ไม่สามารถรวมกับแมทชีนใดได้เลย ก็จะคงแมทชีนไว้ในรูปแบบเดิม ในขั้นต่อมา แมทชีนที่ 3 เป็นแมทชีนตั้งต้น และสามารถรวมกับแมทชีนที่ 1 ได้ จึงได้แมทชีน 3 + 1 ซึ่งได้จากสตริงที่ 3, 5, 1, 4, 7 นั่นเอง

เมื่อจบการทำงานรอบที่ 2 แล้ว จะได้แมทชีนหลังจากยุบแล้ว ทั้งหมด 3 แมทชีน ก็จะนำมาพิจารณายุบแมทชีนในรอบที่ 3 ต่อไป ดังรูปที่ 4.25 หลักการยุบยังคงเป็นเหมือนกันรอบที่ 2 และจะพิจารณาจากแมทชีนแรกไปแมทชีนสุดท้ายแทน (สวนทิศทางกัน) เริ่มจากแมทชีน 3 + 1 มาตั้งต้น แล้วนำแมทชีนที่ 4 และแมทชีน 5 + 2 มายุบรวม ทำไปจนกว่าจะพิจารณาทุกแมทชีน จึงจะถือว่าครบรอบ ในรอบที่ 4, 5 หรือรอบถัดๆไป ก็จะทำซ้ำในลักษณะเดียวกันกับรอบที่ 2 และ 3 เพียงแต่จะเปลี่ยนเงื่อนไขการเกิดสเตทใหม่ไปเรื่อยๆ ตามแต่ที่ได้กำหนดไว้

การพิจารณาว่าสตริงที่เข้ามาใหม่นั้น สามารถรวมเข้ากับแมทชีนเดิมได้หรือไม่ หรือกล่าวอีกนัยหนึ่ง สามารถรวมกลุ่มเป็นสตริงกลุ่มเดียวกันได้หรือไม่ จะพิจารณาภายใต้เงื่อนไข ดังต่อไปนี้

1. รอบที่ 1 จะเป็นการแบ่งกลุ่มที่ละเอียดที่สุด โดยสตริงที่อยู่กลุ่มเดียวกันได้นั้น จะต้องไม่มีการสร้างสเตทใหม่เพิ่มขึ้นเลยแม้แต่สเตทเดียวไม่ว่ากรณีใดๆ (แต่สามารถเพิ่มทรานสิชั่นเข้าไปยังสเตทที่มีอยู่แล้วได้) โดยจะเริ่มจากสตริงแรกไล่ไปจนสตริงสุดท้าย
2. รอบที่ 2 และ 3 จะผ่อนปรนเงื่อนไขมากขึ้น โดยอนุญาตให้มีการสร้างคู่สเตทได้ ถ้าสเตทที่สร้างใหม่มีสถานะเป็นคู่สเตทของสเตทเดิม ก็จะไม่นับเป็นการสร้างสเตทใหม่ และสามารถรวมสตริงให้เป็นกลุ่มเดียวกันได้ โดยรอบที่ 2 จะไล่จากกลุ่มสุดท้ายขึ้นมากลุ่มแรก และรอบที่ 3 จะไล่กลับจากกลุ่มแรกไปยังกลุ่มสุดท้าย
3. รอบที่ 4 และ 5 ถ้าการสร้างสเตทใหม่เป็นสเตทที่มีทิศทางติดกับสเตทเดิม เช่น สเตทแทนทิศทาง 1 และ 2 แม้จะไม่ใช่คู่สเตทก็ตาม ก็จะไม่นับเป็นการสร้างสเตทใหม่ และอนุญาตให้รวมสตริงกันได้ โดยรอบที่ 4 จะไล่จากกลุ่มสุดท้ายขึ้นมากลุ่มแรก และรอบที่ 5 จะไล่กลับจากกลุ่มแรกไปยังกลุ่มสุดท้าย
4. รอบที่ 6 และ 7 การสร้างสเตทที่อยู่ต่อกับสเตทเริ่มต้น หรือสเตทที่แทนอินพุตตัวแรกของสตริง จะไม่นับเป็นการสร้างสเตทใหม่ โดยรอบที่ 6 จะไล่จากกลุ่มสุดท้ายขึ้นมากลุ่มแรก และรอบที่ 7 จะไล่กลับจากกลุ่มแรกไปยังกลุ่มสุดท้าย
5. รอบที่ 8 จะทำการเรียงลำดับกลุ่มสตริงจากกลุ่มที่มีขนาดใหญ่สุด หรือมีสตริงจำนวนมากที่สุด ไปยังกลุ่มที่มีขนาดเล็กสุด หรือมีสตริงจำนวนน้อยที่สุด จากนั้น จะทำการยุบรวมกลุ่มย่อยๆเข้าไปในกลุ่มที่มีขนาดใหญ่ โดยเลือกยุบกลุ่มที่มีการเพิ่มสเตทน้อยที่สุดจนถึงกลุ่มที่มีการเพิ่มจำนวนสเตทไม่เกินค่าที่กำหนดไว้ ทำซ้ำโดยให้กลุ่มใหญ่เป็นหลักเปรียบเทียบกับกลุ่มย่อย ถ้าเพิ่มสเตทไม่เกินจำนวนที่กำหนด ก็จะรวมกลุ่มที่เพิ่มสเตทน้อยไปมาก

การเปรียบเทียบสตริงสองกลุ่มว่าจะสามารถยุบรวมกันได้หรือไม่นั้น จะใช้วิธีเปรียบเทียบเฉพาะบางสตริงที่เป็นตัวแทนกลุ่มนั้นเท่านั้น สตริงที่เป็นตัวแทนกลุ่มจะเป็นสตริงที่ทำให้แมทชีนของกลุ่มนั้น มีลักษณะเปลี่ยนแปลงไป ตามการขั้นตอนการเปรียบเทียบในรอบที่ 2 ถึง 8 เงื่อนไขที่ผ่อนปรนไม่นับสเตทบางสเตท แต่ก็จะถือว่าเป็นสตริงที่ทำให้แมทชีนมีลักษณะเปลี่ยนแปลง และบันทึกไว้สำหรับการเปรียบเทียบกับกลุ่มอื่นๆ

สำหรับการรู้จำสตริงที่ไม่ทราบค่าในวิธีแบบ Multimodal นั้น ก็จะใช้หลักการเดียวกันกับการรู้จำปกติ และนำไปทดสอบกับทุกๆแมทชีนที่มี โดยแมทชีนที่ชนะก็คือ แมทชีนของกลุ่มสตริงใดก็ได้ที่เลเบลว่าเป็นตัวอักษรนั้นๆ การคิดคะแนนเพื่อหาผู้ชนะยังคงใช้หลักการเช่นเดิม

บทที่ 5

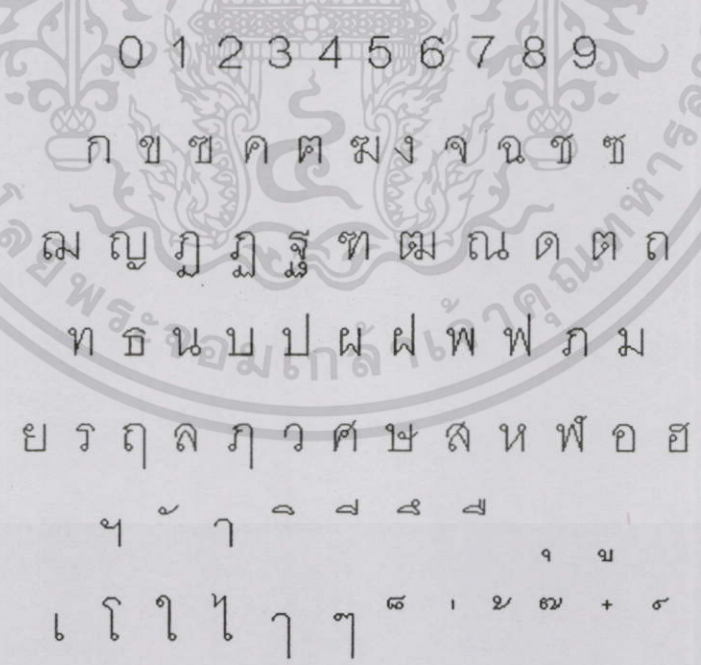
การทดลองและผลการทดลอง

วิทยานิพนธ์ฉบับนี้จะเสนอวิธีการสร้างโมเดลในรูปแบบสเตทแมทชีน หรือ DFA เพื่อการรู้จำสตริง ซึ่งในที่นี้ จะใช้สตริงที่ได้จากตัวอักษรภาษาไทยมาเป็นกรณีศึกษา โดยได้ทำการพัฒนามาจากวิธีการใช้ Context-free Grammar ในการรู้จำ [1]

การทดสอบขั้นตอนวิธีของงานวิจัยนี้ จะดำเนินการทดสอบเป็น 3 แบบ คือ แบบ Positive Training แบบ Negative Training และ แบบ Multimodal ตามที่ได้อธิบายไว้อย่างละเอียดในบทที่ 4 และจะนำผลลัพธ์ที่ได้มาเปรียบเทียบกับวิธี Hidden Markov Modal (HMM)

5.1 ข้อมูลที่ใช้ในการทดลอง

ข้อมูลตัวอย่างที่นำมาใช้ในการทดสอบการสร้างสเตทแมทชีน จะเป็นสตริงที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทยทั้งหมด 77 ตัวอักษร ทั้งพยัญชนะ สระ วรรณยุกต์ และตัวเลขอารบิก โดยสตริงทั้งหมดจะใช้วิธีการนำรูปภาพมาเข้ารหัสเป็นสตริงที่เรียกว่าเชนโค้ด (Chain Code) แบบ 8 ทิศทาง ตามที่ได้อธิบายไว้ในบทที่ 2 รูปที่ 5.1 จะแสดงตัวอย่างอักขระทั้งหมดที่ใช้เป็นอินพุต



รูปที่ 5.1 แสดงตัวอย่างอักขระทั้งหมดที่ใช้เป็นอินพุต

ตัวพิมพ์ตัวอักษรภาษาไทยนั้น จะใช้ฟอนต์ (Font) ตัวอย่างเป็นจำนวน 11 ฟอนต์ เพื่อความหลากหลายของสตริงตัวอย่าง ดังที่แสดงในตารางที่ 5.1 ประกอบด้วยฟอนต์ดังต่อไปนี้

- AngsanaUPC
- BrowalliaUPC
- CordiaUPC
- DilleniaUPC
- EucrosiaUPC
- FreesiaUPC
- IrisUPC
- JasmineUPC
- KodchiangUPC
- Microsoft Sans Serif
- Tahoma

ตารางที่ 5.1 แสดงตัวอย่างฟอนต์แบบต่างๆของอักขระ

	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
AngsanaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
BrowalliaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
CordiaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
DilleniaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
EucrosiaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
FreesiaUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
IrisUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
JasmineUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
KodchiangUPC	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
Microsoft Sans Serif	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ
Tahoma	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ

สตริงที่นำมาใช้ทั้งหมดมี 54277 สตริง จากตัวอักษร 77 ตัว คือมีคลาสทั้งหมด 77 คลาส โดยจะแบ่งสตริงในแต่ละคลาสออกเป็น 2 กลุ่ม กลุ่มหนึ่งเรียกว่า Training Set จะเป็นชุดตัวอย่างที่นำมาใช้สร้างแมทชีน ซึ่งจะประกอบด้วยสตริงคละฟอนต์คลาสละประมาณ 200 สตริง และอีกกลุ่มหนึ่งเรียกว่า Test Set จะเป็นชุดทดสอบความถูกต้องในการรู้จำ ซึ่งจะมีจำนวนสตริงแตกต่างกันไป ตารางที่ 5.2 จะแสดงรหัสแอสกีและชื่อเรียกอักขระแต่ละตัว พร้อมจำนวนสตริงตัวอย่าง

ตารางที่ 5.2 แสดงตัวอักขระทั้งหมดที่ใช้ในการทดลอง

ASCII		Name	#Train	#Test
048	0	ศูนย์	200	555
049	1	หนึ่ง	201	534
050	2	สอง	200	547
051	3	สาม	202	505
052	4	สี่	200	550
053	5	ห้า	205	502
054	6	หก	200	507
055	7	เจ็ด	200	514
056	8	แปด	200	501
057	9	เก้า	200	436
161	ก	ก - ไก่	200	568
162	ข	ข - ไข่	200	568
163	ฃ	ฃ - ขวด	200	568
164	ค	ค - ควาย	200	568
165	ฅ	ฅ - คน	200	568
166	ฆ	ฆ - ระฆัง	200	568
167	ง	ง - งู	201	567
168	จ	จ - จาน	200	568
169	ฉ	ฉ - ฉิ่ง	200	568
170	ช	ช - ช้าง	200	568
171	ซ	ซ - โซ่	200	568
172	ฌ	ฌ - เฌอ	200	568
173	ญ	ญ - หญิง	200	568
174	ฎ	ฎ - ชฎา	200	419
175	ฏ	ฏ - ปฏัก	200	460
176	ฐ	ฐ - ฐาน	200	568
177	ฑ	ฑ - มณโฑ	204	456
178	ฒ	ฒ - ผู้เฒ่า	200	568
179	ณ	ณ - เณร	200	568
180	ด	ด - เด็ก	203	565
181	ต	ต - เต่า	200	568
182	ถ	ถ - ถุง	200	568
183	ท	ท - ทหาร	203	565
184	ธ	ธ - ธง	200	568
185	น	น - หนู	202	566
186	บ	บ - ใบไม้	201	567
187	ป	ป - ปลา	200	419
188	ผ	ผ - ผึ้ง	200	568
189	ฝ	ฝ - ฝา	201	375

ASCII		Name	#Train	#Test
190	พ	พ - พาน	200	568
191	ฟ	ฟ - ฟัน	200	408
192	ภ	ภ - สำเภา	200	568
193	ม	ม - ม้า	200	568
194	ย	ย - ยักษ์	200	568
195	ร	ร - เรือ	202	566
196	ฤ	ตัวรึ	200	423
197	ล	ล - ลิง	200	567
198	ฦ	ตัวลึ	200	444
199	ว	ว - แหวน	200	568
200	ศ	ศ - ศาลา	201	513
201	ษ	ษ - ฤาษี	200	506
202	ส	ส - เสือ	200	568
203	ห	ห - หีบ	200	568
204	ฬ	ฬ - จุฬา	200	454
205	อ	อ - อ่าง	201	567
206	ฮ	ฮ - นกฮูก	202	566
207	ฯ	ไปยาลน้อย	200	206
209	ั	ไม้หันอากาศ	200	492
210	า	สระอา	200	452
212	ิ	สระอิ	200	460
213	ี	สระอี	200	514
214	ึ	สระอึ	200	501
215	ื	สระอือ	200	551
216	ุ	สระอุ	200	444
217	ู	สระอู	200	477
224	เ	สระเอ	200	390
226	โ	สระโอ	200	88
227	ใ	สระใอ	200	387
228	ไ	สระไอ	202	479
229	ๅ	สระประกอบตัวรือ/ตัวลือ	200	400
230	ๆ	ไม้ยมก	200	460
231	็	ไม้ไต่คู้	200	512
232	่	ไม้เอก	200	109
233	้	ไม้โท	200	507
234	๊	ไม้ตรี	200	511
235	๋	ไม้จัตวา	200	502
236	์	ตัวการันย์	203	509

หมายเหตุ : สำหรับอักษร “ญ – หญิง” และ “ฐ – ฐาน” นั้น จะพิจารณาตัวอักษรโดยตัดส่วนประกอบด้านล่างออกไป และใช้เฉพาะตัวอักษรที่มีเส้นเชื่อมต่อกันโดยตลอดเท่านั้น

5.2 ผลการทดลอง

การทดสอบขั้นตอนวิธี จะทำการทดสอบการสร้างโมเดล 3 แบบ คือ แบบ Positive Training แบบ Negative Training และ แบบ Multimodal

5.2.1 วิธีแบบ Positive Training

การทดสอบขั้นตอนวิธีในแบบ Positive Training จะทำการสร้างแมทชีนจากสตริงตัวอย่างที่เป็นบวกเท่านั้น ซึ่งจะได้แมทชีนเป็นจำนวน 77 แมทชีน สำหรับตัวอักษร 77 คลาส โดยจะมีจำนวนสเตทโดยเฉลี่ยเท่ากับ 95.36 สเตทต่อแมทชีน

5.2.2 วิธีแบบ Positive และ Negative Training

การทดสอบขั้นตอนวิธีในแบบ Positive Training และ Negative Training จะทำการสร้างแมทชีนจากสตริงตัวอย่างที่เป็นบวกและลบ โดยการสร้างแมทชีนจากสตริงที่เป็นบวกขึ้นมาก่อนแล้วเพิ่มสเตทกับดักเข้าไปในแมทชีนโดยใช้ข้อมูลจากสตริงที่เป็นลบ ซึ่งจะได้แมทชีนเป็นจำนวน 77 แมทชีน สำหรับตัวอักษร 77 คลาส เท่ากับวิธีแบบ Positive Training แต่จะมีจำนวนสเตทโดยเฉลี่ยเท่ากับ 96.36 สเตทต่อแมทชีน ซึ่งจะมากกว่าวิธีแบบ Positive Training อยู่ 1 สเตท หรือก็คือสเตทกับดักนั่นเอง

5.2.2 วิธีแบบ Multimodal

การทดสอบขั้นตอนวิธีในแบบ Multimodal จะทำการสร้างแมทชีนจากสตริงตัวอย่างที่เป็นบวกเท่านั้น (เฉพาะส่วนที่เป็น Positive Training) และจะได้แมทชีนในแต่ละคลาสเป็นจำนวนแตกต่างกัน การรู้จำจะทำได้โดยการนำสตริงที่ไม่ทราบค่ามาทดสอบกับ DFA ทุกๆตัวของแต่ละคลาส และเลือกผู้ชนะที่มีค่าความผิดพลาดรวมน้อยที่สุด

วิธีทั้ง 3 ที่ได้นำเสนอในวิทยานิพนธ์ฉบับนี้ จะนำมาเปรียบเทียบกับวิธี Hidden Markov Model (HMM) ซึ่งผลการรู้จำแยกตามตัวอักษรจะแสดงในตารางที่ 5.3 เปรียบเทียบระหว่างวิธีทั้ง 3 และ HMM เมื่อทดสอบกับ Test Set โดยวิธีที่ได้นำเสนอ จะแบ่งผลการรู้จำได้เป็น 2 ประเภท คือ สตริงที่สามารถรู้จำได้ถูกต้อง โดยผู้ชนะคือ DFA ที่ถูกต้อง และมีเพียง DFA เดียวที่เป็นผู้ชนะ (Recognition Rate) และสตริงที่มีผู้ชนะที่มีคะแนนรวมเท่านั้นมากกว่า 2 DFA ขึ้นไป และหนึ่งใน DFA ที่เป็นผู้ชนะเหล่านั้น มี DFA ที่ถูกต้องอยู่ด้วย (Accept more the one DFA)

ตารางที่ 5.4 จะแสดงจำนวนสตริงที่ไปตกตาม DFA ของคลาสต่างๆ ซึ่งจะขอแสดงเฉพาะวิธี Multimodel เท่านั้น ซึ่งได้ผลการรู้จำที่ดีที่สุด ตารางนี้จะช่วยแสดงให้เห็นว่า ตัวอักษรหลายๆตัวที่ผลการรู้จำไม่ดีนัก หรือสามารถยอมรับ DFA ได้มากกว่า 1 ตัวนั้น จะมีความคล้ายคลึงกับอักษรตัวใดได้บ้าง เปรียบเทียบกับผลการรู้จำโดยละเอียดของ HMM ในตารางที่ 5.5

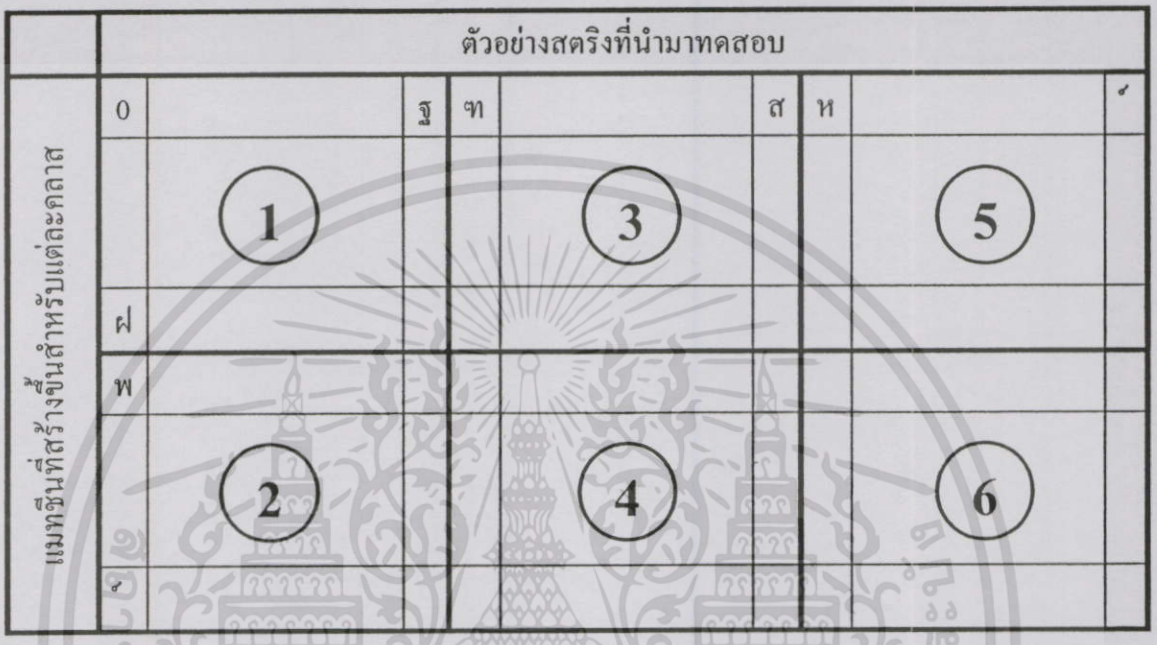
ตารางที่ 5.3 แสดงผลการรู้จำของ Test Set แยกตามตัวอักษรเปรียบเทียบกับวิธีต่างๆ

ASCII	Symbol	Positive			Positive & Negative			Multimodal			HMM
		Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate
48	0	**29.73**	62.70	7.57	**29.55**	62.52	7.93	**92.43**	2.88	4.68	**36.94**
49	1	**86.33**	8.99	4.68	**86.89**	8.99	4.12	**93.07**	0.00	6.93	**66.73**
50	2	**94.52**	0.00	5.48	**94.88**	0.00	5.12	**95.06**	0.00	4.94	**99.45**
51	3	**95.05**	0.00	4.95	**96.44**	0.00	3.56	**95.45**	0.20	4.36	**100.00**
52	4	**91.45**	0.00	8.55	**90.36**	0.18	9.45	**93.45**	0.00	6.55	**94.36**
53	5	**86.26**	0.80	12.95	**87.45**	1.00	11.55	**89.24**	0.20	10.56	**89.04**
54	6	**91.32**	1.97	6.71	**90.73**	1.97	7.30	**95.27**	0.00	4.73	**76.13**
55	7	**92.61**	0.39	7.00	**90.27**	0.39	9.34	**95.53**	0.00	4.47	**95.14**
56	8	**74.05**	16.77	9.18	**74.05**	16.37	9.58	**96.21**	0.80	2.99	**91.02**
57	9	**93.35**	0.00	6.65	**93.58**	0.23	6.19	**93.58**	0.00	6.42	**89.91**
161	ก	**80.99**	8.27	10.74	**80.28**	7.92	11.80	**94.37**	1.58	4.05	**25.70**
162	ข	**15.32**	66.20	18.49	**15.85**	64.96	19.19	**79.40**	6.16	14.44	**63.73**
163	ฃ	**42.43**	31.87	25.70	**38.91**	31.51	29.58	**69.19**	2.11	28.70	**75.53**
164	ค	**69.01**	16.73	14.26	**67.96**	16.55	15.49	**81.51**	5.63	12.85	**8.45**
165	ฅ	**78.17**	12.32	9.51	**78.35**	12.32	9.33	**89.44**	0.70	9.86	**43.66**
166	ฆ	**73.59**	1.06	25.35	**77.99**	0.88	21.13	**76.41**	0.00	23.59	**83.45**
167	ง	**86.60**	1.59	11.82	**85.71**	1.59	12.70	**92.59**	0.35	7.05	**87.13**
168	จ	**85.21**	3.35	11.44	**85.04**	3.52	11.44	**86.09**	1.58	12.32	**86.27**
169	ฉ	**85.04**	0.53	14.44	**86.80**	0.53	12.68	**80.28**	0.53	19.19	**54.23**
170	ช	**47.71**	28.35	23.94	**48.42**	27.99	23.59	**53.17**	2.99	43.84	**19.01**
171	ซ	**39.61**	23.94	36.44	**40.85**	23.42	35.74	**63.56**	0.35	36.09	**74.30**
172	ฌ	**77.64**	0.88	21.48	**79.23**	0.88	19.89	**84.15**	0.00	15.85	**81.51**
173	ญ	**74.65**	11.09	14.26	**75.18**	10.92	13.91	**87.85**	0.35	11.80	**97.18**
174	ฎ	**51.55**	23.87	24.58	**53.94**	24.11	21.96	**79.47**	1.43	19.09	**72.55**
175	ฏ	**67.39**	2.39	30.22	**66.52**	2.61	30.87	**59.35**	0.65	40.00	**35.22**
176	ฐ	**85.39**	3.52	11.09	**84.86**	3.52	11.62	**83.80**	1.76	14.44	**64.02**
177	ฑ	**94.30**	0.22	5.48	**95.39**	0.22	4.39	**94.52**	0.00	5.48	**100.00**
178	ฒ	**85.74**	0.00	14.26	**92.78**	0.18	7.04	**89.96**	0.35	9.68	**56.51**
179	ณ	**82.39**	3.87	13.73	**82.04**	3.35	14.61	**85.39**	0.35	14.26	**90.49**
180	ด	**92.39**	1.42	6.19	**93.98**	1.42	4.60	**66.55**	13.81	19.65	**100.00**
181	ต	**77.46**	8.27	14.26	**79.05**	7.57	13.38	**86.27**	3.17	10.56	**21.83**
182	ถ	**92.43**	0.00	7.57	**91.73**	0.00	8.27	**85.39**	0.18	14.44	**99.12**
183	ท	**87.43**	5.31	7.26	**86.37**	4.78	8.85	**93.81**	0.88	5.31	**25.66**
184	ธ	**80.46**	5.99	13.56	**80.11**	5.99	13.91	**87.68**	0.88	11.44	**77.99**
185	น	**92.76**	1.59	5.65	**92.93**	1.41	5.65	**89.58**	0.53	9.89	**97.70**
186	บ	**40.74**	44.62	14.64	**40.56**	44.27	15.17	**86.42**	3.35	10.23	**93.12**
187	ป	**89.50**	0.72	9.79	**88.78**	0.72	10.50	**92.12**	0.24	7.64	**7.88**
188	ผ	**93.31**	0.18	6.51	**93.49**	0.00	6.51	**90.67**	0.00	9.33	**85.21**
189	ฝ	**89.87**	0.27	9.87	**91.47**	0.00	8.53	**92.00**	0.00	8.00	**33.60**

ตารางที่ 5.3 (ต่อ)

ASCII	Symbol	Positive			Positive & Negative			Multimodal			HMM
		Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate
190	พ	**41.37**	41.37	17.25	**41.90**	41.37	16.73	**90.14**	0.35	9.51	**67.43**
191	ฟ	**82.60**	3.19	14.22	**80.88**	3.19	15.93	**80.39**	0.49	19.12	**55.88**
192	ภ	**91.73**	2.64	5.63	**91.55**	2.64	5.81	**94.89**	1.94	3.17	**94.37**
193	ม	**48.06**	36.44	15.49	**50.18**	36.27	13.56	**77.29**	11.62	11.09	**84.15**
194	ย	**92.61**	0.00	7.39	**94.89**	0.00	5.11	**94.37**	0.00	5.63	**99.82**
195	ร	**77.74**	15.90	6.36	**77.39**	15.90	6.71	**73.32**	1.24	25.44	**87.28**
196	ฤ	**92.91**	0.00	7.09	**93.14**	0.00	6.86	**90.54**	0.00	9.46	**42.32**
197	ล	**65.08**	20.99	13.93	**65.43**	20.81	13.76	**92.95**	0.00	7.05	**92.06**
198	ภ	**90.32**	2.70	6.98	**88.96**	2.70	8.33	**96.40**	0.23	3.38	**95.50**
199	ว	**90.67**	4.75	4.58	**90.49**	4.75	4.75	**96.83**	0.00	3.17	**89.96**
200	ศ	**94.35**	0.00	5.65	**95.13**	0.00	4.87	**85.77**	0.78	13.45	**91.42**
201	ษ	**88.14**	0.40	11.46	**90.12**	0.20	9.68	**82.61**	0.20	17.19	**97.04**
202	ส	**88.91**	0.00	11.09	**94.54**	0.00	5.46	**79.75**	0.18	20.07	**96.30**
203	ห	**94.01**	0.00	5.99	**92.78**	0.00	7.22	**91.02**	0.18	8.80	**99.82**
204	ฬ	**88.77**	0.22	11.01	**89.21**	0.00	10.79	**92.73**	0.00	7.27	**98.68**
205	อ	**75.31**	11.29	13.40	**75.31**	11.46	13.23	**88.54**	0.18	11.29	**88.36**
206	ฮ	**70.14**	15.19	14.66	**71.20**	15.19	13.60	**87.10**	2.12	10.78	**33.16**
207	ฯ	**82.52**	4.37	13.11	**82.04**	4.37	13.59	**78.64**	2.91	18.45	**81.55**
209	ั	**25.81**	64.02	10.16	**25.81**	63.41	10.77	**59.35**	35.37	5.28	**60.37**
210	า	**79.20**	17.04	3.76	**78.54**	16.59	4.87	**82.30**	9.73	7.96	**86.28**
212	ิ	**60.43**	33.48	6.09	**60.43**	33.48	6.09	**72.61**	19.35	8.04	**86.74**
213	ี	**25.68**	66.54	7.78	**25.88**	66.34	7.78	**66.15**	21.21	12.65	**56.03**
214	ึ	**20.16**	69.66	10.18	**20.36**	69.46	10.18	**61.48**	25.75	12.77	**63.87**
215	ื	**74.77**	16.15	9.07	**75.68**	15.97	8.35	**70.96**	6.35	22.69	**73.68**
216	ุ	**61.71**	27.48	10.81	**61.49**	27.48	11.04	**64.64**	24.32	11.04	**64.19**
217	ู	**33.54**	50.52	15.93	**33.96**	50.10	15.93	**57.65**	20.96	21.38	**57.23**
224	เ	**87.18**	9.49	3.33	**88.46**	8.97	2.56	**85.13**	8.97	5.90	**45.38**
226	โ	**92.05**	0.00	7.95	**92.05**	0.00	7.95	**92.05**	0.00	7.95	**88.64**
227	ใ	**91.99**	3.88	4.13	**92.76**	3.88	3.36	**82.69**	0.78	16.54	**56.07**
228	ไ	**97.29**	0.00	2.71	**95.62**	0.00	4.38	**97.29**	0.42	2.30	**96.24**
229	ๅ	**49.50**	44.75	5.75	**49.50**	44.50	6.00	**73.25**	19.00	7.75	**20.50**
230	ๆ	**81.09**	2.61	16.30	**80.00**	2.17	17.83	**87.39**	0.87	11.74	**26.09**
231	็	**77.15**	8.20	14.65	**77.54**	8.01	14.45	**76.56**	1.76	21.68	**50.98**
232	่	**8.26**	78.90	12.84	**7.34**	78.90	13.76	**27.52**	66.06	6.42	**66.06**
233	้	**74.36**	19.92	5.72	**74.36**	19.72	5.92	**78.50**	8.09	13.41	**86.59**
234	๊	**84.74**	1.17	14.09	**85.91**	1.17	12.92	**94.32**	0.20	5.48	**98.04**
235	๋	**87.45**	2.99	9.56	**87.05**	3.19	9.76	**83.67**	0.20	16.14	**90.24**
236	์	**77.01**	15.72	7.27	**77.60**	14.54	7.86	**96.46**	0.00	3.54	**89.45**

เนื่องจากตารางที่ 5.4 และ 5.5 มีขนาดใหญ่มาก และไม่สามารถบรรจุภายในหน้าเดียวได้ จึงต้องแบ่งตารางออกเป็น 6 ส่วน รูปที่ 5.2 จะแสดงรูปแบบของตารางอย่างกว้างๆ เพื่อให้เข้าใจได้ง่ายขึ้น โดยในแนวตั้งของตาราง จะเป็นแมทชีนที่สร้างขึ้นสำหรับตัวอักษรแต่ละตัว และแนวนอนจะเป็นตัวอย่างสตริงที่นำมาใช้ทดสอบ



รูปที่ 5.2 แสดงรูปแบบของตารางที่ 5.4 และ 5.5

สตริงตัวอย่างที่สามารถรู้จำได้ถูกต้อง ก็จะต้องไปตกที่แมทชีนของคลาสนั้นๆ ดังนั้น จึงต้องพิจารณาที่จำนวนสตริงที่ถูกต้องตามแนวทแยงมุม จากมุมซ้ายบน ลงมาที่มุมขวาล่าง ซึ่งตารางทั้งสองนี้ ทำให้ทราบได้ว่า สตริงที่เกิดการรู้จำผิดพลาดนั้นไปตกที่แมทชีนตัวใดบ้าง เป็นจำนวนเท่าใด ซึ่งทำให้สามารถวิเคราะห์ปัญหาที่เกิดขึ้นในการรู้จำได้

ตารางที่ 5.4 แสดงผลการรู้จำของ Test Set โดยละเอียดแยกตามตัวอักษรของวิธี Multimodal

จำนวนสตริง		555	534	547	505	550	502	507	514	501	436	568	568	568	568	568	568	567	568	568	568	568	568	568	419	460	568
ASCII		48	49	50	51	52	53	54	55	56	57	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
		0	1	2	3	4	5	6	7	8	9	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ	ฐ
48	0	529	0	0	0	0	0	3	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	3	0
49	1	0	497	0	0	0	0	0	2	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	2	0
50	2	0	0	520	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
51	3	0	0	0	483	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	4	0	0	0	0	514	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
53	5	0	0	3	0	0	449	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
54	6	0	0	0	0	0	0	483	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	7	0	9	9	4	0	0	0	491	0	3	0	0	4	0	0	0	0	0	1	15	17	0	0	0	1	1
56	8	0	0	0	2	0	1	0	0	486	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	1
57	9	0	0	2	6	0	10	0	0	0	408	0	0	0	0	0	1	5	0	0	3	4	0	0	0	2	0
161	ก	0	0	0	0	0	0	0	0	0	0	545	0	0	2	6	0	0	0	0	0	0	0	0	0	0	0
162	ข	0	0	0	0	0	0	0	0	0	0	0	486	93	0	0	49	0	0	0	61	31	0	0	0	0	0
163	ฃ	0	0	0	0	0	0	0	0	0	2	0	83	405	0	0	42	0	0	0	33	67	0	0	0	0	2
164	ค	0	0	0	0	0	0	0	0	0	0	0	0	0	495	20	0	0	0	0	0	1	0	0	0	0	0
165	ฅ	0	0	0	0	0	0	0	0	0	0	1	0	0	74	512	0	0	0	0	0	0	0	0	0	0	0
166	ฆ	0	0	0	0	0	0	0	0	0	2	0	6	8	0	0	434	0	0	0	2	6	2	0	0	0	0
167	ง	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	527	7	2	0	0	0	0	0	0	4
168	จ	7	0	0	3	0	0	1	0	1	0	0	1	1	0	0	0	6	498	34	0	0	0	0	4	8	17
169	ฉ	1	0	0	0	0	0	0	0	2	0	1	0	0	1	2	0	4	35	459	0	0	0	0	19	7	0
170	ช	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	319	19	0	0	0	0	1
171	ซ	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	106	363	0	0	0	0	0
172	ฌ	0	0	0	0	1	0	0	0	0	0	0	0	2	0	0	6	0	0	0	0	0	478	22	0	0	0
173	ญ	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	15	501	0	0	0
174	ฎ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	339	100	0
175	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	276	0
176	ฐ	0	0	9	1	2	9	14	1	11	1	3	1	3	0	0	0	3	0	0	16	3	0	0	2	5	486
177	ฑ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	7	0	0	0	0	0	10	0	0	0	0
179	ณ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	9	0	0	0
180	ด	0	0	0	0	0	0	0	0	0	0	0	0	0	22	1	0	0	0	0	0	0	0	0	0	0	0
181	ต	0	0	0	0	0	0	0	0	0	0	0	0	0	5	13	0	0	0	0	0	1	1	0	0	0	2
182	ถ	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
183	ท	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0
184	ธ	0	0	1	0	0	5	0	0	0	1	0	3	2	0	0	1	0	0	1	1	0	0	0	0	0	17
185	น	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	5	0	0	0	0
186	บ	0	0	0	0	0	0	0	0	0	0	0	3	3	0	0	2	0	0	0	0	1	0	0	0	0	0
187	ป	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.4 (ต่อ)

จำนวนสตริง		555	534	547	505	550	502	507	514	501	436	568	568	568	568	568	568	567	568	568	568	568	568	568	419	460	568
ASCII		48	49	50	51	52	53	54	55	56	57	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
		0	1	2	3	4	5	6	7	8	9	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ	ฐ
190	พ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
193	ม	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	6	0	0	0	0
194	ย	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
195	ร	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0
197	ล	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
198	ฦ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	32	0
199	ว	0	0	0	5	0	2	0	0	0	5	0	0	0	0	0	0	0	1	10	0	2	0	0	9	17	0
200	ศ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0
201	ษ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
202	ส	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
203	ห	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	2	1	0	0	0
204	ฬ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2
205	อ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
206	ฮ	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	8	4	0	0	0	0	8
207	ฯ	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	10	10	0	0	0	0	0
209		0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12
210	า	0	4	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	4	8	0	1	0	0	0	0	0
212	◌ั	0	0	0	0	2	2	0	1	0	0	1	0	0	1	2	0	0	1	0	0	5	0	0	0	0	3
213	◌ิ	0	0	1	0	1	8	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	2
214	◌ี	0	0	0	0	12	0	0	10	2	4	0	0	1	1	2	0	6	0	0	0	0	0	0	3	0	1
215	◌ึ	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	2	0	0	0	0	0	2	3	0	1	1
216	◌ื	25	5	0	1	7	0	0	0	0	5	0	1	15	0	0	5	2	19	31	1	2	0	0	3	7	3
217	◌ุ	1	1	0	0	0	0	0	0	0	0	0	14	12	0	0	4	1	6	2	8	3	2	1	0	0	2
224	เ	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
226	โ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
227	ใ	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
228	ไ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
229	ๅ	0	10	0	2	0	0	0	1	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0
230	ๆ	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	1	1	3	4	0	0	1	0	0
231	◌็	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
232	◌่	9	4	0	0	2	0	1	0	0	2	0	1	6	0	0	0	6	2	16	0	1	0	0	0	0	0
233	◌้	0	0	0	0	6	1	0	0	0	0	0	3	0	0	0	1	2	0	0	1	0	8	1	0	0	2
234	◌๊	0	0	0	0	1	2	0	0	0	0	0	0	9	0	1	6	0	0	0	0	24	3	29	0	0	0
235	◌๋	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	2	0
236	◌์	3	0	0	0	0	0	2	3	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	3

ตารางที่ 5.4 (ต่อ)

จำนวนสตริง		456	568	568	565	568	568	565	568	566	567	419	568	375	568	408	568	568	568	566	423	567	444	568	513	506	568
ASCII		177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202
		ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ	ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	0
50	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	1
51	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0
54	6	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
55	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	1	0	0	2
56	8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
161	ก	1	0	0	1	8	72	2	0	0	0	0	0	0	0	0	28	0	0	0	3	0	0	0	11	0	0
162	ข	1	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	6	0	0	0	0	0	0	0	3	0
163	ฃ	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	3	0	0	0	0	0	0	8	1
164	ค	0	0	0	14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0
165	ฅ	0	0	0	12	50	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	1	0
166	ฆ	0	0	0	0	0	0	0	0	0	6	1	0	0	0	2	0	112	1	0	0	0	0	0	0	2	0
167	ง	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0
168	จ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
169	ฉ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
170	ช	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1
171	ซ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
172	ฌ	0	19	21	0	0	0	4	0	0	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0
173	ญ	0	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
174	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
175	ฎ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	ฐ	1	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	1	121	0	4	0	0	0	0	5
177	ฑ	431	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	513	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
179	ณ	0	0	487	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
180	ด	0	2	0	454	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0
181	ต	2	2	0	164	508	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	1	5	0	0
182	ถ	0	0	0	0	0	486	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	1	0	0
183	ท	4	4	0	0	3	0	535	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
184	ธ	1	0	0	0	0	0	0	503	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	1	1
185	น	0	1	0	0	0	0	0	0	510	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
186	บ	0	0	0	0	0	0	0	0	16	509	23	9	1	0	1	0	4	13	0	0	0	0	0	0	8	0
187	ป	0	0	0	0	0	0	0	0	0	1	387	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	0	0	0	0	0	0	0	515	11	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	15	345	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.4 (ต่อ)

จำนวนสตริง		456	568	568	565	568	568	565	568	566	567	419	568	375	568	408	568	568	568	566	423	567	444	568	513	506	568
ASCII		177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202
		ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ	ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส
190	พ	0	0	1	0	0	0	0	0	0	0	0	7	0	514	21	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	0	0	0	0	0	0	0	0	0	0	0	0	1	330	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	550	0	0	0	2	0	12	0	0	0	0
193	ม	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	505	1	0	0	0	0	0	0	0	0
194	ย	0	0	0	0	0	0	0	0	0	0	0	2	1	0	1	0	0	536	0	0	0	0	0	0	0	0
195	ร	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	422	0	0	0	0	0	0	0
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	383	0	2	0	0	0	0
197	ล	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	527	0	0	0	0	50
198	ฦ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	429	0	0	0	0
199	ว	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	550	0	0	1
200	ศ	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	444	0	3
201	ษ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	419	0
202	ส	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	454
203	ห	3	0	6	0	0	0	13	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
204	ฬ	0	1	3	0	0	0	0	0	0	0	0	8	8	47	51	0	0	0	0	0	0	0	0	0	0	16
205	อ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
206	ฮ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
207	ฯ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
209		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
210	ะ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	2	11	0	0	0
212	ั	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
213	ิ	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	11	0	0	2	0	6
214	ี	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	16	0	0	0	0	13
215	ึ	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
216	ื	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	1	0	0	2	0	0	0
217	ุ	0	0	2	0	0	0	0	0	18	22	6	8	3	0	0	0	3	11	0	0	0	0	0	0	30	0
224	เ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0
226	โ	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
227	ใ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
228	ไ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
229	า	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
230	ๆ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0
231	็	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
232	่	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	4	0
233	้	3	4	4	0	0	0	0	1	25	35	3	0	0	0	0	0	2	1	0	0	0	0	0	0	25	1
234	๊	4	23	6	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	7	0	10
235	๋	0	0	0	0	0	0	1	0	0	0	0	0	0	5	2	0	0	0	0	0	1	0	0	0	0	0
236	์	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	1

ตารางที่ 5.4 (ต่อ)

จำนวนสตริง		568	454	567	566	206	492	452	460	514	501	551	444	477	390	88	387	479	400	460	512	109	507	511	502	509
ASCII		203	204	205	206	207	209	210	212	213	214	215	216	217	224	226	227	228	229	230	231	232	233	234	235	236
		ห	ฟ	อ	ฮ	ฯ	ั	า	ิ	ี	ึ	ื	ุ	ู	เ	แ	โ	ใ	ไ	ๅ	็	่	้	๊	๋	์
48	0	0	0	10	0	0	0	0	0	0	0	3	0	0	2	0	0	0	0	0	0	0	1	0	0	1
49	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	13	0	4	0	0	0	0	0	0	0
50	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
53	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
55	7	0	0	0	1	2	0	5	0	0	0	0	5	0	0	1	17	5	8	8	0	0	0	0	0	0
56	8	0	0	3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0
57	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
161	ก	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
162	ข	0	0	0	1	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0
163	ฃ	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
164	ค	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	ฅ	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
166	ฆ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0
167	ง	0	0	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
168	จ	0	0	36	5	0	0	0	0	0	0	1	12	0	0	1	0	0	0	0	0	0	0	0	2	0
169	ฉ	0	0	7	1	0	0	0	0	1	0	2	8	1	0	0	0	0	0	0	5	1	0	0	1	0
170	ช	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
171	ซ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
172	ฌ	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
173	ญ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
174	ฎ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
175	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	ฐ	0	1	0	27	1	0	0	0	6	2	0	0	0	0	0	0	0	0	0	21	0	1	0	0	4
177	ฑ	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
179	ณ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
180	ด	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
181	ต	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	1	0	0
182	ถ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
183	ท	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
184	ธ	0	0	0	6	1	0	0	0	0	0	0	0	0	0	4	0	0	0	0	1	0	1	0	1	0
185	น	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
186	บ	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
187	ป	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.4 (ต่อ)

จำนวนสตริง		568	454	567	566	206	492	452	460	514	501	551	444	477	390	88	387	479	400	460	512	109	507	511	502	509
ASCII		203	204	205	206	207	209	210	212	213	214	215	216	217	224	226	227	228	229	230	231	232	233	234	235	236
		ห	ฟ	อ	ฮ	ะ	ั	า	ิ	ี	ึ	ื	ุ	ู	เ	แ	โ	ใ	ไ	ๆ	็	่	้	๊	๋	์
190	พ	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
193	ม	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
194	ย	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
195	ร	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
197	ล	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
198	ฦ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0
199	ว	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
200	ศ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
201	ษ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
202	ส	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
203	ห	518	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
204	ฟ	2	421	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
205	อ	0	0	503	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
206	ฮ	0	0	0	505	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
207	ะ	0	0	0	0	168	0	3	0	2	0	4	0	1	20	0	0	0	0	7	1	2	0	0	2	0
209		0	0	0	0	4	466	0	0	0	0	10	0	8	0	0	0	0	0	0	1	0	61	2	2	0
210	า	0	0	2	0	3	0	416	0	0	0	0	0	0	0	0	1	0	88	9	0	0	0	0	0	0
212	ิ	0	0	0	0	0	0	0	423	18	9	6	3	2	0	0	0	0	0	0	8	0	0	1	2	0
213	ี	2	0	0	0	3	1	0	61	449	156	61	0	3	0	0	0	0	0	5	30	0	3	1	4	1
214	ึ	0	0	0	3	0	0	0	77	96	437	40	2	3	0	0	0	0	0	4	16	2	1	3	12	5
215	ื	3	0	0	0	0	0	0	0	23	9	426	2	5	0	0	0	0	0	0	0	1	0	0	0	2
216	ุ	0	0	3	0	4	0	0	1	0	0	21	395	13	19	0	2	2	0	1	0	62	2	3	17	1
217	ู	0	3	1	0	0	11	0	0	0	0	3	2	375	0	0	0	0	0	0	0	1	20	0	0	0
224	เ	0	0	0	0	8	0	0	0	0	0	0	0	0	367	0	4	0	0	0	0	33	0	0	2	0
226	แ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81	0	0	0	0	0	0	0	0	0	0
227	โ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	323	0	4	2	0	0	0	0	0	0
228	ใ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	468	0	0	0	0	0	0	0	0
229	ไ	0	0	0	0	0	0	65	0	0	0	0	0	0	0	0	7	0	369	0	0	0	0	0	0	0
230	ๅ	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	22	3	1	406	0	0	0	0	0	0
231	็	0	0	0	0	0	1	0	2	1	23	1	1	0	0	0	0	0	0	0	401	0	0	3	0	2
232	่	0	0	0	0	14	0	7	0	0	0	28	115	16	37	0	1	3	2	2	0	102	7	0	22	2
233	้	15	2	0	0	0	196	0	0	0	0	3	1	103	0	0	0	0	0	0	2	0	439	2	1	0
234	๊	13	0	0	0	8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	483	0	0
235	๋	0	0	0	0	0	1	0	0	0	2	0	0	1	0	0	0	0	0	0	2	0	5	1	421	1
236	์	0	0	2	3	0	0	0	10	29	19	4	11	8	0	0	0	0	0	1	28	1	3	0	14	491

ตารางที่ 5.5 แสดงผลการรู้จำของ Test Set โดยละเอียดแยกตามตัวอักษรของวิธี HMM

จำนวนสตริง		555	534	547	505	550	502	507	514	501	436	568	568	568	568	568	568	567	568	568	568	568	568	568	419	460	568
ASCII		48	49	50	51	52	53	54	55	56	57	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
		0	1	2	3	4	5	6	7	8	9	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ	ฐ
48	0	205	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	1	0	357	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	2	0	0	544	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	3	0	0	0	505	0	4	0	0	0	9	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	23
52	4	0	1	0	0	519	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
53	5	0	0	0	0	0	447	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	6	5	0	0	0	0	0	386	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	7	0	0	0	0	0	0	0	489	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	8	275	0	0	0	0	0	8	0	456	0	0	0	0	0	0	0	0	23	10	0	0	0	0	3	0	2
57	9	0	0	0	0	0	0	0	0	0	392	0	0	2	0	0	0	6	0	0	0	0	0	0	1	1	0
161	ก	0	0	0	0	0	0	0	0	0	0	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
162	ข	0	0	0	0	0	0	0	1	0	0	0	362	26	0	0	2	0	0	0	41	9	0	0	0	0	0
163	ฃ	0	1	0	0	1	0	0	1	0	0	0	118	429	0	0	42	0	0	0	0	34	0	0	0	0	5
164	ค	0	0	0	0	0	0	0	0	0	0	1	0	0	48	45	0	0	0	0	0	0	0	0	0	0	0
165	ฅ	0	0	0	0	0	0	0	0	0	0	1	0	0	2	248	0	0	0	0	0	1	0	0	0	0	0
166	ฆ	0	0	0	0	4	8	0	0	0	0	0	0	3	0	0	474	0	0	0	0	0	16	0	0	0	1
167	ง	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	494	0	0	0	0	0	0	4	2	9
168	จ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52	490	186	0	0	0	0	12	9	2
169	ฉ	0	23	1	0	5	0	0	0	0	0	0	0	0	0	0	0	0	29	308	0	0	0	0	20	12	2
170	ช	0	2	0	0	0	0	0	0	5	0	0	3	10	0	0	14	0	0	0	108	61	0	0	0	0	59
171	ซ	0	12	0	0	0	0	0	0	0	0	0	20	52	0	0	14	0	0	0	405	422	0	0	0	1	2
172	ฌ	0	0	0	0	0	0	0	4	0	0	0	0	2	0	0	13	0	0	2	0	0	463	5	1	1	0
173	ญ	0	0	0	0	0	0	0	0	0	0	9	0	10	0	38	0	0	0	2	0	1	15	552	0	0	0
174	ฎ	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0	304	260	0
175	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	162	0
176	ฐ	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	5	0	0	0	0	0	0	363
177	ฑ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	0	0	0	0	0	0	0	0	4	0	0	1	0	0	8	0	1	0	0	0	0	0	0	0	0
179	ณ	0	0	0	0	0	0	0	0	0	0	0	0	8	0	2	0	0	0	1	0	0	5	9	0	0	0
180	ด	0	0	0	0	0	0	68	0	0	0	0	0	0	515	211	0	0	0	0	0	0	0	0	0	0	0
181	ต	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
182	ถ	0	0	0	0	0	0	0	0	0	0	390	0	0	1	6	0	0	0	3	0	0	0	0	0	0	0
183	ท	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
184	ธ	0	4	0	0	0	3	2	0	4	0	0	0	3	0	0	0	1	2	5	0	0	0	0	6	1	13
185	น	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	1	0	0	0	9	0	6	0	1	0	0
186	บ	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	1	0	0	0	0	0	0
187	ป	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.5 (ต่อ)

จำนวนสถิติ		555	534	547	505	550	502	507	514	501	436	568	568	568	568	568	568	568	567	568	568	568	568	568	568	419	460	568
ASCII		48	49	50	51	52	53	54	55	56	57	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	
		0	1	2	3	4	5	6	7	8	9	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ	ฐ	
190	พ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	0	0	0	0	0	0	0	0	0	20	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
193	ม	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	61	0	0	0	0	0
194	ย	1	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
195	ร	0	1	0	0	0	19	0	0	0	1	0	0	0	0	0	0	3	0	1	0	0	0	0	1	1	50	
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
197	ล	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
198	ฦ	0	2	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0
199	ว	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	19	9	0	
200	ศ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	
201	ษ	0	0	0	0	0	0	0	0	0	1	0	4	5	0	0	0	0	0	0	4	7	0	0	0	0	5	
202	ส	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
203	ห	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	5	0	0	0	0	0	0	0	
204	ฬ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
205	อ	0	9	0	0	1	19	0	0	0	15	0	0	0	0	0	0	0	0	2	0	8	0	1	2	0	4	
206	ฮ	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	
207	ฯ	0	80	0	0	0	0	0	0	0	6	0	0	5	0	0	0	9	0	0	0	0	0	0	0	0	0	
209		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
210	า	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
212	ำ	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
213	ิ	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
214	ี	2	2	0	0	0	0	17	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
215	ึ	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
216	ุ	6	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	11	4	0	0	0	0	0	0	0	
217	ู	0	0	0	0	0	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	1	0	0	0	0	
224	เ	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
226	โ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
227	ใ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
228	ไ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
229	ๅ	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
230	ๆ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	
231	็	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	7	
232	่	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
233	้	0	0	0	0	0	0	0	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0	1	
234	๊	3	0	0	0	3	1	1	12	10	0	1	0	3	2	5	0	0	0	21	0	23	1	0	0	0	0	
235	๋	0	4	0	0	7	0	1	0	8	0	0	0	0	0	0	0	0	9	3	0	0	0	0	0	0	0	
236	์	55	0	0	0	10	0	20	7	13	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	

ตารางที่ 5.5 (ต่อ)

จำนวนสตริง		456	568	568	565	568	568	565	568	566	567	419	568	375	568	408	568	568	568	566	423	567	444	568	513	506	568
ASCII		177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202
		ท	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ	ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0
52	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
53	5	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	6	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
161	ก	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
162	ข	0	0	0	0	0	0	0	0	1	14	21	0	0	0	0	0	34	1	0	0	0	0	0	0	0	0
163	ฃ	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
164	ค	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
165	ฅ	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
166	ฆ	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0
167	ง	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
168	จ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
169	ฉ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
170	ช	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
171	ซ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
172	ฌ	0	214	21	0	8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
173	ญ	0	0	31	0	39	3	5	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	2	0	0
174	ฎ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
175	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	ฐ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
177	ท	456	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	321	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
179	ณ	0	25	514	0	9	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
180	ด	0	0	0	565	351	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0
181	ต	0	0	0	0	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
182	ถ	0	0	0	0	8	563	0	21	0	0	0	0	0	0	0	24	0	0	0	18	39	0	0	0	0	0
183	ท	0	0	0	0	0	0	145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
184	ธ	0	0	0	0	0	0	0	443	0	0	0	0	0	0	0	0	0	0	37	0	2	0	0	0	0	10
185	น	0	0	1	0	0	0	0	0	553	8	15	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
186	บ	0	0	0	0	0	0	0	0	6	528	346	0	0	0	0	0	33	0	0	0	0	0	0	0	0	0
187	ป	0	0	0	0	0	0	0	0	4	14	33	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	0	0	0	0	0	0	0	484	245	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	81	126	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.5 (ต่อ)

จำนวนสตริง		456	568	568	565	568	568	565	568	566	567	419	568	375	568	408	568	568	568	566	423	567	444	568	513	506	568
ASCII		177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202
		ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ	ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส
190	พ	0	0	0	0	0	0	0	0	0	0	0	0	0	383	172	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	0	0	0	0	0	0	0	0	0	0	0	0	176	228	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	536	0	0	9	0	0	19	0	0	0	0
193	ม	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	478	0	0	0	0	0	0	0	0	0
194	ย	0	0	0	0	0	0	0	0	0	0	0	3	4	0	0	0	0	567	0	0	0	0	0	0	9	0
195	ร	0	0	0	0	0	0	0	72	0	0	0	0	0	0	0	0	0	0	494	0	0	0	1	0	0	0
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	179	0	1	0	0	0	0
197	ล	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	23	0	522	0	0	0	0	0
198	ฦ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	226	0	424	17	0	0	0
199	ว	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	511	0	0	0
200	ศ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	469	0	4
201	ษ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	491	0
202	ส	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	547
203	ห	0	0	0	0	0	0	365	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
204	ฬ	0	0	0	0	0	0	0	0	0	0	0	0	0	9	7	0	0	0	0	0	0	0	0	0	0	0
205	อ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
206	ฮ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
207	ฯ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
209		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
210	า	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
212	ิ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
213	ี	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
214	ึ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
215	ื	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
216	ุ	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
217	ู	0	2	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
224	เ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
226	โ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
227	ใ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
228	ไ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
229	ๅ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
230	ๆ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
231	็	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
232	่	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
233	้	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
234	๊	0	3	1	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	7
235	๋	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
236	์	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.5 (ต่อ)

จำนวนสตริง		568	454	567	566	206	492	452	460	514	501	551	444	477	390	88	387	479	400	460	512	109	507	511	502	509
ASCII		203	204	205	206	207	209	210	212	213	214	215	216	217	224	226	227	228	229	230	231	232	233	234	235	236
		ห	ฟ	อ	ฮ	ฯ	ะ	า	ิ	ี	ึ	ื	ุ	ู	เ	แ	โ	ใ	ไ	ๅ	็	่	้	๊	๋	์
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
50	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	6	0	0	1	0	0	1	0	14	0	0	0	0	1	0	0	0	0	0	0	43	0	0	0	0	1
55	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	8	0	0	51	1	0	0	0	6	0	0	0	1	0	4	0	0	0	0	0	0	0	0	0	2	1
57	9	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
161	ก	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
162	ข	0	0	0	0	0	0	0	0	0	0	0	0	17	20	0	0	0	0	0	0	0	0	0	0	0
163	ฃ	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	12	0	8	0	0	0	0	0	0
164	ค	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	ฅ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
166	ฆ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
167	ง	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
168	จ	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
169	ฉ	0	0	0	1	2	0	9	0	0	0	1	4	0	1	1	62	2	47	0	0	0	2	0	4	0
170	ช	0	0	0	227	1	0	1	0	0	1	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0
171	ซ	0	0	0	4	9	0	0	0	0	0	0	0	0	0	0	66	1	10	179	0	0	0	3	0	0
172	ฌ	0	2	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	2	0	3	0	1	1	0
173	ญ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	1	0	0
174	ฎ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
175	ฏ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	ฐ	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	1	0	0
177	ฑ	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	ฒ	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
179	ณ	0	3	0	0	3	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	2	0	1	0	0
180	ด	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
181	ต	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0
182	ถ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
183	ท	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
184	ธ	0	0	8	59	0	0	1	0	0	0	0	4	0	0	5	10	0	0	0	4	0	0	0	2	0
185	น	0	0	0	0	0	0	0	0	0	0	0	1	11	0	0	0	0	0	0	0	0	0	0	0	0
186	บ	0	0	0	0	1	0	1	0	0	0	0	0	99	1	0	0	0	0	0	0	0	1	0	0	0
187	ป	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
188	ผ	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
189	ฝ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ 5.5 (ต่อ)

จำนวนสตริง		568	454	567	566	206	492	452	460	514	501	551	444	477	390	88	387	479	400	460	512	109	507	511	502	509
ASCII		203	204	205	206	207	209	210	212	213	214	215	216	217	224	226	227	228	229	230	231	232	233	234	235	236
		ห	ฟ	อ	ฮ	ฯ	◌	า	◌	◌	◌	◌	◌	◌	เ	โ	ใ	ไ	ๅ	ๆ	◌	◌	◌	◌	◌	◌
190	พ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
191	ฟ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	ภ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
193	ม	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
194	ย	0	0	0	0	0	4	0	0	0	0	0	0	0	5	0	0	0	0	0	6	0	0	0	0	0
195	ร	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
196	ฤ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
197	ล	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
198	ฦ	0	0	0	0	0	0	9	0	0	0	0	0	0	2	0	21	0	57	116	0	0	0	0	0	0
199	ว	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
200	ศ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
201	ษ	0	0	0	4	0	0	3	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0
202	ส	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
203	ห	567	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
204	ฟ	0	448	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
205	อ	0	0	501	40	1	0	1	0	0	0	2	4	0	4	0	2	0	6	0	2	0	0	0	0	2
206	ฮ	0	0	0	188	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0
207	ฯ	0	0	0	1	168	0	18	0	0	0	0	2	1	2	0	0	0	105	0	0	0	7	0	3	0
209	◌	0	0	0	0	0	297	0	0	5	0	2	0	7	0	0	0	0	0	0	1	0	30	0	0	0
210	า	0	0	0	0	0	0	390	0	0	0	0	12	0	0	0	0	0	84	0	0	0	0	0	0	0
212	◌	0	0	0	0	0	0	0	399	0	14	0	14	7	0	0	0	0	0	0	0	4	1	0	0	0
213	◌	0	0	0	0	0	0	0	2	288	24	15	0	0	0	0	0	0	0	0	1	0	0	0	1	1
214	◌	0	0	0	0	0	0	0	18	24	320	8	6	0	0	0	0	0	1	0	25	10	0	0	0	6
215	◌	0	0	0	0	0	0	0	1	142	40	406	0	0	0	0	0	0	0	0	0	0	0	0	3	1
216	◌	0	0	1	0	0	0	0	0	0	0	0	285	2	0	0	0	0	0	0	0	0	0	0	0	0
217	◌	0	0	0	0	0	26	0	0	12	0	73	0	273	0	0	0	0	0	0	0	0	8	3	2	0
224	เ	0	0	0	0	1	1	0	0	0	0	0	2	1	177	0	0	0	0	0	0	4	0	0	0	0
226	โ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78	0	0	0	0	0	0	0	0	0	0
227	ใ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	217	0	2	0	0	0	0	0	0	0
228	ไ	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	461	0	0	0	0	0	0	0	0
229	ๅ	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	82	0	0	0	0	0	0	0
230	ๆ	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0	0	120	0	0	0	0	0	0
231	◌	0	0	0	10	0	0	0	0	6	3	4	0	0	0	0	0	0	0	0	261	0	0	0	0	21
232	◌	0	0	0	0	4	0	0	0	0	0	21	38	6	2	0	0	0	0	0	0	72	0	0	0	0
233	◌	0	0	0	13	1	136	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	439	0	1	1
234	◌	0	0	0	1	6	22	0	7	6	8	11	7	15	3	0	0	0	0	16	56	0	15	501	20	15
235	◌	0	0	0	0	0	5	9	2	7	3	1	53	13	142	0	5	2	0	0	1	6	2	0	453	5
236	◌	0	0	1	0	0	0	0	11	24	88	7	9	5	8	0	0	0	0	0	106	7	0	0	5	458

จากการทดสอบการรู้จำด้วยวิธีต่างๆทั้ง 3 วิธี ดังที่แสดงข้างต้น จะสามารถสรุปผลการรู้จำเป็นเปอร์เซ็นต์โดยรวมได้ในตารางที่ 5.5 เมื่อนำมาทดสอบกับ Training Set และ Test Set โดยวิธีที่นำเสนอในวิทยานิพนธ์ฉบับนี้ จะแบ่งเปอร์เซ็นต์การรู้จำเป็นสตริงที่มีผู้ชนะเป็น DFA ตัวเดียว และสตริงที่มีผู้ชนะเป็น DFA ตั้งแต่ 2 ตัวขึ้นไป

ตารางที่ 5.6 แสดงผลรวมการรู้จำด้วยวิธีต่างๆสำหรับข้อมูลขนาดใหญ่

Method	Training Set (15434 samples from 77 classes)			Test Set (38843 samples from 77 classes)		
	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate
Positive	84.16% (12989)	15.71% (2425)	0.13% (20)	74.78% (29046)	13.77% (5349)	11.45% (4448)
Positive & Negative	84.20% (12995)	15.71% (2424)	0.10% (15)	75.16% (29193)	13.63% (5294)	11.21% (4356)
Multimodal	93.97% (14503)	5.02% (775)	1.01% (156)	84.01% (32633)	3.73% (1449)	12.26% (4761)
HMM	75.37% (11629)	-	24.63% (3805)	72.61% (28206)	-	27.39% (10637)

จะเห็นได้ว่า ผลการรู้จำของวิธี Positive Training และวิธีที่ใช้ทั้ง Positive และ Negative Training นั้น ไม่มีความแตกต่างกันมากนัก สำหรับข้อมูลที่มีขนาดใหญ่ คือ ดีขึ้นประมาณ 0.38% จึงทำให้เกิดแนวคิดที่จะใช้วิธี Multimodal ขึ้น เพื่อให้สามารถรู้จำได้ดีขึ้น โดยการพยายามแบ่งกลุ่มตัวอักษรที่มีลักษณะใกล้เคียงกันมาอยู่กลุ่มเดียวกัน และสร้างโมเดลสำหรับแต่ละกลุ่มขึ้นมา ซึ่งช่วยลดความกำกวมของตัวอักษรได้มากขึ้น คือ ได้ผลดีกว่าสองวิธีแรกประมาณ 9 – 10% เลยทีเดียว และลดการยอมรับ DFA หลายๆตัวไปได้ประมาณ 10% เช่นกัน

แต่สำหรับข้อมูลขนาดเล็ก ซึ่งได้ทำการทดลองมาก่อนหน้านี้ กับกลุ่มตัวอย่างที่มีจำนวนสตริงทั้งหมด 5906 สตริง จากตัวอักษร 66 คลาส จะพบว่า วิธีการใช้ Negative Training เข้ามาร่วมด้วยนั้น มีผลให้การรู้จำดีกว่าการใช้ Positive Training ประมาณ 2.64% ดังตารางที่ 5.6 จึงมิอาจสรุปได้ว่า วิธีการนำ Noise มาช่วยปฏิเสธสตริงที่ไม่ต้องการออกไปนั้น ไม่มีผลใดๆกับการรู้จำเลย ทั้งนี้น่าจะขึ้นอยู่กับลักษณะข้อมูลแต่ละตัว รวมทั้งการทดสอบกับกลุ่มข้อมูลที่มีขนาดแตกต่างกัน ก็มีผลทำให้เปอร์เซ็นต์มีความเปลี่ยนแปลงได้

ดังนั้น สำหรับกลุ่มข้อมูลขนาดใหญ่ ที่มีผลการรู้จำใกล้เคียงกัน สำหรับวิธีที่ใช้และไม่ใช้ขั้นตอน Negative Training น่าจะถือได้ว่า การใช้เฉพาะข้อมูลตำแหน่งและความยาวก็พอเพียงต่อการแยกแยะตัวอักษรแล้ว ทำให้การใช้ Noise ของสตริงที่เป็นลบซ้ำซ้อนกัน และไม่ส่งผลให้เห็น

ความแตกต่างในการรู้จำ ในขณะที่การแบ่งกลุ่มข้อมูล มีผลทำให้การรู้จำดีขึ้นกว่าเดิมมาก แต่ในกรณีข้อมูลขนาดเล็ก ไม่เหมาะที่จะใช้วิธีการแบ่งกลุ่มข้อมูล

ตารางที่ 5.7 แสดงผลรวมการรู้จำด้วยวิธีต่างๆสำหรับข้อมูลขนาดเล็ก

Method	Training Set (4920 samples from 66 classes)			Test Set (986 samples from 66 classes)		
	Recognition Rate	Accept more than one DFA	Inaccuracy Rate	Recognition Rate	Accept more than one DFA	Inaccuracy Rate
Positive	98.68% (4855)	1.20% (59)	0.12% (6)	82.05% (809)	0.61% (6)	17.34% (171)
Positive & Negative	98.72% (4857)	1.20% (59)	0.08% (4)	84.69% (835)	0.61% (6)	14.71% (145)
Multimodal	99.02% (4872)	0.24% (12)	0.73% (36)	80.22% (791)	0.51% (5)	19.27% (190)
HMM	85.61%	-	14.39%	82.56%	-	17.44%

นอกจากผลการรู้จำแล้ว เมื่อเปรียบเวลาที่ใช้ในการสร้างแมทชีน (Training) และเวลาในการรู้จำเปรียบเทียบระหว่างวิธีที่นำเสนอทั้งสาม กับวิธี HMM พบว่า วิธีที่นำเสนอ โดยเฉพาะวิธี Positive Training และวิธีที่เพิ่ม Negative Training นั้น ใช้เวลาในการสร้างแมทชีนน้อยมาก อย่างวิธี Positive Training ใช้เวลาเพียง 15 วินาที ก็สามารถสร้างแมทชีนได้ครบทั้ง 77 แมทชีนแล้ว หรือโดยเฉลี่ยสามารถสร้างแมทชีนได้ 5 – 6 แมทชีนต่อ 1 วินาที ส่วนวิธีที่ใช้ทั้ง Positive และ Negative Training ก็ใช้เวลาที่ต้องทำการทดสอบกับสตริงที่เป็นลบเพิ่มขึ้นมา จึงใช้เวลาในการสร้างแมทชีนละประมาณ 2 – 3 วินาที ซึ่งนับว่าน้อยมาก แม้กระทั่งวิธีที่ต้องอาศัยเวลาในการสร้างแมทชีนนานที่สุด อย่างวิธี Multimodal ซึ่งต้องมีการสร้างแมทชีนขึ้นมาเป็นจำนวนมาก รวมทั้งต้องเสียเวลาในการพยายามยุบรวมแมทชีนหลายต่อหลายรอบ แต่ก็ยังใช้เวลาสร้างแมทชีนเฉลี่ยเพียงแมทชีนละ 31 วินาทีเท่านั้น เมื่อเปรียบเทียบกับวิธี HMM ที่ต้องใช้เวลาในการ Training นานมากถึงกว่า 2 วันครึ่ง หรือเฉลี่ยแมทชีนละ 47 นาทีเลยทีเดียว

ตารางที่ 5.8 จะเป็นตารางเปรียบเทียบเวลาที่ใช้ในขั้นตอนการสร้างแมทชีน หรือการ Training ของวิธีทั้ง 3 ที่นำเสนอ และวิธี HMM แยกตามรายตัวอักษร และตารางที่ 5.9 จะเป็นตารางสรุปเวลาที่ใช้ทั้งในการ Training และการรู้จำ (Recognition) ของแต่ละวิธี โดยเวลาที่ปรากฏในตารางเหล่านี้นั้น ได้ทำการทดสอบรันโปรแกรมบนเครื่องคอมพิวเตอร์ซึ่งมี Intel® Core™2Duo 2.66 GHz และมี RAM 2 GB บนระบบปฏิบัติการ Windows XP (service pack 2)

ตารางที่ 5.8 แสดงเวลาที่ใช้ในขั้นตอน Training แยกตามตัวอักษรเปรียบเทียบกับวิธีต่างๆ

ASCII	Symbol	#Training Samples	Training Time (hh:mm:ss)			
			Positive	Positive & Negative	Multimodal	HMM
48	0	200	Less than 1 sec	0:00:01	0:00:02	2:34:02
49	1	201	Less than 1 sec	0:00:02	0:00:07	0:50:41
50	2	200	Less than 1 sec	0:00:01	0:00:24	0:17:58
51	3	202	Less than 1 sec	0:00:01	0:00:26	0:30:21
52	4	200	Less than 1 sec	0:00:02	0:00:12	2:16:49
53	5	205	Less than 1 sec	0:00:01	0:00:42	1:03:35
54	6	200	Less than 1 sec	0:00:01	0:00:06	2:17:40
55	7	200	Less than 1 sec	0:00:01	0:00:11	0:26:53
56	8	200	Less than 1 sec	0:00:01	0:00:00	1:14:29
57	9	200	Less than 1 sec	0:00:01	0:00:07	0:30:39
161	ก	200	Less than 1 sec	0:00:01	0:00:16	0:03:45
162	ข	200	Less than 1 sec	0:00:02	0:00:36	0:08:57
163	ฃ	200	Less than 1 sec	0:00:02	0:01:01	0:09:13
164	ค	200	Less than 1 sec	0:00:01	0:00:21	0:16:59
165	ฅ	200	Less than 1 sec	0:00:02	0:00:27	1:02:52
166	ฆ	200	Less than 1 sec	0:00:02	0:01:26	1:23:41
167	ง	201	Less than 1 sec	0:00:00	0:00:09	0:10:00
168	จ	200	Less than 1 sec	0:00:02	0:00:20	0:13:30
169	ฉ	200	Less than 1 sec	0:00:02	0:00:45	1:39:09
170	ช	200	Less than 1 sec	0:00:02	0:01:07	0:15:59
171	ซ	200	Less than 1 sec	0:00:02	0:01:37	0:23:22
172	ฌ	200	Less than 1 sec	0:00:01	0:01:21	1:18:26
173	ญ	200	Less than 1 sec	0:00:01	0:00:48	0:19:46
174	ฎ	200	Less than 1 sec	0:00:02	0:01:21	1:47:11
175	ฏ	200	Less than 1 sec	0:00:01	0:01:48	2:01:50
176	ฐ	200	Less than 1 sec	0:00:02	0:01:15	0:40:42
177	ฑ	204	Less than 1 sec	0:00:01	0:01:08	0:10:22
178	ฒ	200	Less than 1 sec	0:00:01	0:01:39	1:04:35
179	ณ	200	Less than 1 sec	0:00:01	0:01:30	0:18:50
180	ด	203	Less than 1 sec	0:00:01	0:00:16	1:18:12
181	ต	200	Less than 1 sec	0:00:02	0:00:29	0:09:25
182	ถ	200	Less than 1 sec	0:00:00	0:00:24	0:08:25
183	ท	203	Less than 1 sec	0:00:01	0:00:07	0:15:00
184	ธ	200	Less than 1 sec	0:00:02	0:00:41	0:15:46
185	น	202	Less than 1 sec	0:00:01	0:00:25	0:19:52
186	บ	201	Less than 1 sec	0:00:02	0:00:10	2:18:29
187	ป	200	Less than 1 sec	0:00:01	0:00:12	0:18:25
188	ผ	200	Less than 1 sec	0:00:00	0:00:14	0:27:52
189	ฝ	201	Less than 1 sec	0:00:01	0:00:32	0:04:30

ตารางที่ 5.8 (ต่อ)

ASCII	Symbol	#Training Samples	Training Time (hh:mm:ss)			
			Positive	Positive & Negative	Multimodal	HMM
190	พ	200	Less than 1 sec	0:00:01	0:00:26	0:08:06
191	ฟ	200	Less than 1 sec	0:00:01	0:00:22	0:08:09
192	ภ	200	Less than 1 sec	0:00:01	0:00:26	0:32:28
193	ม	200	Less than 1 sec	0:00:01	0:00:26	0:10:49
194	ย	200	Less than 1 sec	0:00:01	0:00:35	0:22:52
195	ร	202	Less than 1 sec	0:00:01	0:00:24	1:02:30
196	ฤ	200	Less than 1 sec	0:00:01	0:00:24	0:19:36
197	ล	200	Less than 1 sec	0:00:01	0:00:42	0:11:31
198	ฦ	200	Less than 1 sec	0:00:01	0:00:28	0:08:45
199	ว	200	Less than 1 sec	0:00:01	0:00:12	0:07:14
200	ศ	201	Less than 1 sec	0:00:01	0:00:57	0:44:21
201	ษ	200	Less than 1 sec	0:00:01	0:01:19	0:11:23
202	ส	200	Less than 1 sec	0:00:02	0:01:49	0:26:44
203	ห	200	Less than 1 sec	0:00:01	0:00:40	0:17:00
204	ฬ	200	Less than 1 sec	0:00:01	0:01:03	0:12:29
205	อ	201	Less than 1 sec	0:00:02	0:00:16	2:25:10
206	ฮ	202	Less than 1 sec	0:00:02	0:00:29	0:58:03
207	ฯ	200	Less than 1 sec	0:00:03	0:00:13	0:13:17
209	ๆ	200	Less than 1 sec	0:00:01	0:00:03	2:08:49
210	า	200	Less than 1 sec	0:00:01	0:00:03	2:11:32
212	ิ	200	Less than 1 sec	0:00:02	0:00:01	2:12:36
213	ี	200	Less than 1 sec	0:00:01	0:00:08	0:09:42
214	ึ	200	Less than 1 sec	0:00:03	0:00:08	0:16:47
215	ื	200	Less than 1 sec	0:00:02	0:00:16	0:30:32
216	ุ	200	Less than 1 sec	0:00:02	0:00:03	1:57:16
217	ู	200	Less than 1 sec	0:00:03	0:00:14	0:40:32
224	เ	200	Less than 1 sec	0:00:02	0:00:02	0:18:36
226	โ	200	Less than 1 sec	0:00:01	0:00:09	0:10:26
227	ใ	200	Less than 1 sec	0:00:02	0:00:11	0:17:25
228	ไ	202	Less than 1 sec	0:00:02	0:00:12	0:23:17
229	ๅ	200	Less than 1 sec	0:00:01	0:00:04	0:09:38
230	ๆ	200	Less than 1 sec	0:00:01	0:00:36	0:11:11
231	็	200	Less than 1 sec	0:00:03	0:00:22	1:56:58
232	่	200	Less than 1 sec	0:00:02	0:00:01	2:04:31
233	้	200	Less than 1 sec	0:00:02	0:00:19	0:11:08
234	๊	200	Less than 1 sec	0:00:02	0:01:00	1:47:59
235	๋	200	Less than 1 sec	0:00:02	0:00:18	1:21:49
236	์	203	Less than 1 sec	0:00:05	0:00:06	2:23:42
Total Time			**0:00:15**	**0:01:54**	**0:40:09**	**61:13:05**
Average Time per machine			**Less than1sec**	**0:00:01**	**0:00:31**	**0:47:42**

ตารางที่ 5.9 แสดงผลสรุปเวลาที่ใช้ในขั้นตอน Training และ Recognition เปรียบเทียบกับวิธีต่างๆ

Method	Training Time		Recognition Time			
	Training Set (15434 samples from 77 classes)		Training Set (15434 samples from 77 classes)		Test Set (38843 samples from 77 classes)	
	Total Time	Average Time per machine	Total Time	Average Time per sample	Total Time	Average Time per sample
Positive	0:00:15	≈ 0.19 s	0:06:06	≈ 0.024 s	0:19:33	≈ 0.030 s
Positive & Negative	0:01:54	0:00:01	0:04:53	≈ 0.019 s	0:16:54	≈ 0.026 s
Multimodal	0:40:09	0:00:31	0:20:44	≈ 0.081 s	1:08:24	≈ 0.106 s
HMM	61:13:05	0:47:42	0:23:24	≈ 0.091 s	2:28:44	≈ 0.230 s

สำหรับวิธี HMM นั้น นอกจากจะใช้เวลาในการสร้างโมเดลนานมากแล้ว เวลาที่ใช้ในการรู้จำของสตริงแต่ละตัวก็ยังใช้เวลามากกว่าอีกด้วย จากตารางที่ 5.9 จะเห็นว่า เวลาที่ใช้ในการรู้จำ (Recognition Time) ของวิธี HMM นั้นมากกว่าเวลาที่ใช้ของวิธีที่นำเสนอทั้งสาม โดยสำหรับวิธีที่มี 1 โมเดลต่อ 1 คลาสเหมือนกัน คือ วิธีที่ใช้ Positive Training และวิธีที่ใช้ Positive และ Negative Training นั้น ใช้เวลาน้อยกว่าวิธี HMM ถึง 4 – 5 เท่า หรือแม้กระทั่งวิธีที่มีหลายๆโมเดลต่อ 1 คลาส อย่างวิธี Multimodal ก็ตาม วิธี HMM ก็ยังคงใช้เวลาในการรู้จำมากกว่าเช่นกัน จึงอาจสรุปได้ว่าข้อดีของวิธีที่นำเสนอทั้งสามแบบนี้ มีความได้เปรียบในเรื่องของเวลาที่ใช้ทั้งในขั้นตอนการสร้างโมเดล และการรู้จำเป็นอย่างมาก

สำหรับวิธี Multimodal นั้น มีข้อสังเกตในเรื่องจำนวนแมทชีนที่ได้สำหรับแต่ละคลาส ซึ่งการสร้างแมทชีนหลายๆแมทชีนเพื่อเป็นตัวแทนของสตริงในแต่ละกลุ่มนั้น โดยจะเริ่มจากการสร้างแมทชีนตามกลุ่มขนาดย่อยๆที่คล้ายคลึงกันมากที่สุดก่อน จากนั้นจะยุบรวมแมทชีนของกลุ่มสตริงที่ใกล้เคียงกันเข้าไว้ด้วยกัน จำนวนแมทชีนสุดท้ายที่ได้จะต้องไม่มากและไม่น้อยเกินไป ทั้งนี้ขึ้นอยู่กับเงื่อนไขในแต่ละรอบของการยุบรวมแมทชีนนั้นๆด้วย ตารางที่ 5.10 จะแสดงจำนวนแมทชีนที่ได้ในแต่ละรอบ ทั้ง 8 รอบนับตั้งแต่การสร้างแมทชีนเริ่มต้น และการยุบรวมตามเงื่อนไขต่างๆ โดยเงื่อนไขในการพิจารณาว่าสตริงหรือแมทชีนสามารถนำมารวมกันได้หรือไม่ จะเป็นไปตามเงื่อนไขที่ระบุไว้ในบทที่ 4

ตารางที่ 5.10 แสดงจำนวนแมทชีนที่ได้ในแต่ละรอบสำหรับวิธี Multimodal

ASCII	Symbol	#Training Samples	Number of Machines in each Round							
			1st	2nd	3rd	4th	5th	6th	7th	8th
48	0	200	18	7	7	4	4	4	4	4
49	1	201	98	58	44	26	20	16	15	9
50	2	200	158	102	73	35	29	22	19	6
51	3	202	169	82	69	20	18	14	14	8
52	4	200	101	70	63	46	41	32	31	10
53	5	205	185	140	114	42	25	8	8	4
54	6	200	94	40	11	10	6	6	6	3
55	7	200	103	86	69	39	35	22	18	6
56	8	200	20	9	6	5	5	5	5	4
57	9	200	89	50	19	8	6	3	3	2
161	ก	200	157	72	54	29	21	16	16	7
162	ข	200	180	129	97	55	39	33	24	11
163	ช	200	185	157	137	92	69	33	28	10
164	ค	200	169	76	50	25	20	16	16	8
165	ค	200	168	99	57	22	14	8	7	4
166	ฆ	200	190	183	143	100	66	30	21	10
167	ง	201	110	53	40	10	10	8	8	7
168	จ	200	168	92	71	24	21	18	17	10
169	ฉ	200	179	136	95	67	46	33	28	15
170	ช	200	182	147	129	94	75	64	53	16
171	ซ	200	192	183	171	118	90	58	55	16
172	ฌ	200	198	164	145	58	46	22	22	8
173	ญ	200	173	145	104	65	34	28	22	6
174	ฎ	200	188	165	142	61	47	26	23	14
175	ฏ	200	180	162	137	105	62	33	28	11
176	ฐ	200	188	162	151	79	66	45	40	20
177	ฑ	204	190	170	106	63	33	27	21	6
178	ฒ	200	191	155	129	75	42	29	24	14
179	ณ	200	196	150	141	54	39	26	19	7
180	ด	203	138	52	35	18	15	12	11	5
181	ต	200	155	92	67	18	17	17	17	6
182	ถ	200	166	60	53	26	22	20	18	7
183	ท	203	72	50	22	20	12	12	8	4
184	ธ	200	181	120	91	60	42	22	21	10
185	น	202	174	106	59	34	21	20	17	5
186	บ	201	113	55	43	29	26	19	18	7
187	ป	200	123	57	39	28	24	20	19	13
188	ผ	200	122	83	47	30	21	17	16	9
189	ฝ	201	166	118	86	30	19	15	14	11

ตารางที่ 5.10 (ต่อ)

ASCII	Symbol	#Training Samples	Number of Machines in each Round							
			1st	2nd	3rd	4th	5th	6th	7th	8th
190	พ	200	159	112	56	36	27	22	20	8
191	ฟ	200	147	97	41	22	16	14	11	5
192	ภ	200	171	124	82	34	23	12	11	4
193	ม	200	168	71	41	31	26	22	19	9
194	ย	200	180	111	85	38	27	24	23	11
195	ร	202	156	114	51	28	25	12	11	4
196	ฤ	200	165	54	41	20	18	15	13	5
197	ล	200	186	98	79	25	16	11	11	5
198	ภ	200	154	79	53	35	23	17	17	11
199	ว	200	148	50	35	13	12	5	5	3
200	ศ	201	171	135	107	67	47	28	26	10
201	ษ	200	186	155	125	97	75	61	54	15
202	ส	200	190	170	138	94	64	37	34	6
203	ห	200	164	99	70	43	24	21	21	11
204	ฬ	200	188	138	95	33	29	13	13	7
205	อ	201	143	64	47	17	16	14	14	10
206	ฮ	202	154	115	79	54	34	19	18	10
207	ฯ	200	114	70	48	37	24	22	18	16
209	๊	200	66	24	17	9	9	5	5	3
210	า	200	55	29	19	14	12	8	8	6
212	็	200	43	28	18	12	8	4	4	3
213	่	200	108	47	32	15	13	10	9	5
214	้	200	87	53	19	15	8	8	8	8
215	๋	200	131	73	44	29	23	21	12	8
216	์	200	67	39	20	13	10	9	9	8
217	๎	200	125	63	39	23	19	19	15	10
224	เ	200	50	25	17	11	10	9	9	4
226	โ	200	102	69	26	16	3	3	3	2
227	ใ	200	127	76	32	21	17	15	12	5
228	ไ	202	135	67	54	21	14	9	9	7
229	า	200	92	37	19	14	7	6	6	5
230	ๆ	200	176	110	95	52	39	31	25	12
231	็	200	146	116	93	61	40	36	29	20
232	ั	200	32	17	8	6	5	5	4	3
233	ี	200	145	107	80	69	44	24	20	12
234	ื	200	187	154	137	68	40	22	19	9
235	ิ	200	124	73	64	48	40	23	22	14
236	ี	203	71	45	27	19	16	11	11	8

5.3 การวิเคราะห์และการเปรียบเทียบกับวิธีต่างๆ

จากการทดสอบขั้นตอนวิธีต่างๆข้างต้น โดยได้ทำการสร้างแบบจำลองและรู้จำกับชุดสตริงจากตัวอักษรต่างๆ จะเห็นว่า เปอร์เซ็นต์การรู้จำอาจจะยังไม่ดีเท่าที่ควร ในหัวข้อนี้ จะวิเคราะห์และอธิบายปัญหาต่างๆที่พบในการสร้างโมเดลลักษณะนี้ และเปรียบเทียบกับวิธีการอื่นๆ รวมทั้งความเหมาะสมกับข้อมูลตัวอย่างในรูปแบบอื่นๆด้วย

5.3.1 วิเคราะห์ปัญหาที่พบในการรู้จำสำหรับวิธีที่นำเสนอ

เมื่อพิจารณาผลการรู้จำรายตัวอักษรแล้ว จะพบว่า แม้บางตัวอักษรที่ได้ผลลัพธ์ที่ดี คือ มีเปอร์เซ็นต์การรู้จำสูง แต่จะมีบางตัวอักษรที่มีเปอร์เซ็นต์การรู้จำ (Recognition Rate) ค่อนข้างต่ำในขณะที่มีจำนวนสตริงที่มีความกำกวม หรือ สตริงที่มีผู้ชนะตั้งแต่ DFA ขึ้นไป (Accept more than one DFA) สูง และฉุดให้เปอร์เซ็นต์การรู้จำโดยรวมตกลงไป นั่นแสดงให้เห็นว่า มีบางตัวอักษรที่มีลักษณะคล้ายคลึงกัน จึงทำให้เมื่อทดสอบแล้ว ผลที่ได้จึงอาจจะระบุว่าเป็นตัวอักษรได้มากกว่า 1 ตัว จึงอาจจำแนกลักษณะความผิดพลาดในการรู้จำได้เป็นดังนี้

1. ความคล้ายคลึงของตัวอักษรที่แตกต่างกันเพียงความยาวของหางที่ต่อออกไปซึ่งสั้น – ยาวไม่เท่ากัน เช่น บ – ป, ผ – ฝ, พ – ฟ เป็นต้น ซึ่งงานวิจัยนี้ ได้พยายามแก้ปัญหาการรู้จำตัวอักษรที่หางสั้น – ยาวในลักษณะนี้ โดยการนำค่าความยาวของแต่ละอินพุตมาช่วยแยกแยะความแตกต่างทั้งความยาวที่มากที่สุด (MaxLength) และความยาวที่น้อยที่สุด (MinLength) ในแต่ละช่วงของสตริงตัวอย่าง และผลปรากฏว่า ค่าที่สามารถแยกแยะได้จริง จะเป็นค่าความยาวมากที่สุดเท่านั้น เนื่องจาก อินพุตอาจจะถูกตัดแบ่งความยาวเป็นช่วงๆ เช่น เมื่อมี Noise แทรกเข้ามาตัดกึ่งกลางสตริง

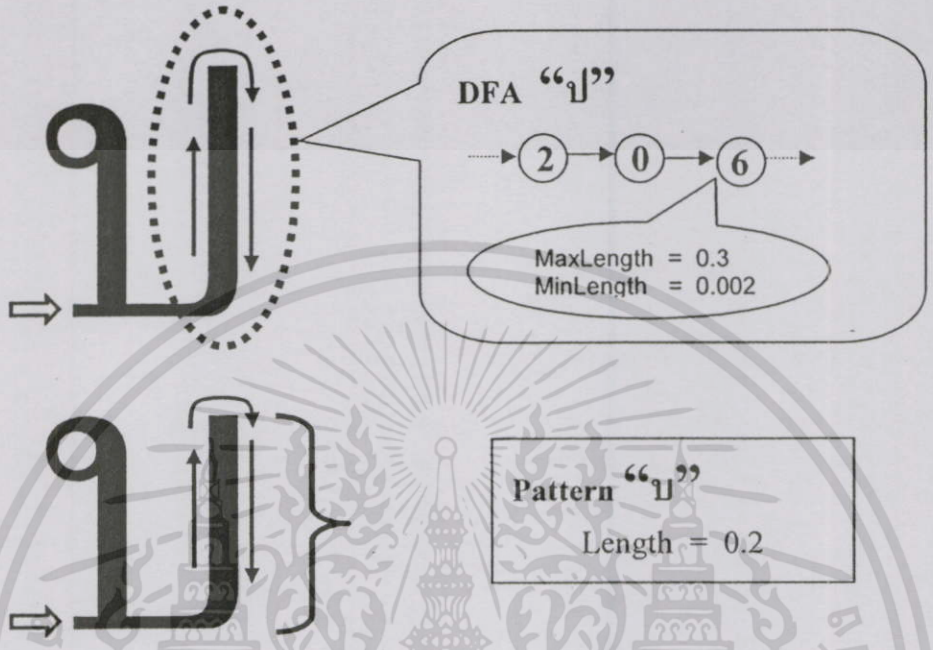
สตริงปกติที่ยาวต่อเนื่องกัน : 22222222222222222222222222222

สตริงที่ถูกตัดทอนความยาวออกไป : 22222222662222222222220222222

ซึ่งทำให้ค่าความยาวที่น้อยที่สุดเกิดความผิดพลาดไปในทันที เนื่องจากไม่สามารถแยกแยะได้ว่า การเกิดอินพุตใหม่ที่แม้เป็น Noise นั้น เป็นเพียง Noise จริง หรือเป็นอินพุตในช่วงถัดไป ดังนั้นค่า MinLength จึงต่ำกว่าที่ควรจะเป็น และไม่สามารถนำมาพิจารณาได้ เพราะคงไม่มีช่วงความยาวใดที่ต่ำกว่าค่า MinLength ที่ถูกแบ่งช่วงความยาวไปแล้ว

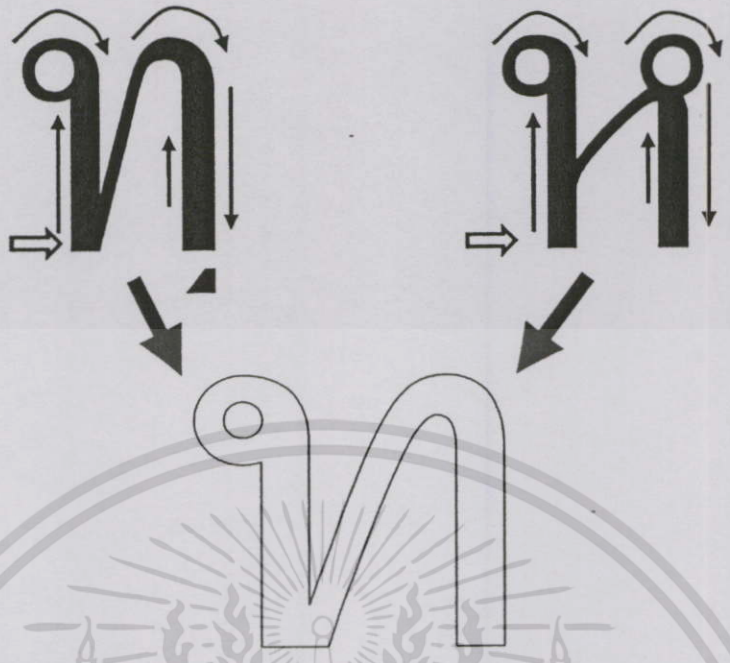
จากตารางที่ 5.3 จะเห็นว่า ตัวอักษรที่แตกต่างกันเพียงหางสั้น – ยาวนั้น ตัวอักษรที่ไม่มีหางหรือหางสั้นกว่า จะมีเปอร์เซ็นต์การรู้จำต่ำกว่า แต่มีเปอร์เซ็นต์ความกำกวมสูง เนื่องจาก สตริงที่เป็นตัวอักษรไม่มีหาง จะไปตกแมทชีนที่ถูกต้อง และแมทชีนของตัวอักษรที่มีหางยาวด้วย เช่น สตริง “บ” จะตกทั้ง DFA “บ” และ “ป” ด้วย เนื่องจาก เมื่อทดสอบสตริง “บ” ที่หางสั้น กับ DFA “ป” นั้น ความยาวของหางย่อมต่ำกว่าค่าความยาวสูงที่สุดของช่วงนั้นๆ (MaxLength) จึงสามารถยอมรับโดย DFA “ป” ได้ โดยที่ค่าความยาวน้อยที่สุด (MinLength) ไม่สามารถนำมาใช้ได้ จึงแยกแยะตัวอักษรที่หางสั้นกว่า “ป” ไม่ได้นั่นเอง รูปที่ 5.3 จะแสดงลักษณะที่สตริง “บ” สามารถไป

ตกที่ DFA "ป" ได้ เนื่องจากค่าความยาวอยู่ในช่วงที่กำหนด แต่หลังจากทำการแบ่งกลุ่มข้อมูลโดยใช้วิธี Multimodal แล้ว ปรากฏว่า สามารถทำให้การใช้ MinLength แบ่งแยกความแตกต่างของสตริงได้ดียิ่งขึ้น และลดปัญหาความกำกวมลงได้มาก



รูปที่ 5.3 แสดงตัวอย่างความคล้ายคลึงของตัวอักษรที่มีหางยาวและหางสั้น

2. ความคล้ายคลึงของตัวอักษรที่มีหัว และไม่มีหัว ทั้งนี้จะนับหัวเป็นทุกๆส่วนมีลักษณะเป็นหัวกลมๆ โดยไม่จำเป็นต้องเป็นส่วนเริ่มต้นตัวอักษร เช่น ก – ถ – ภ, ฆ – ฌ – ญ (ญ จะนับเฉพาะส่วนตัวอักษรที่ต่อเนื่องกัน โดย "ไม้หันอากาศ" ที่อยู่ใต้ ญ จะถือเป็นอีกอักขระหนึ่ง), น – บ, ห – ท เป็นต้น ส่วนหัวของตัวอักษรนี้จะถือว่าเป็นส่วนที่มีความกำกวมค่อนข้างมากของตัวอักษร เพราะเป็นส่วนที่มีความโค้ง ทั้งยังมักจะอยู่ตามมุมของตัวอักษร ที่มีการเปลี่ยนทิศทางในลักษณะเส้นโค้งเช่นกัน จะทำให้ได้ลำดับของสเตทมีได้หลากหลาย จึงไม่มีความแตกต่างกันมากนัก ระหว่างอักษรที่มีหัว และไม่มีหัว รูปที่ 5.4 จะแสดงลักษณะโครงร่างของตัวอักษรที่มีหัว และไม่มีหัว ที่มีความคล้ายคลึงกัน



รูปที่ 5.4 แสดงตัวอย่างความคล้ายคลึงของตัวอักษรที่มีหัวและไม่มีหัว

3. ความคล้ายคลึงของตัวอักษรที่มีรอยหยัก และไม่มีรอยหยัก เช่น ข – ฃ, ค – ฅ, ช – ซ, ด – ฎ, ท – ฑ, ม – ฆ, ฎ – ฏ เป็นต้น ส่วนรอยหยักของตัวอักษรนี้ จะเป็นช่วงสตริงสั้นๆที่นำมาต่อกัน และมีทิศทางไม่แน่นอน ดังนั้น จึงเกิดรูปแบบของสเตทได้หลากหลายมาก และจะมีโอกาสเกิดทรานสิชั่นที่กระโดดข้ามสเตทหนึ่งไปอีกสเตทหนึ่งได้ง่าย อันเกิดจากความพยายามในการใช้สเตทที่มีอยู่เดิมให้ได้มากที่สุด จากตารางที่ 5.4 จะพบว่า ตัวอักษรที่มีความแตกต่างกันที่รอยหยักนั้น สตริงของตัวอักษรที่ไม่มีรอยหยัก จะสามารถไปตก DFA ของตัวอักษรที่มีรอยหยักมากกว่า แสดงให้เห็นว่า ลักษณะของสเตทแมทชีนนั้น มีการข้ามส่วนรอยหยักไปได้ ซึ่งอาจเกิดจากรอยหยักของสตริงต้นแบบที่นำมาสร้างสเตทแมทชีนไม่มีความชัดเจนพอ ทำให้มีการสร้างทรานสิชั่นกระโดดในลักษณะนี้ขึ้น

4. ความคล้ายคลึงของตัวอักษรที่มีส่วนหางเสริมออกมาจากส่วนใดส่วนหนึ่งของตัวอักษร ซึ่งตัวอักษรลักษณะนี้จะแตกต่างกับอักษรที่หางยาวดังที่กล่าวไปแล้ว กล่าวคือ ตัวอักษรตามข้อ 1 จะต่อหางออกไปจากปลายหางเดิม ทำให้รูปแบบสเตทจะเหมือนกันทุกประการ ข้อแตกต่างมีเพียงความยาวของสตริงในช่วงนั้นเท่านั้นที่มีค่าไม่เท่ากัน ในขณะที่ตัวอักษรที่จะพิจารณาในข้อนี้ จะเป็นอักษรที่ต่อส่วนหางเสริมเข้าไปในจุดอื่นๆ ที่ไม่ใช่ส่วนปลาย ตัวอย่างอักษรแบบนี้ เช่น ล – ส, ค – ศ เป็นต้น จากตารางที่ 5.4 พบว่า อักษรที่มีหางเพิ่มเข้ามา สามารถไปตก DFA ของอักษรที่ไม่มีหางได้ด้วย ซึ่งอาจจะเกิดจากตัวสตริงทดสอบเองที่มีหางไม่ชัดเจน จนกลายเป็นอักษรไม่มีหางไป

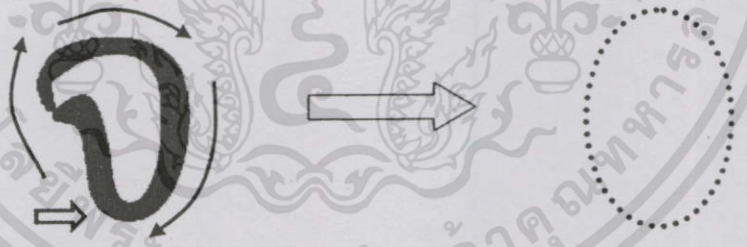
หรืออาจจะเกิดจากความกำกวมของสเตท ณ บริเวณนั้น ซึ่งเป็นส่วนโค้งที่จะมีความกำกวมมากเป็นพิเศษ

5. ตัวอักษรที่มีลำดับในการเขียนคล้ายคลึงกัน เช่น 1 – 7 – สระอา, 5 – ร – ธ – ฐ (สำหรับ ฐ จะพิจารณาเฉพาะอักขระส่วนบนที่เขียนต่อเนื่องกันเท่านั้น), ข – บ – สระอู เป็นต้น ตัวอักษรเหล่านี้จะมีลักษณะการเขียนคล้ายคลึงกัน แตกต่างเล็กน้อยในส่วนทิศที่เอียง หรือส่วนโค้ง ซึ่งเป็นจุดที่สเตทมีความซับซ้อนและหลากหลายอยู่แล้ว จึงมีโอกาสที่จะเกิดความผิดพลาดได้มาก

<u>หมายเหตุ</u> : สำหรับความคล้ายของ “ข – บ – สระอู” นั้น เป็นเพราะการปรับความยาวของสตริงให้เท่ากัน (Normalization) ดังนั้น จึงไม่สามารถแยกแยะตัวอักษรที่มีขนาดไม่เท่ากันได้ แม้เราจะทราบอยู่แล้วว่า “สระอู” มีขนาดเล็กกว่าตัวอักษรมากก็ตาม

6. สระบนและสระล่างเกือบทุกตัวจะมีข้อผิดพลาดในการรู้จำค่อนข้างมาก เนื่องจากอักขระมีขนาดเล็ก ทำให้ลักษณะของสตริงไม่ชัดเจนมากพอจะแยกแยะได้ เช่น สระอิ, สระอี, สระอึ, สระอือ เมื่อนำมาแสกนและแปลงเป็นสตริงนั้น ก็เกือบจะเรียกว่าเหมือนกันมากเลยทีเดียว จึงทำให้การรู้จำผิดพลาดได้ง่าย นับตั้งแต่สตริงที่นำมาสร้างโมเดลตัวอย่าง จนถึงสตริงชุดทดสอบ

7. ตัวอักษรที่เกิดจากความผิดพลาดในการแปลงรูปภาพให้เป็นสตริง ทำให้สตริงขาดหายไปเป็นช่วงใหญ่ๆ เช่น “จ” มีหัวปิดไป และวนเส้นขอบผิดพลาด จนไปคล้ายกับเลขศูนย์ได้ รูปที่ 5.5 แสดงความผิดพลาดในการวนเส้นขอบของอักษร “จ”



รูปที่ 5.5 แสดงตัวอย่างความผิดพลาดในการแปลงรูปภาพให้เป็นสตริง

5.3.2 วิเคราะห์ปัญหาที่พบในการรู้จำสำหรับวิธี HMM

สำหรับวิธี HMM นั้น มีตัวอักษรบางตัวที่มีเปอร์เซ็นต์ในการรู้จำค่อนข้างต่ำ ซึ่งปัญหาที่ทำให้ความถูกต้องในการรู้จำลดลงนั้น ก็มีลักษณะคล้ายคลึงกับปัญหาที่พบในการรู้จำของวิธีที่นำเสนอนั่นเอง ดังที่กล่าวไปในข้างต้น ลักษณะตัวอักษรที่ไม่สามารถแยกแยะได้ มักเกิดจากความคล้ายคลึงของตัวอักษรนั้นๆเองด้วย ซึ่งเป็นลักษณะพิเศษของตัวอักษรในภาษาไทย ซึ่งมีอักษรหลายๆตัวที่มีลักษระการเขียนที่ใกล้เคียงกัน อาจจะมีความแตกต่างกันเพียงเล็กน้อย เช่น รอยหยัก หัวของ

ตัวอักษร หรือหางของตัวอักษร เป็นต้น ซึ่งในจุดที่ทำให้เกิดความสับสนได้ง่ายเหล่านี้ อาจจำเป็นต้องใช้คุณลักษณะอื่นๆมาช่วยแยกแยะความแตกต่างได้ดียิ่งขึ้น

ทั้งนี้ ตัวอักษรที่มีความคล้ายคลึงกันมาก จนทำให้เกิดปัญหาในการรู้จำของวิธี HMM บ่อยๆ ได้แก่

1. ตัวอักษรที่มีหางสั้น – ยาว ไม่เท่ากัน เช่น บ – ป, ผ – ฝ, พ – ฟ เป็นต้น ดังที่กล่าวไปแล้วข้างต้น ตัวอักษรเหล่านี้เหมือนกันทุกประการ แตกต่างเพียงความยาวของหาง ซึ่งวิธี HMM ให้ผลการรู้จำค่อนข้างต่ำ เนื่องจากวิธีนี้ ไม่ได้ใช้คุณสมบัติเรื่องความยาวของสตริงมาช่วยในการรู้จำเลย จึงไม่อาจแยกแยะความแตกต่างระหว่างสตริงของตัวอักษรที่หางสั้นและหางยาวได้ จากตารางที่ 5.4 และ 5.5 เมื่อเปรียบเทียบกับวิธีที่นำเสนอ จะพบว่า แม้ว่าวิธีที่นำเสนอจะพบปัญหาในการรู้จำตัวอักษรลักษณะนี้ก็ตาม แต่เปอร์เซ็นต์การรู้จำยังทำได้ดีกว่าวิธี HMM เนื่องจากมีการใช้ความยาวเป็นฟีเจอร์หนึ่งในการรู้จำนั่นเอง

2. ตัวอักษรที่มีรอยหยัก และไม่มีรอยหยัก เช่น ๐ – ๘, ข – ฃ, ค – ฅ, ช – ซ, ท – ฑ, ฎ – ฏ เป็นต้น สำหรับอักษรที่แตกต่างกันที่รอยหยักนี้ ได้สร้างความสับสนในการรู้จำสำหรับวิธี HMM และวิธีที่นำเสนอเป็นอย่างมาก ซึ่งไม่อาจแยกแยะรอยหยักของตัวอักษรได้

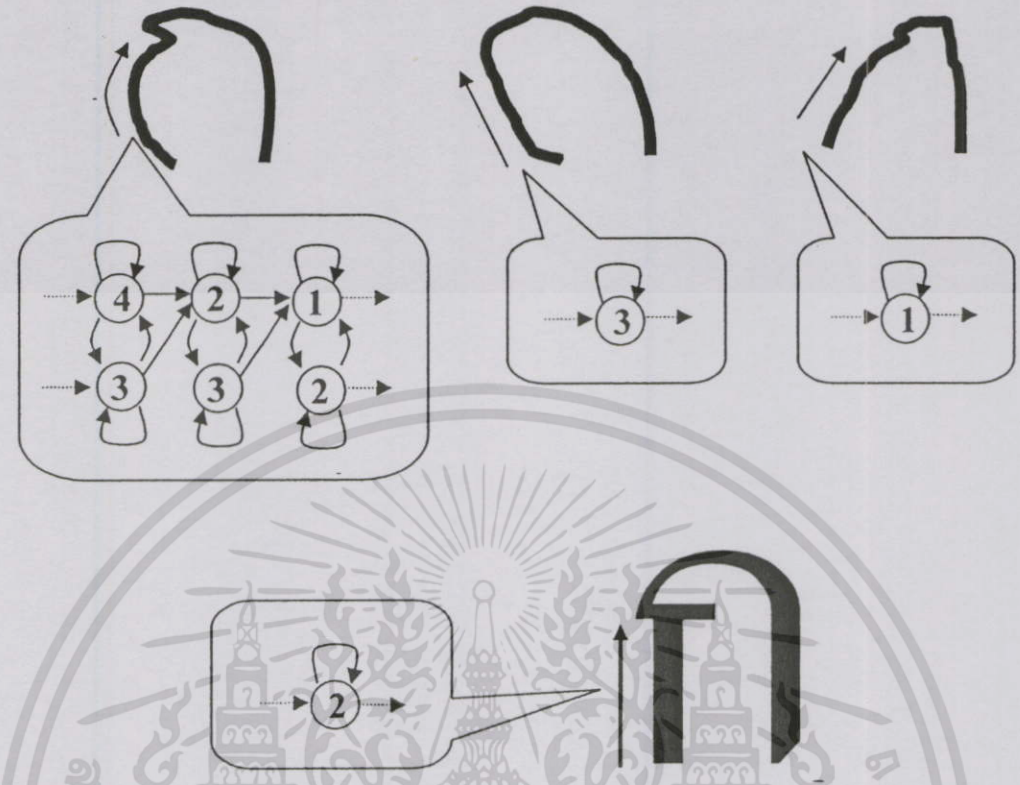
3. ตัวอักษรที่มีหัวและไม่มีหัว เช่น ก – ถ, ท – ห เป็นต้น ซึ่งลักษณะความคล้ายคลึงใกล้เคียงกับตัวอักษรที่แตกต่างกันเฉพาะรอยหยัก

4. ตัวอักษรที่มีหัวกลับด้านกัน เช่น ค – ด, ศ – ต เป็นต้น ซึ่งเกิดความผิดพลาดในการรู้จำของวิธี HMM เป็นอย่างมาก ซึ่งในการรู้จำตัวอักษรลักษณะนี้ วิธีที่นำเสนอทำได้ดีกว่า เนื่องจากใช้ข้อมูลทิศทางของสตริงเป็นหลัก

นอกจากนั้น ยังพบตัวอักษรที่มีความคล้ายคลึงกันอีกหลายประเภท ซึ่งก็คล้ายกับปัญหาที่พบในวิธีที่นำเสนอ ในหัวข้อที่ 5.3.1 ซึ่งตัวอักษรเหล่านี้ มักเป็นตัวอักษรที่มีความผิดพลาดในการรู้จำอยู่เสมอๆ เนื่องจากมีวิธีการเขียนที่คล้ายคลึงมาแต่เดิมอยู่แล้วนั่นเอง

5.3.3 การรู้จำสตริงที่ได้จากลายมือเขียน

วิธีที่ใช้งานวิจัยในวิทยานิพนธ์ฉบับนี้ ได้นำมาทดสอบกับสตริงที่ได้มาจากตัวอักษรตัวพิมพ์แทนที่จะเป็นอักษรที่ได้จากลายมือเขียน เนื่องจาก ลักษณะของตัวอักษรที่เป็นลายมือเขียน จะมีความหลากหลายอย่างมาก ไม่สามารถกำหนดรูปแบบโดยอาศัยทิศเพียงอย่างเดียวได้ ตัวอย่างเช่น การเขียนเส้นตรงในตัวพิมพ์ ทิศทางย่อมเป็นทิศนั้นเพียงอย่างเดียว แต่ในขณะที่ลายมือเขียน ส่วนของเส้นตรงในตัวอักษร อาจจะมีความโค้งงอไปบ้าง มากน้อยตามแต่ลักษณะการเขียนของคนนั้นๆ ซึ่งเมื่อนำสตริงมาสร้างเป็นสเตทแมทชีน การเขียนเส้นโค้งนั้น จะทำให้เกิดลำดับสเตทได้หลายรูปแบบ ยิ่งถ้าความโค้งของเส้นแตกต่างกันมากๆ ยิ่งจะเพิ่มสเตทที่แตกต่างกันมากขึ้น ทำให้ไม่สามารถสร้างลักษณะทั่วไป หรือ Generalization ของส่วนของตัวอักษรนั้นๆได้



รูปที่ 5.6 แสดงตัวอย่างอักษรลายมือเขียนเปรียบเทียบกับตัวพิมพ์

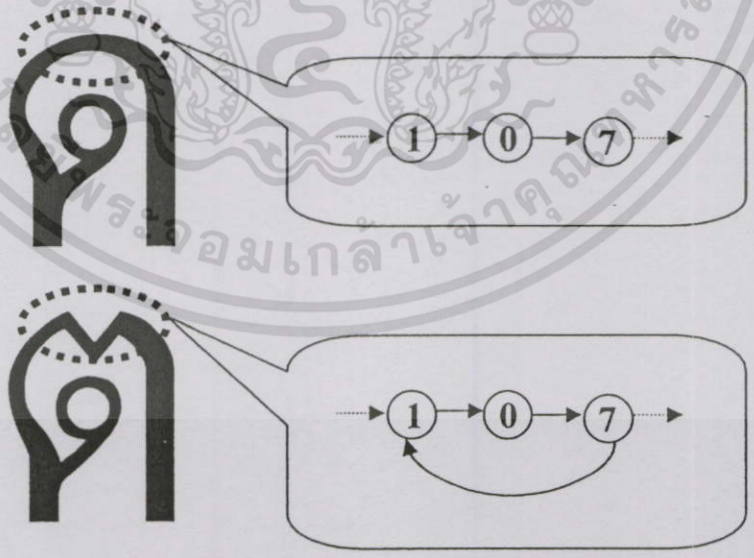
รูปที่ 5.6 จะแสดงตัวอักษรในรูปแบบลายมือเขียนของอักษร "ก" ที่มีความหลากหลายเปรียบเทียบกับอักษรตัวพิมพ์ เมื่อพิจารณาการลากเส้นตรงช่วงแรกตามตำแหน่งในภาพ ถ้าเป็นตัวพิมพ์จะได้สเตทเป็นทิศทางเดียว ในขณะที่ลายมือเขียน จะมีรูปแบบสเตทได้มากมาย ซึ่งทั้งหมดนั้น จะแทนช่วงของเส้นเดียวกันกับตัวพิมพ์ จะเห็นว่า วิธีการสร้างสเตทแมทชีนนั้น ไม่เหมาะกับการรู้จำสตริงที่มีความหลากหลายมากเช่นสตริงที่ได้จากลายมือเขียน ซึ่งควรจะมองเป็นรูปภาพตัวอักษรโดยรวมมากกว่าจะนำมาแปลงให้เป็นสตริง และทำการรู้จำสตริงอีกที และเมื่อทดลองทำการรู้จำโดยใช้ขั้นตอนวิธีแบบ Positive Training อย่างเดียว หรือจะนำวิธี Negative Training มาประกอบด้วยก็ตาม พบว่าได้ผลการรู้จำต่ำมาก เพียงประมาณ 40 – 42% เท่านั้น ส่วนเมื่อนำไปทดสอบโดยใช้วิธี Multimodal ก็พบว่าเกิดการแบ่งกลุ่มข้อมูลเป็นจำนวนมาก จนเกือบจะเรียกได้ว่ามีสตริงเพียงหนึ่งสตริงต่อหนึ่งกลุ่มเลยทีเดียว พิจารณาจากจำนวนสตริงตัวอย่างใน 1 คลาส มีทั้งหมด 200 สตริง แต่กลุ่มสตริงที่ได้มีมากถึงกว่า 100 กลุ่ม แสดงว่าสตริงแต่ละตัวนั้น ไม่มีความคล้ายคลึงกันเลย จึงยากต่อการนำมาสร้างแมทชีนที่จะสรุปลักษณะของสตริงที่มีความหลากหลายเช่นนี้ได้ ซึ่งอาจจะเป็นข้อพิสูจน์ได้ว่า การสร้างสเตทแมทชีนนั้น ไม่เหมาะกับสตริงที่ไม่มี Generalization ที่ดีพอ

5.3.4 การรู้จำโดยในวิธี DFA Learning แบบอื่นๆ

จากงานวิจัยที่เกี่ยวกับการสร้าง DFA ที่ได้อธิบายไว้แล้วในบทที่ 3 นั้น คือ วิธี Trakhtenbrot – Barzdin วิธี Self-Adaptive Greedy Estimate และวิธี Evidence-Driven State Merging ซึ่งเป็นวิธีการสร้าง DFA จากสตริงตัวอย่างทั้งสตริงที่เป็นบวก และสตริงที่เป็นลบ แต่ไม่ได้นำวิธีการเหล่านี้มาเปรียบเทียบผลการทดลองรู้จำกับวิธีที่ใช้ในงานวิจัยนี้ เนื่องจาก

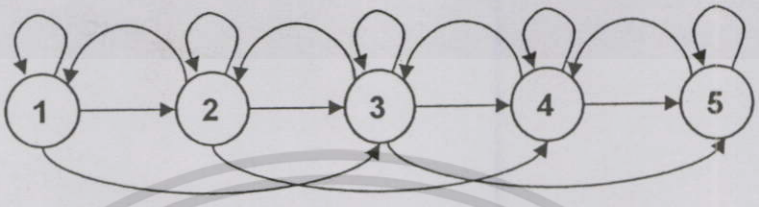
1. วิธี DFA Learning อื่นๆ จะมีลักษณะจำเพาะเจาะจง (Specific) กับสตริงตัวอย่างมาก เนื่องจากเป็นการนำสตริงทุกๆคลาสมาสร้างเป็นทรี และลาเบลทุกๆโหนดในทรีว่าอยู่ในคลาสใด แล้วยุบทรีโดยไม่ให้ลาเบลขัดกัน ลักษณะแมทชีนที่ได้ จะเป็นการรู้จำแบบตรงตัว (Exact Matching) ดังนั้น เมื่อทำการทดสอบกับสตริงไม่ทราบค่า อินพุตแม้เพียงตัวเดียวเพิ่มเข้ามาในสตริง ก็อาจจะทำให้ผลการรู้จำผิดพลาดไปได้ ตัวอย่างการรู้จำที่ผิดพลาดได้อธิบายไว้แล้วในบทที่ 4 (ดูรูปที่ 4.1)

2. วิธี DFA Learning อื่นๆนั้น อนุญาตให้สามารถสร้างลูปย้อนกลับภายในแมทชีนอย่างไรก็ได้ ซึ่งไม่เหมาะสมกับสตริงที่มีลำดับชัดเจนอย่างสตริงที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทย ที่ไม่สามารถอนุญาตให้เกิดลูปได้อย่างอิสระ เนื่องจากลำดับของเชนโค้ดมีการวนลำดับไปในทิศทางเดียว การสร้างลูปอาจจะทำให้เกิดปัญหาสำหรับตัวอักษรที่มีความละเอียดอ่อนได้ เช่น ตัวอักษรที่แตกต่างกันเฉพาะรอยหยัก รูปที่ จะแสดงตัวอย่างอักษร “ด” และ “ต” ซึ่งมีรอยหยักที่ด้านบน หากอนุญาตให้เกิดลูปภายในแมทชีน “ด” ก็อาจจะทำให้ไปเหมือนกับแมทชีน “ต” ได้ ดังรูปที่ 5.7



รูปที่ 5.7 แสดงตัวอย่างการเกิดลูปที่ไม่ถูกต้องของตัวอักษรที่คล้ายคลึงกัน

3. เราไม่สามารถกำหนดโครงสร้างสเตทภายในแมทชีนของวิธีอื่นๆได้ เนื่องจากวิธีเหล่านี้พัฒนาขึ้นมาเพื่อรู้จำสตริงในรูปแบบใดก็ได้ ไม่จำเพาะเจาะจง ซึ่งลักษณะของแมทชีนที่ได้จะแตกต่างกับแมทชีนของสตริงที่มีรูปแบบเฉพาะ ในที่นี้ สตริงจะได้จากตัวอักษรภาษาไทย จึงควรกำหนดโครงสร้างสเตทให้มีความเหมาะสม

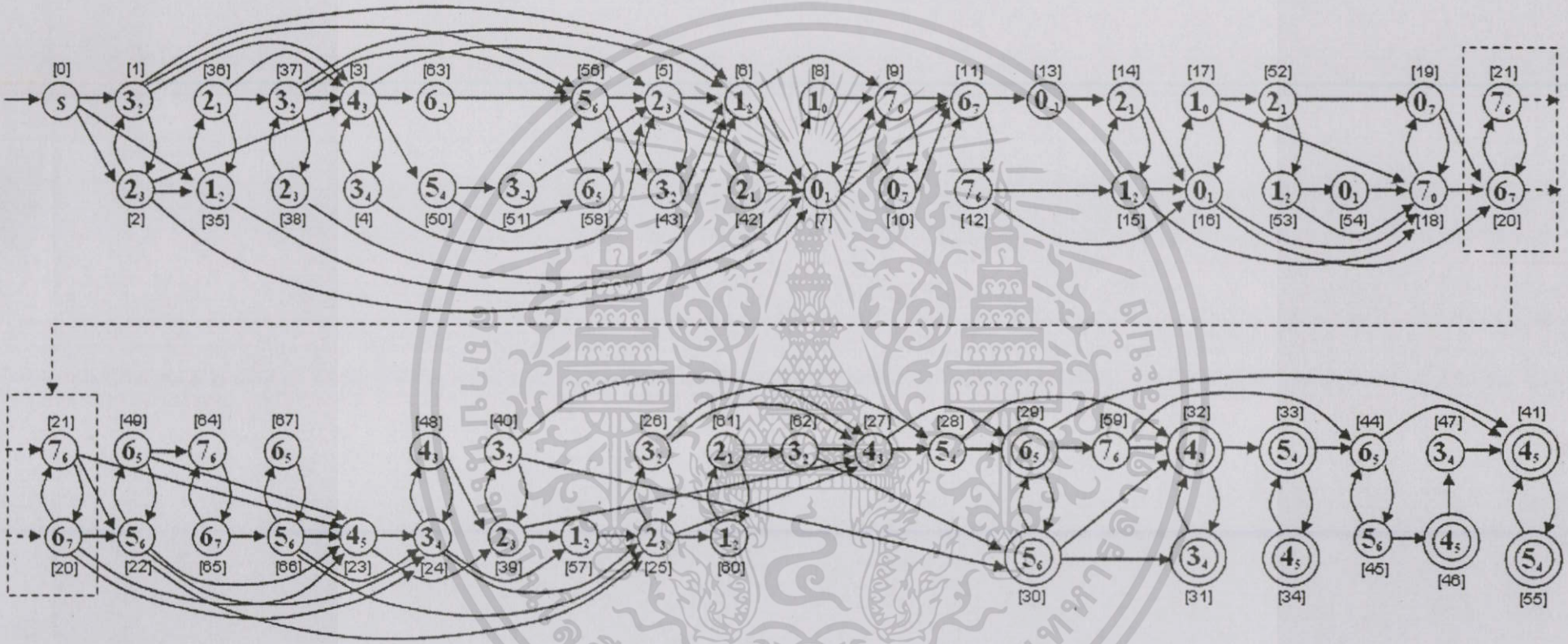


รูปที่ 5.8 แสดงโครงสร้างของวิธี Hidden Markov Model แบบ Left-Right-Left

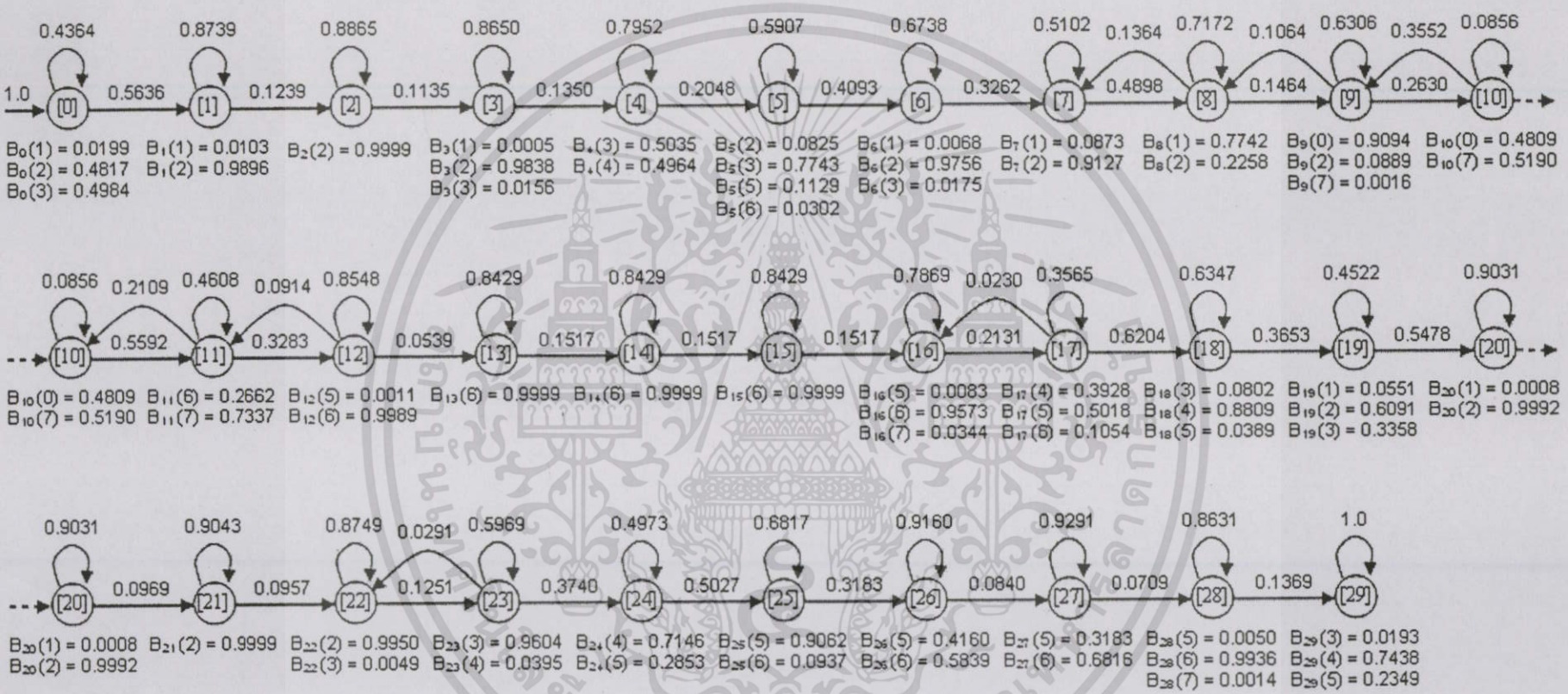
ดังนั้น ผลการทดลองจึงได้ทำการเปรียบเทียบกับวิธี Hidden Markov Modal ซึ่งสามารถเลือกโครงสร้าง (Topology) ให้ใกล้เคียงกับรูปแบบสเตทที่ต้องการได้ และเลือกรูปแบบ Left-Right-Left ที่สามารถมีทรานสิชั่นย้อนกลับได้ไม่เกิน 2 สเตท ดังรูปที่ 5.8 ซึ่งใช้วิธีการตาม [17] ส่วนวิธีอีก 3 วิธีใน DFA Learning นั้น ผลการรู้จำจะต่ำมาก จึงไม่นำมาพิจารณาสำหรับกรณีนี้

เพื่อให้เห็นภาพได้ชัดเจน จึงได้ทำการทดสอบสตริงตัวอย่างจากอักษร "ท" ทั้งหมด 203 สตริง (สามารถดูสตริงตัวอย่างทั้งหมดได้ที่ภาคผนวก ก.) และนำมาเขียนเป็นโมเดลเปรียบเทียบ 3 วิธี ระหว่างวิธีที่นำเสนอ วิธี HMM และวิธี DFA Learning แบบอื่นๆ รูปที่ 5.9 จะแสดงโมเดลที่ได้จากวิธี Positive Training ซึ่งเป็นวิธีที่นำเสนอ โดยจะขอละทรานสิชั่น Loop ที่วนเข้าสเตทตัวเองเพื่อให้ง่ายต่อการเขียน และในกรณีที่เป็น Negative Training รูปร่างโมเดลก็ยังคงเดิม เพื่อแต่เพิ่มสเตทกับดักขึ้นอีก 1 สเตท และเติมทรานสิชั่นจากสเตทอื่นๆเข้าไปยังสเตทกับดักเท่านั้น จะเห็นว่าโมเดลที่ได้ มีทรานสิชั่นที่ไปข้างหน้าเพียงอย่างเดียวเท่านั้น (ยกเว้นทรานสิชั่น Loop และ Loop Pair ที่อนุญาตให้มีทรานสิชั่นย้อนกลับเข้าที่สเตทเดิม หรือที่คู่สเตทของมัน) ดังนั้น จึงเหมาะกับสตริงที่ต้องการลำดับจากเริ่มต้นไปยังสุดท้ายในทิศทางเดียว และสามารถสร้าง Generalization ได้ดี

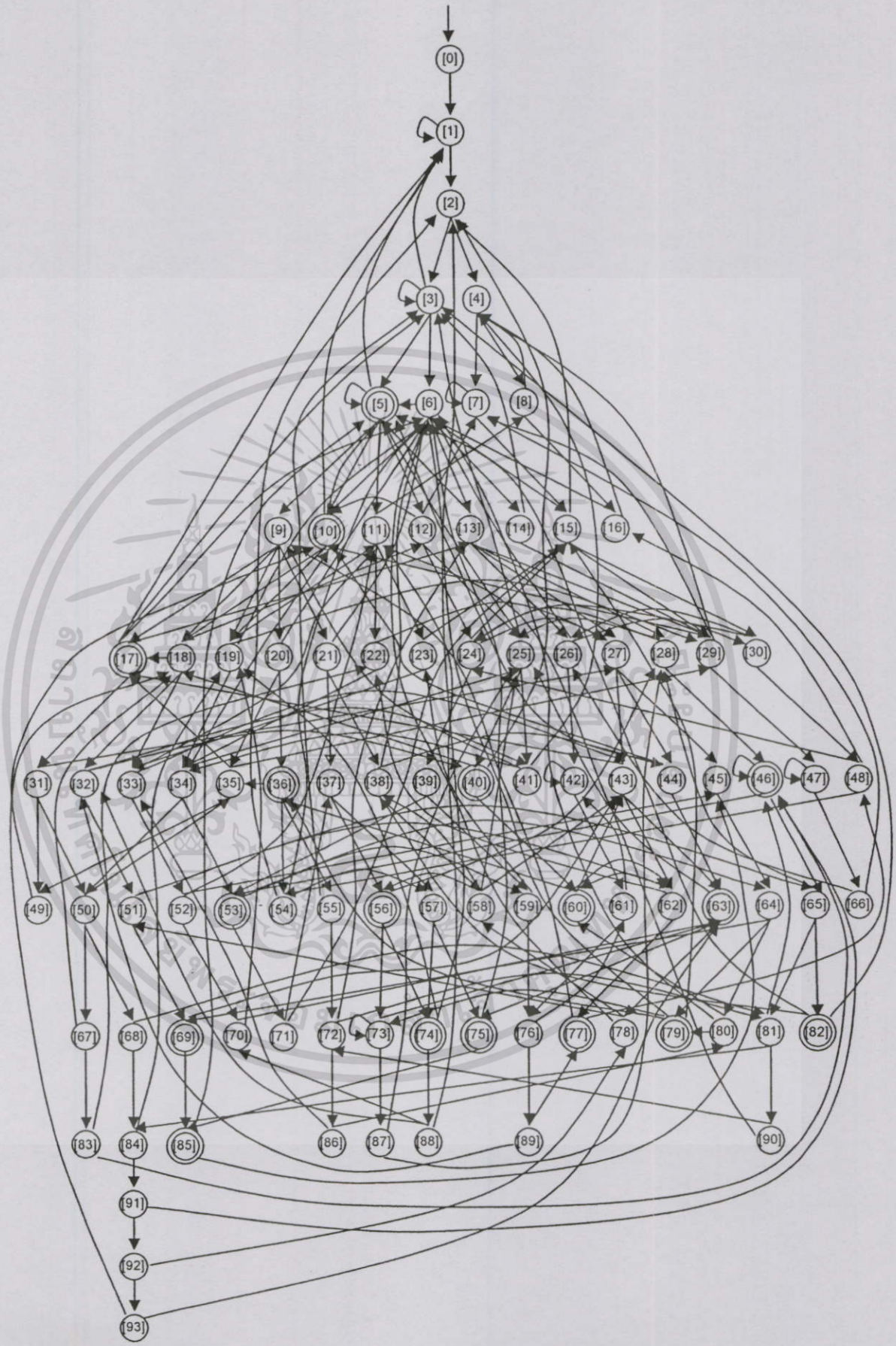
รูปที่ 5.10 จะแสดงโมเดลที่ได้จากสตริงชุดเดียวกันกับโมเดลในรูปที่ 5.9 โดยใช้วิธี HMM แบบ Left-Right-Left Topology และกำหนดให้มีสเตททั้งหมด 30 สเตท ตัวเลขในโหนดจะแสดงลำดับสเตทที่ [0] – [29] ตัวเลขบนทรานสิชั่นจะเป็นค่าความน่าจะเป็นของการเปลี่ยนจากทรานสิชั่นหนึ่งไปอีกทรานสิชั่นหนึ่ง สำหรับสเตทที่ [0] จะมีค่าความน่าจะเป็นในการเป็นสเตทเริ่มต้น ให้เป็น 1.0 (มีสเตทเริ่มต้นเพียงสเตทเดียว คือ สเตท [0]) แต่ละสเตทที่ i จะมีค่าความน่าจะเป็นของการเกิดอินพุต $B_i(a)$ เมื่อ a คือ อินพุต 0 – 7 จะเห็นว่า วิธี HMM ก็สามารถควบคุมรูปแบบโมเดลได้ตามลักษณะ Topology ที่กำหนดไว้ จึงมีลักษณะใกล้เคียงกับสตริงตัวอย่างเช่นกัน แต่ข้อเสียคือ ใช้เวลาในการ Training มากพอสมควร



รูปที่ 5.9 แสดงโมเดลของตัวอักษร "ท" ที่ได้จากวิธี Positive Training



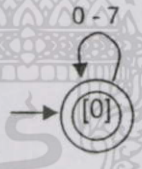
รูปที่ 5.10 แสดงโมเดลของตัวอักษร "ท" ที่ได้จากวิธี HMM



รูปที่ 5.11 แสดงโมเดลของตัวอักษร "ท" ที่ได้จากวิธี EDSM

รูปที่ 5.11 จะเป็นรูปแบบโมเดลที่ได้จากสตริงชุดเดียวกัน โดยใช้วิธีแบบ EDSM หรือ Red-Blue Algorithm [4] ตามที่ได้อธิบายในบทที่ 3 งานวิจัยที่เกี่ยวข้อง ในที่นี้ โหนดที่เป็นวงกลมซ้อนกันสองวง หมายถึงโหนดที่ Accept และโหนดวงกลมธรรมดา หมายถึง Reject ตัวเลขภายในโหนดคือ ลำดับของเสตท นอกจากทรานสิชั่นที่แสดงในภาพ ทุกๆโหนดจะมีทรานสิชั่นย้อนกลับไปที่โหนด [0] ทั้งหมด การทดสอบการสร้างโมเดลโดยใช้วิธีนี้ จะใช้สตริงจากตัวอักษร "ท" เป็นสตริงที่เป็นบวก (Positive Training Set) และสตริงจากตัวอักษรอื่นๆทุกตัว ให้เป็นสตริงที่เป็นลบ (Negative Training Set) ทั้งหมด จากรูป จะเห็นว่า ทรานสิชั่นเกิดขึ้นในทุกๆทิศทาง และเกิดลูปขึ้นจำนวนมากมาย จึงไม่สามารถควบคุมรูปแบบโมเดลได้เลย ซึ่งอาจจะทำให้เกิดปัญหาการรู้จำผิดพลาดได้ดังที่ได้กล่าวไปแล้ว นอกจากนั้น ลักษณะโมเดลยังค่อนข้างจะเป็น Exact Matching กล่าวคือ ถ้ามีอินพุตที่เกินมาเพียงตัวเดียว ก็อาจจะทำให้ผลลัพธ์เปลี่ยนไปก็เป็นได้ (อาจจะจาก Accept กลายเป็น Reject เป็นต้น) จึงทำให้วิธีนี้มีโอกาสเกิด Under Generalization หรือ Over Generaliztion เป็นอย่างมาก

อนึ่ง สำหรับขั้นตอนวิธี EDSM นี้ เราไม่สามารถใช้เฉพาะสตริงตัวอย่างที่เป็นบวกมาสร้างโมเดลเพียงอย่างเดียวได้ เนื่องจากจะทำให้มีเพียงโหนดที่ลาเบลเป็น Accept และโหนดที่ไม่ได้ลาเบล (Unknown) เท่านั้น นั่นหมายความว่า จะไม่มีโหนดที่มีลาเบลขัดแย้งกันเลย และสามารถยุบโหนดทั้งหมดเข้าเป็นโหนดเพียงโหนดเดียวได้ ดังรูปที่ 5.12 หรือเกิด Over Generalization นั่นเอง



รูปที่ 5.12 แสดงการเกิด Over Generalization ของโมเดลที่ได้จากวิธี EDSM

จากการเปรียบเทียบโมเดลของทั้งสามวิธี จึงเห็นได้ว่า วิธี DFA Leraning แบบอื่นๆไม่เหมาะที่จะนำมาทดสอบกับสตริงตัวอย่างที่ได้จากเซนโค้ดของตัวอักษรภาษาไทย จึงขอเปรียบเทียบผลการรู้จำเฉพาะวิธี HMM กับวิธีที่นำเสนอในวิทยานิพนธ์เท่านั้น ซึ่งมีลักษณะที่สามารถควบคุม Generalization ของโมเดลได้ดีกว่าในรูปแบบอื่นๆ

บทที่ 6
สรุปผลการทดลองและข้อเสนอแนะ

6.1 สรุปผลการทดลอง

งานวิจัยที่นำเสนอในวิทยานิพนธ์ฉบับนี้ เป็นวิธีการสร้างโมเดลโดยการเรียนรู้จากกลุ่มสตริงตัวอย่าง ซึ่งโมเดลหรือแบบจำลองที่ได้ จะอยู่ในรูปของสเตทแมทชีน หรือที่เรียกว่าออโตมาตานั่นเอง โดยจะทำการกำหนดกรอบปัญหาให้ชัดเจน และออกแบบโมเดลให้มีความเหมาะสมกับปัญหานั้นๆ ในที่นี้ ผู้วิจัยได้เลือกสตริงตัวอย่างที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทยเป็นตัวตั้งต้นของปัญหา

จากวิธีในงานวิจัย [1] ซึ่งเป็นการสร้างแกรมม่าจากกลุ่มสตริงตัวอย่างเช่นกัน ผู้วิจัยได้ยึดรูปแบบของโมเดลตามวิธีนี้ แต่ได้นำมาสร้างให้เป็น DFA แทนที่จะใช้แกรมม่า และได้ปรับปรุงแก้ไขวิธีเดิม ซึ่งมีจุดอ่อนที่ไม่สามารถแยกแยะตำแหน่งและความยาวของสตริงได้ รวมทั้ง ไม่ได้ใช้สตริงที่เป็นลบ (Negative Training Set) มาช่วยในการสร้างโมเดลเลย จึงไม่สามารถปฏิเสธสตริงที่มาจากคลาสอื่นได้

ในขั้นแรก ผู้วิจัยได้ออกแบบวิธีการสร้างโมเดล โดยเพิ่มข้อมูลตำแหน่งและความยาวสตริงเข้าไปในสเตท เพื่อแก้ปัญหาการเกิดลูปย้อนกลับที่ไม่ถูกต้อง และเพื่อให้แยกแยะอักษรที่มีลักษณะคล้ายคลึงกันได้ดียิ่งขึ้น นอกจากนี้ ยังได้พยายามจะทำให้โมเดลมีความสามารถในการยอมรับสตริงของคลาสนั้นๆ ไปพร้อมๆกับปฏิเสธสตริงที่มาจากคลาสอื่นๆอีกด้วย จึงได้วิธีการสร้างโมเดลโดยแบ่งเป็น 2 ขั้นตอน คือ

1. Positive Training จะสร้างโมเดลจากสตริงตัวอย่างของคลาสนั้นๆ เพื่อให้โมเดลสามารถยอมรับสตริงจากตัวอักษรที่คล้ายคลึงกันได้ โดยการกำหนดรูปแบบสเตทตามลักษณะสตริงเป็น 2 รูปแบบคล้ายกับ [1] คือ Loop ซึ่งแทนสตริงจากอักษรทิศเดียวกัน และ Loop Pair ซึ่งแทนสตริงที่มาจาก 2 ทิศที่อยู่ติดกัน แต่จะเพิ่มข้อมูลตำแหน่งของสตริงและความยาวของสตริงเข้ามา เพื่อไม่ให้เกิดลูปย้อนกลับที่ไม่ต้องการ และนำมาช่วยในการรู้จำต่อไป
2. Negative Training จะนำโมเดลที่ได้จาก Positive Training มาเพิ่มสเตทกับดัก (Trap State) เพื่อกันไม่ให้โมเดลยอมรับสตริงจากคลาสอื่นๆได้ โดยจะนำสตริงจากคลาสอื่นมาเก็บข้อมูล ถ้าเกิดมีทรานสิชั่นใดที่ทำให้สตริงที่เป็นลบมีเส้นทางไปถึงสเตทสุดท้ายได้ ก็จะสร้างทรานสิชั่นไปยังสเตทกับดักแทน

ส่วนขั้นตอนการรู้จำนั้น นอกจากที่ DFA จะยอมรับเมื่อสตริงจบที่สเตทสุดท้าย (Final State) แล้ว ผู้วิจัยได้เพิ่มค่าความผิดพลาดบางอย่าง เพื่อช่วยในการรู้จำด้วย ซึ่งจะได้ใช้ค่าตำแหน่งและ

ความยาวสตริงอย่างเต็มประสิทธิภาพมากขึ้น ทั้งนี้เพื่อแก้ปัญหางานวิจัยเดิม ที่ไม่สามารถแยกแยะความยาวของตัวอักษรได้

จากวิธีการสร้างโมเดลทั้ง 2 ขั้นตอนนี้ ผู้วิจัยได้ทำการทดลองรู้จำกับกลุ่มตัวอย่างสตริงที่มีขนาดเล็กก่อน คือ มีจำนวนสตริงเพียง 5906 ตัวอย่าง จากทั้งหมด 66 คลาส แบ่งเป็น Training Set 4920 สตริง และ Test Set 986 สตริงเท่านั้น ดังที่เห็นในงานวิจัยที่ตีพิมพ์ผ่านมาแล้ว พบว่า การใช้สตริงที่เป็นลบมาสร้างสเตทกับดักหรือขั้นตอน Negative Training นั้น ทำให้ผลการรู้จำดีขึ้นกว่าการใช้ขั้นตอน Positive Training อย่างเดียวพอสมควร ประมาณ 2.64% แม้ว่าจะให้ผลการรู้จำที่ไม่แตกต่างกับวิธี HMM มากนักก็ตาม

แต่เมื่อเพิ่มจำนวนสตริงตัวอย่างให้มีขนาดใหญ่มากขึ้น กลับพบว่า การใช้ขั้นตอน Negative Training ร่วมด้วย กลับไม่ให้ผลแตกต่างมากนักกับวิธีที่ใช้ Positive Training เพียงอย่างเดียว แสดงว่า การใช้ข้อมูลตำแหน่งและความยาวเข้ามาช่วยก็เพียงพอต่อการสร้างโมเดลแล้ว ไม่จำเป็นต้องใช้สตริงที่มาจากคลาสอื่นๆ ซึ่งทำให้ใช้เวลาในการสร้างโมเดลนานออกไปอีก เพราะต้องนำสตริงที่เป็นลบของทุกๆคลาสมาทดสอบ จึงส่งผลให้เวลาที่ใช้ในการสร้างโมเดลนานขึ้นเป็น n^2 จากเดิมที่ใช้เวลาเพียง n สำหรับขั้นตอน Positive Training อย่างเดียว เมื่อ n เป็นจำนวนคลาสทั้งหมด

เพื่อที่จะพัฒนาวิธีการรู้จำให้เหมาะสมยิ่งขึ้น เมื่อพิจารณาที่สตริงตัวอย่างแล้ว พบว่า มีสตริงบางส่วนที่มีลักษณะแตกต่างกัน แต่อยู่ในคลาสเดียวกัน การพยายามจะสร้างโมเดลเพียงโมเดลเดียวสำหรับคลาสนั้นๆ อาจจะทำให้โมเดลเสียรูปไป เนื่องจากลักษณะร่วมของสตริงนั้นไม่มากพอ ผู้วิจัยจึงได้พยายามแยกสตริงออกเป็นกลุ่มๆ และสร้างโมเดลของแต่ละกลุ่มไว้ กลายเป็นมีหลายโมเดลต่อหนึ่งคลาส และเกิดวิธี Multimodal ขึ้น

วิธี Multimodal ให้ผลการรู้จำที่ค่อนข้างสูงสำหรับข้อมูลที่มีขนาดใหญ่ คือ มีค่าความถูกต้องถึง 84.01% และลดความกำกวมของตัวอักษรไปได้มาก คือ เหลือสตริงที่สามารถยอมรับ DFA ได้มากกว่า 1 ตัว เพียง 3.73% เท่านั้น เมื่อเปรียบกับวิธี HMM ซึ่งได้ผลการรู้จำ 72.61% วิธีจากงานวิจัยจึงมีผลดีกว่าวิธี HMM ถึง 11.4% และใช้เวลาในการสร้างโมเดลหรือขั้นตอน Training น้อยกว่ามาก

แต่เมื่อนำข้อมูลเดิมที่มีขนาดเล็กมาทดสอบ โดยการสร้างโมเดลตามวิธี Multimodal แล้ว ผลปรากฏว่า เปอร์เซ็นต์การรู้จำกลับลดต่ำลงมาก และต่ำกว่าวิธี Positive Training ธรรมดา แสดงว่าวิธีการแบ่งกลุ่มข้อมูลนั้น อาจจะไม่เหมาะกับข้อมูลที่มีขนาดเล็ก ควรจะต้องใช้ข้อมูลจำนวนมากเพียงพอ จึงจะสามารถแบ่งกลุ่มตามลักษณะของข้อมูลได้

สรุปได้ว่า สำหรับข้อมูลขนาดใหญ่ การนำค่าตำแหน่งและความยาวมาใช้ในขั้นตอนการสร้างโมเดลและการรู้จำ นั้นพอเพียงแล้ว ไม่จำเป็นต้องใช้สตริงที่เป็นลบมาสร้างสเตทกับดักอีก และการแยกโมเดลออกเป็นกลุ่มย่อยๆ ก็ช่วยทำให้โมเดลชัดเจนขึ้น และทำให้การรู้จำดีขึ้นอีกด้วย แต่ในข้อมูลขนาดเล็ก อาจจะใช้วิธีการสร้างโมเดลเพียงโมเดลเดียว โดยอาศัยข้อมูลตำแหน่งและความ

ยาว รวมทั้งการใช้ Noise ในสตริงที่เป็นลบมาสร้างสเตทกับดัก ก็ส่งผลต่อการรู้จำได้ดีกว่า การแบ่งข้อมูลเป็นกลุ่มย่อยๆ เนื่องจากจำนวนสตริงตัวอย่างมีน้อย และไม่พอเพียงต่อการจัดกลุ่ม

6.2 ข้อเสนอแนะ

งานวิจัยนี้ได้ทำการสร้างโมเดลจากกลุ่มสตริงตัวอย่าง ที่ได้จากตัวอักษรตัวพิมพ์ภาษาไทย ในรูปแบบ DFA และจากปัญหาต่างๆที่พบในการวิจัย รวมถึงแนวคิดที่ยังสามารถนำไปต่อยอดได้อีก ผู้วิจัยมีข้อเสนอแนะ ดังนี้

แนวคิดการใช้สตริงที่เป็นลบมาร่วมในการสร้างโมเดลนั้น น่าจะทำได้ หากมีวิธีการนำสตริงมาใช้ได้อย่างมีประสิทธิภาพ กล่าวคือ ต้องสามารถกันสตริงอื่นที่ไม่เกี่ยวข้องไม่ให้ไปตกที่สเตทสุดท้ายได้จริง ทั้งนี้ การใช้สตริงที่เป็นลบ จำเป็นต้องคำนึงถึงเวลาที่ใช้ในการ Training ด้วย เพราะสตริงที่เป็นลบนั้น มีจำนวนมากกว่าสตริงที่เป็นบวกมาก ทำให้เวลาที่ใช้ก็ยิ่งเพิ่มเป็นทวีคูณ การเพิ่มเวลาก็ควรจะคุ้มค่ากับประสิทธิภาพในการป้องกันสตริงที่ไม่ต้องการด้วย

สตริงที่นำมาใช้เป็นกลุ่มตัวอย่าง ควรจะมีความคล้ายคลึงกัน ดังจะเห็นว่า เมื่อทำการแยกโมเดลออกเป็นหลายๆโมเดลต่อ 1 คลาส ทำให้ผลการรู้จำดีขึ้นอย่างเห็นได้ชัด นอกจากนั้น การขจัด Noise ออกไปจากสตริงก็น่าจะช่วยทำให้สามารถสร้างโมเดลได้ดีขึ้น รวมทั้งใช้ข้อมูลตำแหน่งและความยาวได้อย่างมีประสิทธิภาพ ดังที่อธิบายไว้ในหัวข้อ 5.3.1 การมี Noise ภายในสตริง ทำให้ไม่สามารถใช้ค่า MinLength มาพิจารณาแยกแยะอักษรที่มีหางสั้น – ยาวได้ หากมี Noise น้อยลงก็จะช่วยแก้ปัญหานี้ไปได้ ทั้งยังทำให้โมเดลมีลักษณะทั่วไป (Generalization) มากขึ้น เพราะไม่ต้องสร้างสเตทหรือทรานสิชั่นที่ไม่จำเป็นขึ้นมา

วิธี Multimodal ยังสามารถคิดค้นวิธีอื่นๆในการแยกโมเดลได้อีก ในงานวิจัยนี้ ได้ใช้จำนวนสเตทที่เพิ่มขึ้นในการสร้างโมเดล มาเป็นตัวตัดสินว่าสตริงอยู่ในกลุ่มเดียวกันหรือไม่ ซึ่งในความเป็นจริง การแบ่งกลุ่มสตริงที่มีลักษณะคล้ายคลึงกัน สามารถทำได้หลายวิธี นอกจากนั้น การจัดระเบียบสตริงก่อนการสร้างโมเดลก็มีความสำคัญ เพราะลักษณะโมเดลที่ได้ จะขึ้นอยู่กับลำดับสตริงที่นำมาเรียนรู้เพื่อสร้างโมเดล ดังนั้น การนำสตริงที่คล้ายกัน มาเริ่มต้นสร้างโมเดลก่อน แล้วค่อยๆขยายโมเดลออกไปทีละนิด ย่อมจะทำให้ได้โมเดลที่มีลักษณะดีกว่าสุ่มเลือกสตริงใดๆมาสร้าง

วิธีที่ใช้ในงานวิจัยนี้ได้เน้นที่วิธีการนำสตริงมาเรียนรู้เพื่อการสร้างโมเดล โดยกำหนดกรอบปัญหาเอาไว้ก่อน เพื่อออกแบบโมเดลให้เหมาะสม จึงอาจนำแนวคิดไปใช้กับสตริงในรูปแบบอื่นๆได้ สำหรับการรู้จำลายมือเขียนภาษาไทยนั้น ผู้วิจัยได้ชี้แจงถึงความไม่เหมาะสมไว้แล้ว ในหัวข้อที่ 5.3.3 หากต้องการรู้จำลายมือเขียน ควรจะใช้ฟีเจอร์อื่นๆในการรู้จำมากกว่า หรือมองเป็นรูปภาพมากกว่าสตริง

เอกสารอ้างอิง

[1] Kruatrachue B., Polsuntikul P. and Siriboon K. "Dynamic Construction of Context Free Grammar from Sample for On-line Thai Handwriting Recognition." **AISTA 2004: International Conference, Luxembourg,** Nov. 2004.

[2] Trakhtenbrot B. and Barzdin Ya. **Finite Automata: Behavior and Synthesis.** Amsterdam : North Holland Publishing Company. 1973.

[3] Juillé H. and Pollack J. B. "A Sampling-based Heuristic for Tree Search Applied to Grammar Induction." **Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98), Madison, Wisconsin, USA,** 1998. pp. 26-30.

[4] Lang K. J., Pearlmutter B. A. and Price R. A. "Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm." **Lecture Notes in Computer Science,** 1433, 1998. pp. 1-12.

[5] Carrasco R. C. and Oncina J. "Learning Stochastic Regular Grammars by Means of a State Merging Method." **The 2nd Intl. Collo. on Grammatical Inference and Applications,** 1994. pp. 139-152.

[6] Rabiner L. R. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." **Proceeding of the IEEE,** vol. 77, 1989. pp. 257-286.

[7] Hopcroft J. E., Motwani R. and Ullman J. D. **Introduction to Automata Theory, Languages, and Computation.** 2nd ED. Addison-Wesley. 2001.

[8] Sipser M. **Introduction to the Theory of Computation.** International Thomson Publishing. 1996.

[9] Paz A. **Introduction to probabilistic automata (Computer science and applied mathematics).** Orlando, FL : Academic Press. 1971.

[10] Ron D., Singer Y. and Tishby N. "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length." **Machine Learning,** 25, 1996. pp. 117-149.

[11] Angluin D. and Smith C. H. "Inductive Inference: Theory and Methods." **ACM Computing Surveys (CSUR),** vol. 15, no. 3, Sept. 1983. pp. 237-269.

[12] Gold E. M. "Complexity of Automaton Identification from Given Data." **Information and Control,** vol. 37, no.3, 1978. pp. 302-320.

[13] Cicchello O. and Kremer S. C. "Inducing Grammars from Sparse Data Sets: A Survey of Algorithms and Results." **Journal of Machine Learning Research 4**, 2003. pp. 603-632.

[14] Grinstead C. and Snell J. L. **Introduction to Probability.** 2nd ED. Providence, RI : American Mathematical Society. 1997.

[15] Stolcke A. "An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities." **Computational Linguistics,** vol. 21, no. 2, June 1995. pp. 165-201.

[16] Chen S. F. "Bayesian Grammar Induction for Language Modeling." **Proceedings of the 33rd annual meeting on Association for Computational Linguistics, Cambridge, Massachusetts,** June 1995. pp. 228-235.

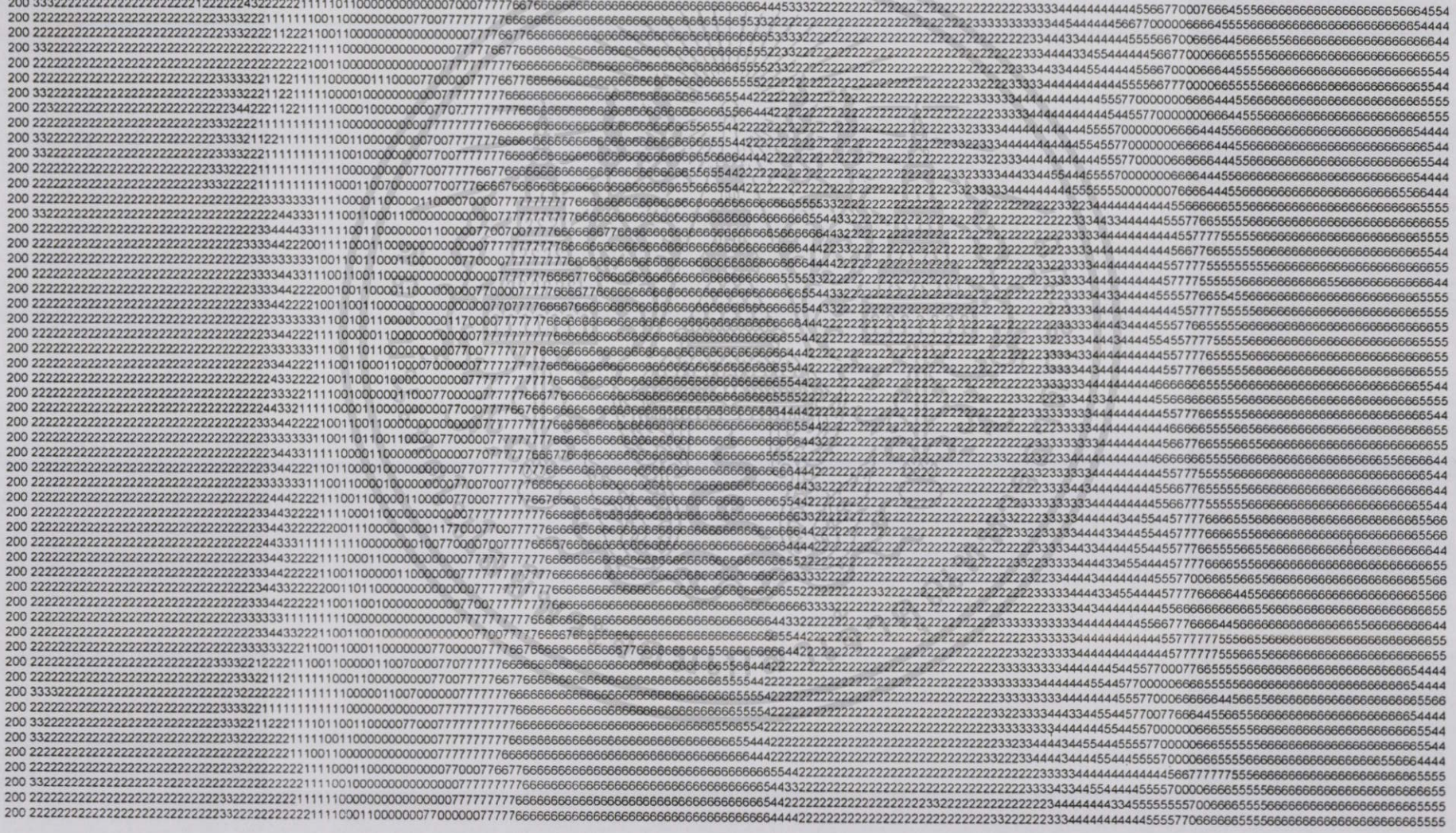
[17] Siriboon K., Jirayusakul A. and Kruatrachue B. "HMM Topology Selection for On-line Thai Handwritten Recognition." **Proceeding of the first International Symposium on Cyber Worlds,** 2002.

ภาคผนวก

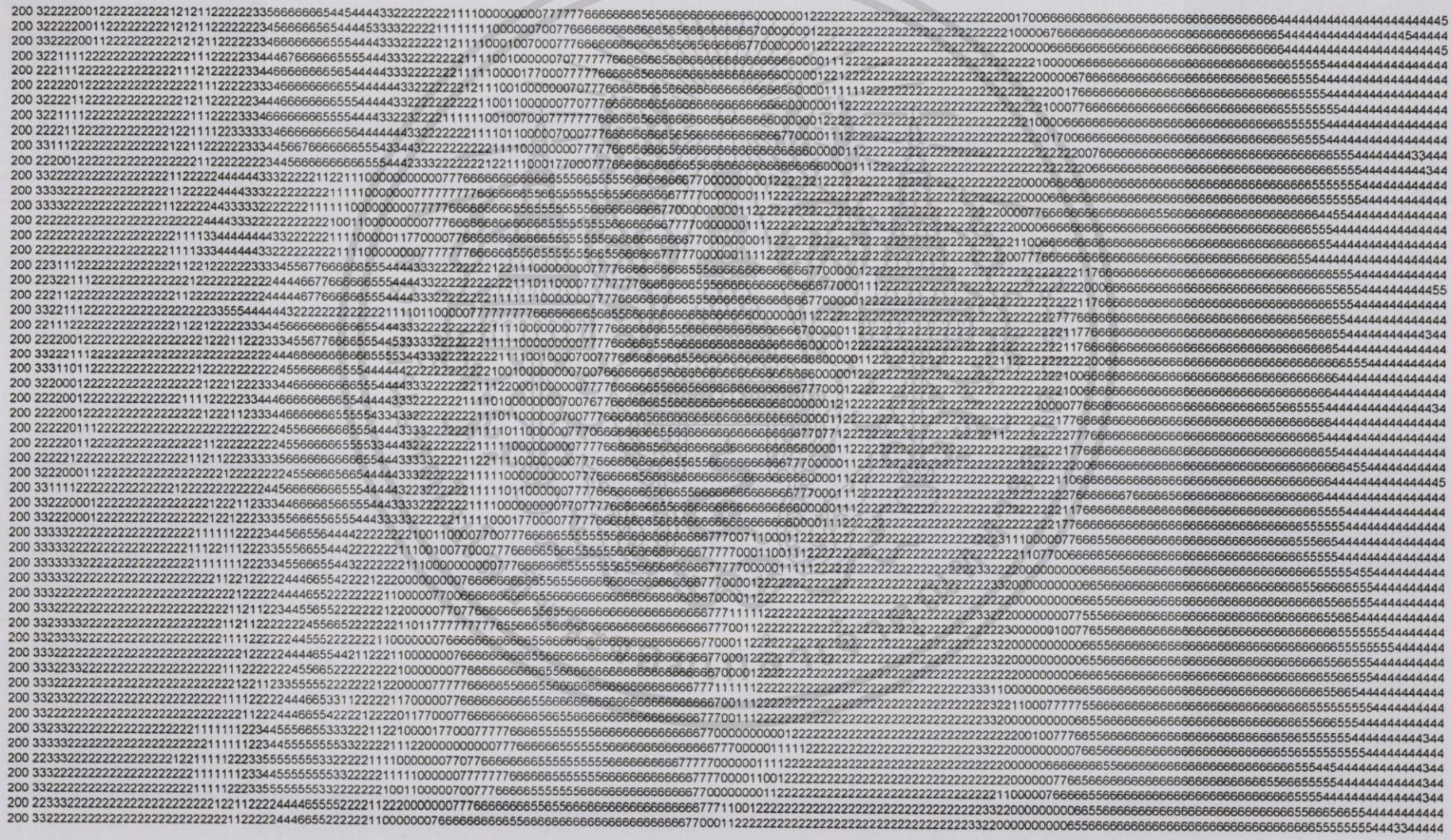
ภาคผนวก ก.

ตัวอย่างสตริงที่ใช้ในการทดสอบ

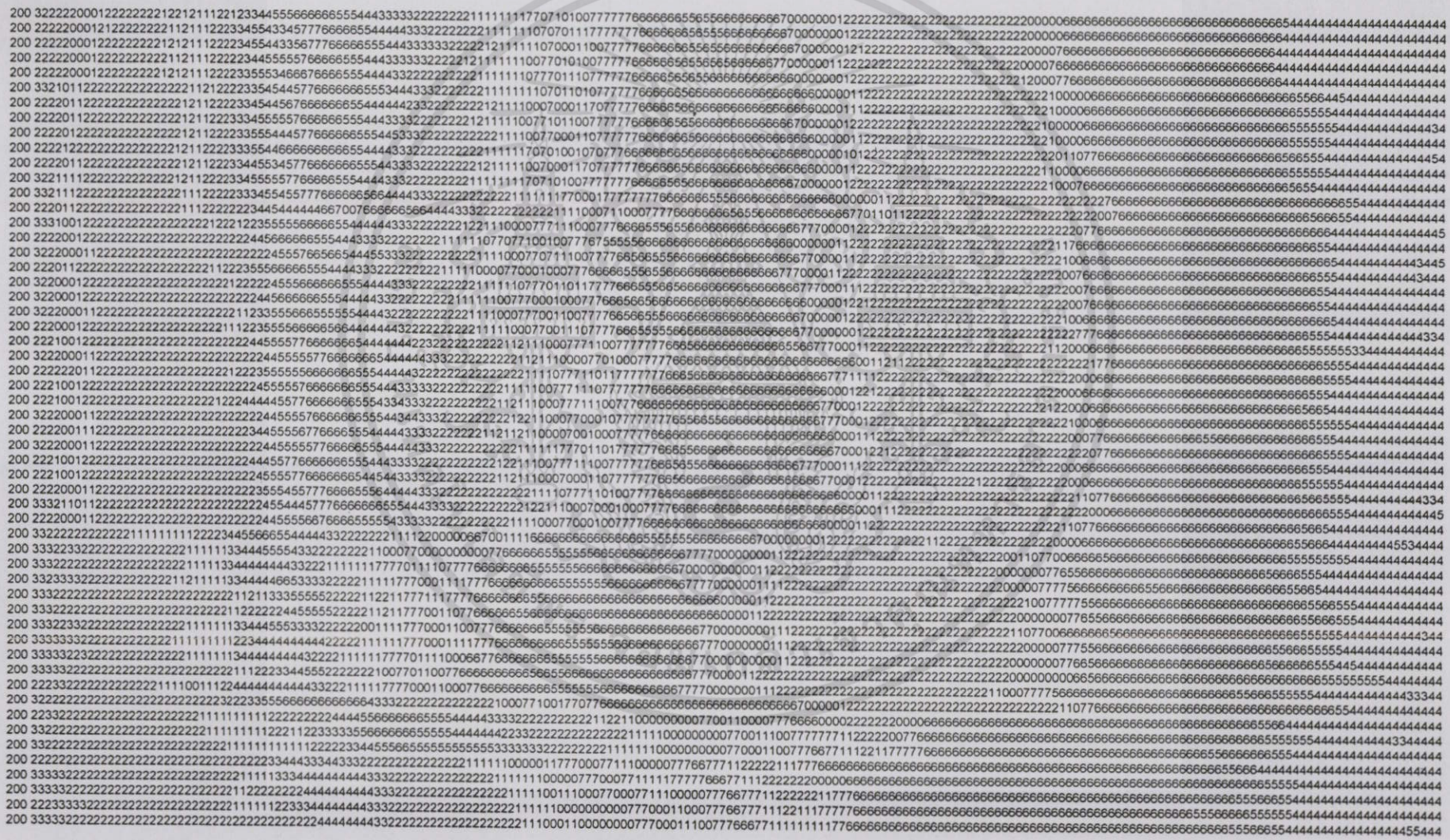
ตัวอย่างสตริงของอักษร “ก”



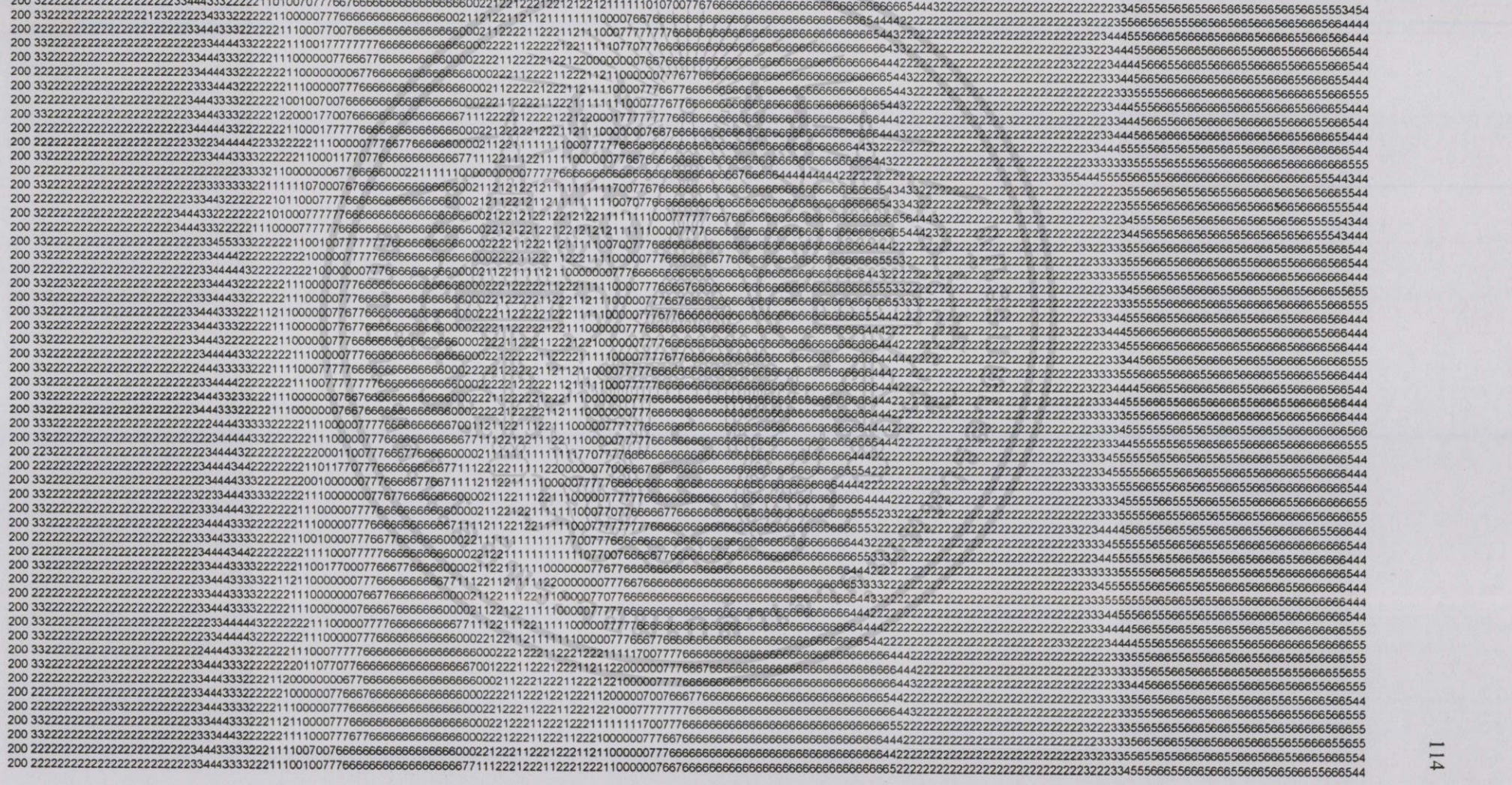
ตัวอย่างสตริงของอักษร “ข”



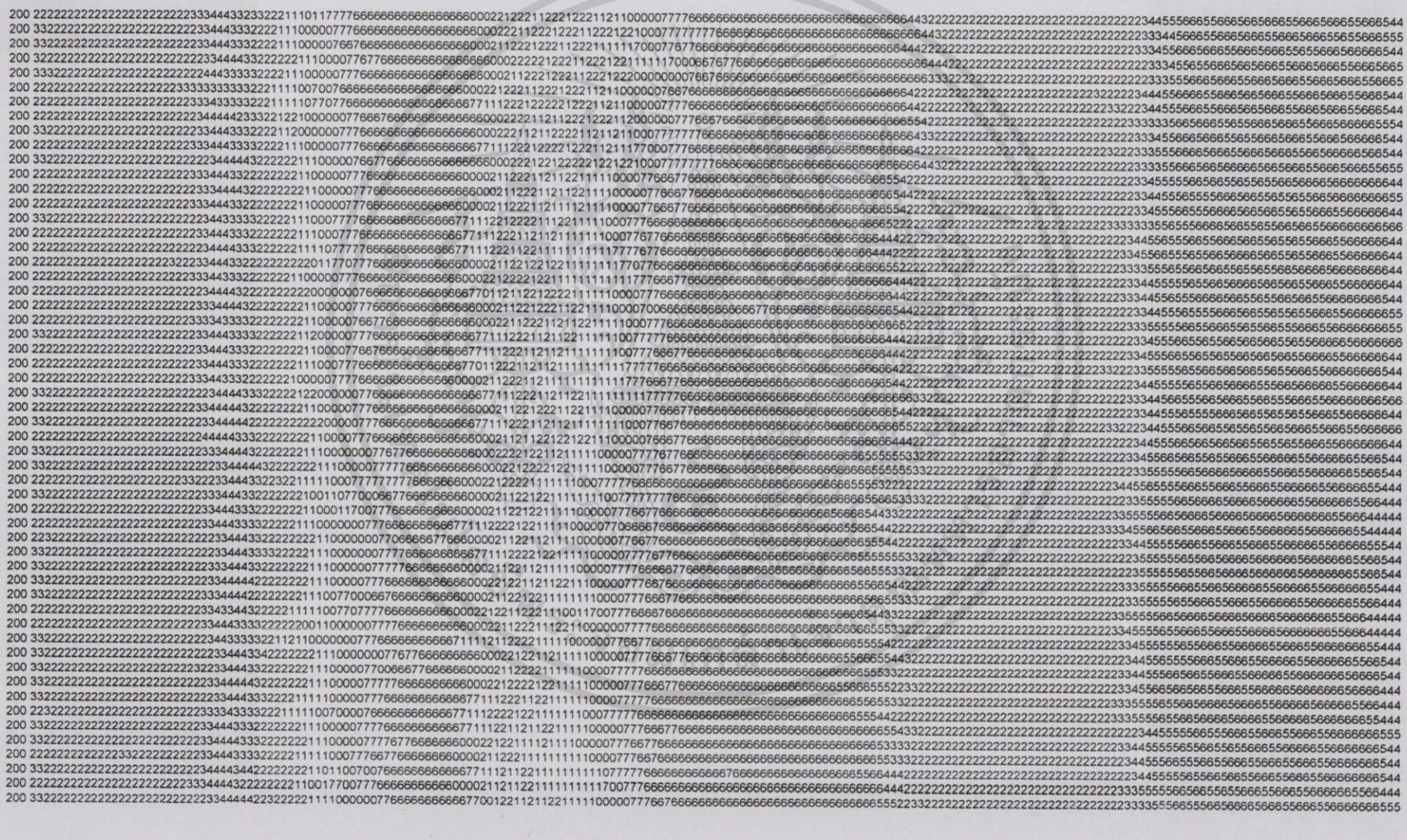
ตัวอย่างสตริงของอักษร “ฃ”



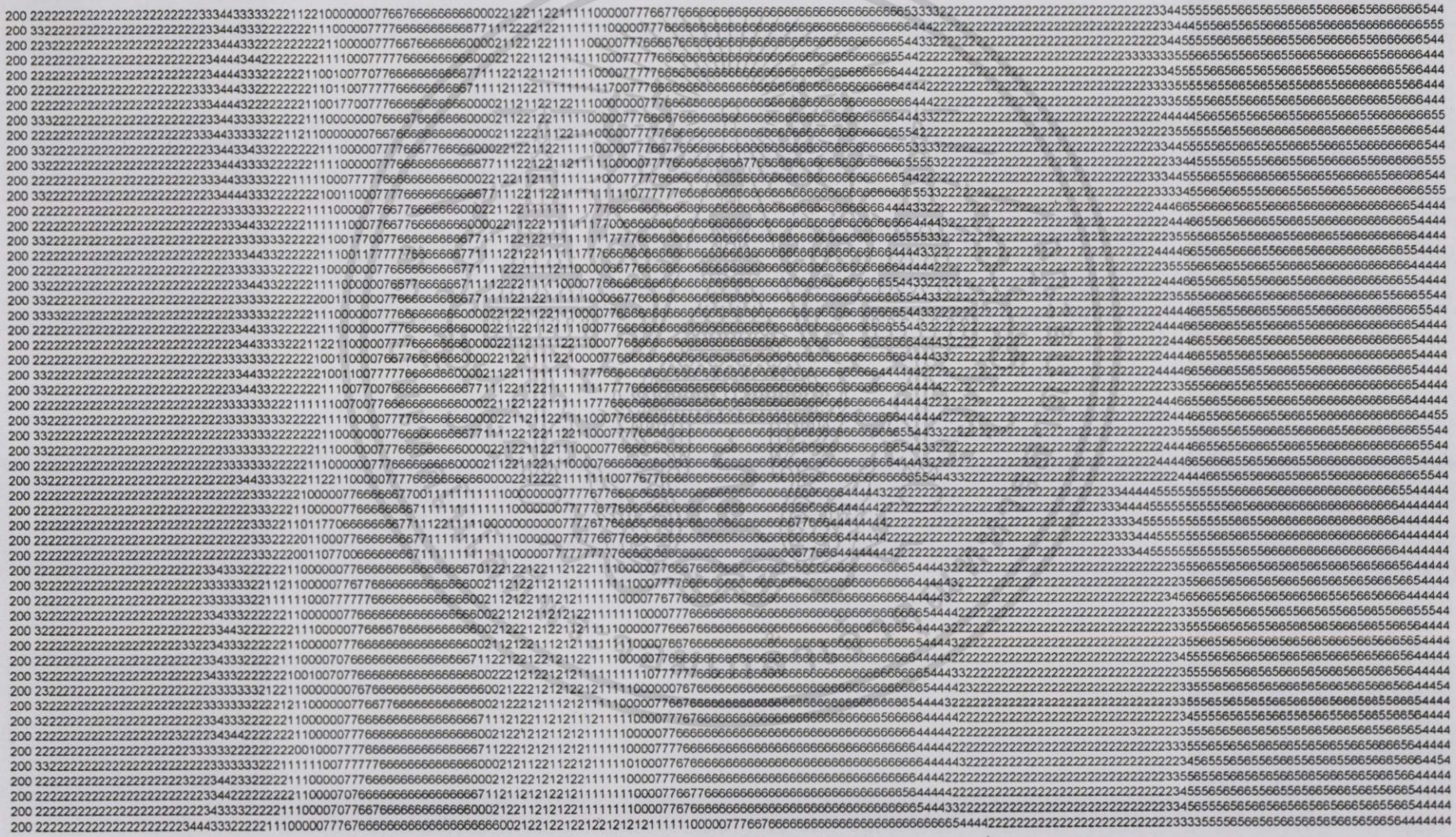
สตริงทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล



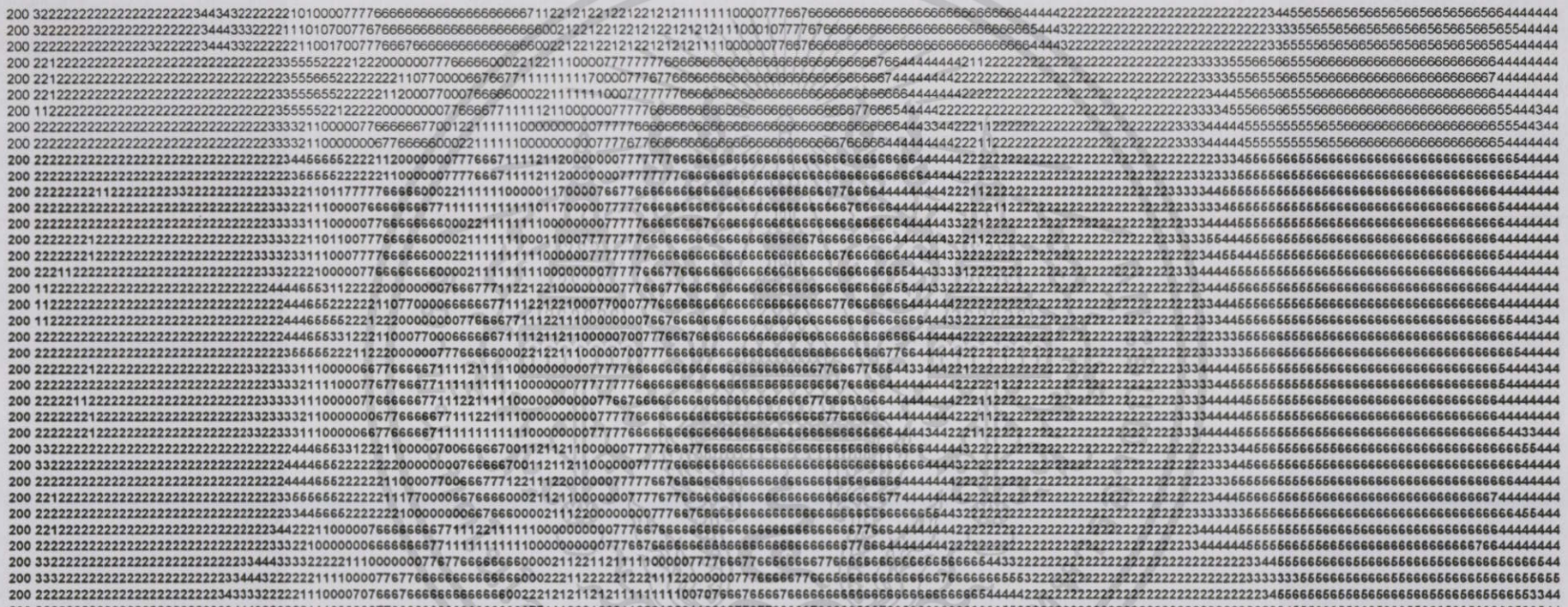
สตริงทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)



สตริงทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)



สตริงทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)



ตัวอย่างสตริงที่เป็นลบทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล

ASCII	Sample String
048	3332223332222333222222222222222222222222222222222222221111111111111111111111111100007770007777777777776667777666666666677766666666666666666666665555666666666666555566665555555555555555544444444444555333333444 33333322233332223332222222222222222222222222222222222222111122211122211111111100000000000000000777777777777666677766666666677766666666666666666666666555566666666666655556666555555555555555544444444444444433344
049	33311111112222222222222222222222222222222222222222221113333333444555222222220001111111111111111111777666666666666666666666666666666666666666666666666666666666666666666666777777744444445554444444444444 1111111112222222222222222222222222222222222222222222223334445555555222222211111111111111111111111777666666666666666666666666666666666666666666666666666666666666666666677777744444444444444444444444
050	2222221111111220011220011122112222222223333334444445555557777666644444422222222211111100000000000007700777766666666665555555555555555555445555777700000000000001111222006666666666544444444444444444444444 22222221111112200111111211112222222222233344444444555556777766666655433332222222222111110000000000000777777776666666666555555555554455555555577000000000000000111222006666666666664444444444444444444444
051	33333332222221100777666655777700000001111111111122223333333333334444222000011111222222233334444444444555666666644442222222111100001177700000077777666666666665555577777766666666666665555544554444444444444 33333222222222100007766655677770000000111111112222333333333344444332000000100111222222223333444444444455666666665544332222222111110000000000007777777766666665556555577777777766666666555555555554444444444444
052	332222222222222222222222233344444444444444444444444444444442222222222111111122111111112211111111221111111100000006666666666666666666666666666666666666677711000066666665544335555666666666666666666664455444 33222222222222223333444444444444444444444444444222222222112111122111112112211122111122111122111121111121100000066666666666666666666666666666666666666677000000006666666554434455566666666666666666655544444
053	33333322200776677000000000000011122222222222223333444444444444455566644422222222222222222222001100077001100000000077664444444444444444455556666666000000000000000777777777666666666666655555555444444444444 33322222222222000007766665557707700000111111222222222223333333344444655544422222222222112222222222220007700000000000110776655544444444444444455666666667700000000000077777776666666666666555555555444444444444444
054	333232232332222222222222222222222222222211221111111111110000117001700707777777766765544444332233334444444544555555556665666666656771111000000000000000777777777766666666666666666655555555554444444444434334 3333333223322222222222222222222222222222222221111111111111110000000000007777777666655553344222222233444444455555555555666666677700000000000000000000776677666666666666666666555555555555444444444444444
055	22111221112222222111222221111122211111221111122333444444444444444444444455555442222222222112220000077001117700000000000001177000666666666665556665556665556665556666666666666666666666677744444444444444 222222222222222111222222221112211222112221122111221111122333444444444444444444444444443332211100770000000011770001177700000000000000066666566665555566655566655566665555666666665556666666666666664444444
056	33333222333222222222222221111111111333332223332222222222111111001110000000011777000117770000077700000777666666666666666555555566777777777777666666666666655555555555555554444444444444444444444444444444433444 2223322222222222222222222111111111333222333222222222222222222111110001110000011100777111770000007770077777766666666666666666555666666666677766666666666666666555555555444555444444444444444444444444443333444
057	332233222211777766770000000000000000001122112211222233555544444444444444444444333322332222222222222222222222111111111110000000000000000077777777776677666666667766666655566666666555666555555555444444444444444443344 3333222221100000006666667770000111111111111122222233444554444444444443333333332222222222222222222222111111111111100000000077000777777667766677666666666666666666666666555666666555566665555444444444444444444444
161	333222222222222222222222122222243222222111110110000000000000700077777667666666666666666666666666666666666444533322222222222222222222222222222223333344444444444455667700076664555666666666666666666666664554 2222222222222222222222222333322211111110011000000000077007777777777666666666666666666666666666555666553322222222222222222222222222222333333333344544444455667700000666455556666666666666666666666654444
162	3222220012222222222121211222222335666666665445444433222222221111000000000777777666666666556666666666666000000012222222222222222222222222222001700666666666666666666666666666666666444444444444444444444445 32222200112222222222121211222222234566666665544445333322222111111100000070077666666666666666666666666667000000012222222222222222222222222222221000067666666666666666666666666666666665444444444444444444544444
163	32222200012222222221221211122123344555666666655544433333222222221111111770710100777777666666665555666666666667000000012222222222222222222222222220000066666666666666666666666666666666544444444444444444444444 2222200012122222222211211122233455433457776666665544444333222222221111110707011177777776666666655556666666666670000000122222222222222222222222222220000066666666666666666666666666666666644444444444444444444444
164	332222222222322223322222222222222221111100110000000077007777776666666666666666666666666666666666644222222222222222222222222222222332222333344444444444555555666666660000221100000776666666644555644556666666666655 22222222222222222222222222222222222211111000000017700077777776666666666666666666666666666666666544332222222222222222222222222222222333334444445555566666666660012211000776666666666455333555556665556666666666544
165	333222322222222222222222221111111110770711000177077776666666666666666666676666666666533444422212222222222222222222222222333445544433355555666666666666666670112222221222111000076654456666666666666666556444444 3333222222222222222222222222211211100000070001000000777766666766666666666666666666666644444443221222222222222222222222222233345554333455555566666666666666667677122222212221121000770666455566666666666666555444
166	33322222111101111211222222222224444444445777706666665553444422222222222111110071100077776666666666665557777077777771122222222222222222222222220666666666666666666666666666666666644333333333445566666665544444 333222222211110122122112122334555556666666665444443332222222122111100771100007777676666655566666700070777077712222222222222222222222222200007666666666666666666666666666665553443233333334444466665665554444
167	3322233222332222223333444334442222111007777777777777766677777777777711222222222222222222222222222233344444443322222222222222211100000000077777666666666666666666666666666666666666666666666666666655554444444444 333332233322333223333333333334442222211007777777777777776667766777771112222222222222222222222222222223344444333332222222222222111100000000000777667766666666666666666666666666666666666666666666655566555555544444
168	33222223322222223333344443223322222221110000007776677667766666776660000222222222222222222222222222333444444444444445555544332222201111000011000000000077777777777777666666666655666666666666666666666665555334445 5 3322222223322222233445543333222222222211000000776676677666667766666600022222222222222212222222223333333344444444455445555533332210011001100000000000000000000777776677666666666666666666666666556666666666655444444444
169	3322222222221100007776677666666666771111110011111222222222222333333333334444444445555555331111001110000000000000777777777666666666666667777777666666666666655444433222222222233445555555555665544422222222222223334444 322222222222233433222222222110000077776666666666666600021111101112222222223323333344344444445544544555333211001101110000000000000070007766777666676666666666707777666666666665544444443333222222233555555555655544444

ตัวอย่างสตริงที่เป็นลบทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)

ASCII	Sample String
170	22222001112222222222222222222222222333565544444444433222222222222221111111111000000077700777770011111111111111177665555555555555567777766666776666666666666666666666666666666666666444444444444444444444444 22222200112222222222221122222222222222222224456666666666555544444433332222222221111111001100770000777777001100001111111117766666445555556677667766666666666666666666666666666555554444444444444444444444
171	33322222222222222222111111334444445642222222221111077770111117776666666665555555666666666666777700000011122222222222222222222322221111222210070066555555567666676666666666666666666555555544444444444344 33222222222222222222110112334444444444222221110007700010000766666666555555555555666666667776770000000011222222222222222222222232211122211000007766555666677666666666666666666666666555555444444444444344
172	22222222222222223332222211100010070000077666666666666777770771222222222222222112200666666666666666666666666666444233356564444444442222220012222222222223323344444444555770776556666666770007666665444444 22222222222222233222322211101100000007007776666666666666777700122222222222222222211166666666666666666666666666444423334656564443222221001222222222222333344444455555570000065666666666777666645444
173	33322222222222223332222111011000000077777766666666666666666777001112222222222222222221106666666666666666666666644453444432222222222222222223333333444444556577000006666666666777066666444444 32222222222222222332222211011000000007070066766666666666666666670000012222222222222222222221107666666666666666666666656444444444432322222222222222223334344544467007555666666700077666665444444
174	33222212010077112176666701222222222222222222222223333344444455567000766556566666666666666545444333222221010112222244442222121111101001070170077777676666666666666666666666666666444443436544444 33322221100000002176666772222222222222222222222223334443445464566600070666656666666666666554444433222211010012222224442222222111111000000070707777767666667666666666666666666666644444344444544
175	32222211000112000666670111677012222222222222222222223234444444446567076644656566666666666554444333222211110002222222222222211100000000007777766666666666666666666666666666543333456532234656444444 33222211000012107666670111117712222222222222222222222323444344644456700666666666666666666554444333222121001122222222222222110100000000007767666666666666666666666666655333456653323456564444
176	33223322223344444433322222211000000066776666776666776677112222222222222222222233344434444444444444222211111000000000000007700006533444444433444455555770000007700077777766666666666666666666554444 23323322223333333322211111110000077667666666666666667770012222222222222222232334434444444444444644222212212211110001000000700007700000007656444444455777777777776666666666666666666666554444
177	33222222222221212112222345434677666665544433222222221110107701017077666666666666666660221212122122121111110077777667666666666666666666666544332222222222222222222223456556566566566566666666444445 33222222222212121222222464346776666665444333222222221110100770110077766666666666666660221222122122121211110007007667666666666666666666666544422222222222222222222233565566566566566566666666444
178	32222222233222222222211100667111000007767666666666677712222222222222221120006666666666666666666666665443233666556544333221111222222222222344454433565556666677777771233211107776666445656666664 32322222222222222222111000007010000777676666666666667670000011222222222222323210007666666666666666665544444434366443222111222222222323344543445566666666666667112122222222100066666666566566555444
179	32222222222222222222344222100100000000077776667666666666666600020011112222222222222222766666666666667707776666565444332222344555555622222222222222222223333333444444455670075555566666770777777656544 33222222222222211222322211101100000000077777777666666666666677767700000112222222222222222232111007666666666666666666766666664443355544444444333223222222222222222333344444445566666666666666777655544
180	33222223322222222222222222222211001100000007700777777666666666666666666666666666411222222233222222222222333333334444444456446555666666666666666777777771122122333322221110000776666666666555555566656 32222222222222222222222222221110010000000000007777666666666666666666676656563222222222222222222222222333444444444445555566666666666666667001211111343222222111000077777666666555666555554
181	22233223233222222222222222221101100077000001100077776666666666666666666666666666442222222222222222222222222334436555622333444455666666666666676677667111123333222222110000066666666666555555566544 22222223222222222222222222211111111017077011107007777767766676666666666666666666555434322222222222222222222223333336454334345566666666666666676667111213332321221100000077666666655555555555564434
182	33332222222222222222222333332222211111100000117700000077777776666666666666666666666666666666652222222222222222222222222223333333334444444455555577000000006555555766666666677000066666666554444 33322222222222222222211222333312211111100010000000000070007777676766666666666666666666666666666566644443222222222222222222222222322334433444444454655557077007766555555666667777077766666665555443
184	32222220012222222222222222223456444222222221121111111000001700000011070000000107766665544564444444444344445444455577000000000000007700077777666776666666666666666666654444444444444444444444444 32222220112222222222222222234465444222222221121111111100110000000000077011000000656665544445444433444444445544465770000000000000007077777776667766666666666666666654444444444444444444444444
185	22222222222222222222222233333332222211110000000776666666666666666666666666677111001101122222222222222222222222001176666666666666666666666667700777666666665544444443322222233455555555555644444 22332222222222222222222222223344433332222211110000077006666666666666666666666667777111001100111222222222222222222211007766666666666666666677700777766666666666555634443322223222233445555555555555
186	33222222222222222222222233334444332222221100000000776666666666666666666667766666667777111711000011113322222222222222222222222222211006666666666666666666666666666666666666445444444444444444444 22322222222222222222222344433322222222001000000777666666666666666666666666666700000000077000111122222222222222222222222222222006666666666666666666666666666666666555544444444444444444344
187	31111222222222233222234443322221110000777666666666666666666666666777700000000000011122222222222222222222222222222222222222206666666666666666666666666666666666666666444444444444444444444 33322222222222222222334443222222211000000777766666666666666666666667770000011000122222222222222222222222222222222222222120066666666666666666666666666666666666666555444444444454444444
188	33222222222222222222222222222221121110000000776766665554445556666666666667012111211120006767776777122222222222222222222201107066666666666666666666666666666654443223233232233345666666666666544 33222222222222222222222222222222221100000007666666655444556666666666666771112211100067777777711222222222222222222222220776666666666666666666666666666655444332333323334456666666666555
189	32222222222222222222222222211100000007666666454445666666666671121121121777767777712222222222222222222222222222222000076666666666666666666666666666666666666544443223233232345666666666544 33222222222222222222222222121100000007666666654455666666666670121121211776776777771222222222222222222222222222222000076666666666666666666666666666666666666544433232333233566666666665445

ตัวอย่างสตริงที่เป็นลบทั้งหมดของอักษร “ท” ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)

ASCII	Sample String
190	22222222222222222222233444322222111770076666666666666666600002112222212222122200776667666676666777122222222222223322177666666666666666666666666666666666644332223222232322322322244465666656666656666656666544 22222222222222222222334443332221210000007776676666666666666660022212122122122122776676676676676676712222222222222222200017666666666666666666666666666666644432322322322322232323366666666666656666666664444
191	23222222222222222224443322221200000077766666666666671122111221212212212206676767676676602222222222222222222222222221701766666666666666666666666666666666666666444322232322322323233566566666666656666664444 23222222222222222234333322221100777666666666666671112212221221222106667676667676660022222222222222222222222222207666666666666666666666666666666666644232223223223223233566566666666656666665666
192	33332211220000001122222222222222222224444111100111000000000000000777777666666666666666666666666666666666644332222222222222222222222222223344334444444444456667777445666666666666666666666666655444444 33332222211101112222222212223333222111111000100000000007000777777777666666666666666666666666666666444444222222222222222222222222223233333344444444444456667700707754656666666666666666666666655654444
193	33222222221010111222222222233644333322221211100000077077667666666666666666666666677007707000002222222222222222222222222220000006666666666666666666666666666666666666666544444333334334344446656655444444 33233222221111110112222222223334443322222221111000777776666666666666666666777700077077770071122222222222222222222222222277776666666666666666666666666666555633344333344333445566666666655554434
194	22222222222222222233322222222111100000000777666655445534455667700007000066444444456666666666667700000000000111122222222222222222222222222200056666666666666666666666666666666544444444444444444 22222222222222222113234433222222211111100000777766665544444445677770000000666444445666666666666600000000000111122222222222222222222222221106666666666666666666666666666655455444444444444444
195	33322222222222100110001222222222333333444444444445533322222112221111111111000000000000000000011000666666664445544444444443345544455667000000000077777777776666666666666666666666666555555554444444 22333222222211111000001122222222233333334444444444554422222222111222112211000110000000000007711000000666666654444444444444444444455567000000070000777766667666666666666666666666555555555444443
196	22222222222222222222222222222232233334344444545567700006666555666566660007707766666665544444433322222222222222221113443221122111111111001000710700070777766667666666666666666666666666666666544434 22222222222222222222222222222222222233223334444444445567700006666566666666677700076666666566444433322222222222222222233442222111111000010071170007777766766666666666666666666666666666555544
197	33222222222222111221112211007776676677667112222222232333333444444455666566666664444222221111111111000000000077777776666666666666666666666666644332223322233223332233655555566666600000666666666444444 33222222222222222222221010000000777777777676671222222222222233343444444444545446442222210100011000000000700077777776766666766666666666666666666544444332323323323333333445566666700007776666655644444434
198	32222222222222222222222222222222233333434444544546667000000666556666666666666666555444444333222222111100012222222244442222221221111111110010000000070077777776766666666667666666666666666666666666664444 32222222222222222222222222222222233333434444444545666700000066655566666666666666555544444433322222211100001222222224444222222122111111110010000000070070777776676667666666666666666666666666666554444
199	33333322222211100011111111113332222222222222222222223333334444444444456665555666666666666644422222222222222111111111000111000000777000777777777666666666666666666666666666666666666555555444444 33222222222222110000011111222222222222222222222333333333333444445555555555566666666664444422222222221122222111111100110000000007700777777777666677766666666666666666666666666666666555555554444444
200	32223322233232322222222222111111101000000000770700111776655666766666666666666666666666666666444432222222222222222222222344454422012334434454444556666666660022221111000070766666664455666666666666644 22223322332323232222222222221121100101100000007000707121177765556676666666666666666666666666666444432222222222222222222222334464421013334444444445555666666660022221111000077666666645566666666666656644
201	22220012222222222222223443332221211000007767666666666666666666670000000001122222344444434332222221100077666667022222222222220001766666666660211111765655555566666666666654444444444444444444344 22220122222222222222223433322221010007776666666666666666666666700000000001222223444343333322121100007666666701222222222222222001076666666666666660021117656555556666666666554444444444444444444344
202	33222222222222222222222220000000000001111133344444444544554433211100100000117000000777771221117766555556666666666666666666666666666666544322222233233322333333365666666666700000666666666645544444 33222222222222222222221110000000000110113334434444455444555566652222211111100110000000077007771110076655566666666666666666666666666666555534422222222222223333333444444446666666777777776655555544444
203	22222222222222222222223344443322222221001007000777676666666666666666666671122112111112113223222111101007777666666665676676666666666666666666544432222222222222222223365566555555555555555555555344 22222222222222232222222222343332222211100000076776666666666666666666671111111111111333222221100000007677666666677667766666666666666666666544443222222222222222222345655555555555555555555544
204	32222222222222222444333222111000077766666666666602121221212222006667666767676002222222222222223443332222101000777771210666555566666666666666666666666666666544442322323222232232566566566666656544 32222222222222222344343222211000000767666666666660021212212212211767667676766660022222222222223444333222210010007777121206666555666666666666666666666666666544442323223222332323668666666666656554
205	33222222222222222222221110000777766556665544555566667777000000000001111222222222222222222222333333446533455555554444221111100000000000007700776666666666666666666666666666554444444444444444 33233222222222222222222211100000777666665564445566666667766700007110001122222222222222222222222222344443444444445555554422211111000001100077000777777777766666666666666666666666555654544444444344
206	33222222222222222211110007776666656645566677077107000001112222222222222222222444454444444444444434433322211010100001000000007070071122000066655665676666666666666666666666555554444444444444434 33332222222222222222210100077776666555546666670000071100000012222222222222222223345445544444444444434433332111101000010000000007077712100000666555566766666666666666666666555554444444444444444
207	22211111222222222222222223344455544444444444444444333332222222222222220001100000000777666666665554444455577777000000001111111111122222222200077766666666666666666666666666666666555555444 22222111222222222222222222224444455544444444444444433333332222222222222211100000000000066666666664445544455577007771110000011111111222222221117777766666666666666666666666666666555555555555
209	33334444443333322222222221111111111100000777777666666666777777000007777771111100000000000000011111100000111111111177777666666555555555555554444444444444444444444444444444444444444 33332222222222222222222222221111100000000777777777666666666666666666666660000000000001111111111111111111122222111100006666666666665555555556666444455555554444444444444444444444444333344444

ตัวอย่างสตริงที่เป็นลบทั้งหมดของอักษร "ท" ที่ใช้ในการทดสอบการสร้างโมเดล (ต่อ)

ASCII	Sample String
210	3333222222222222222222222222222222222222222222222222222222222233322223333333344444444655555553333111111111000000000000000000777777776666777766666666666666666666666666666666666666666666666666666666 2222222222222222222222222222222222222222222222222222222222222333322233333334444444444444445555555633311110000111100000000000000000000777777777766666677776666666666666666666666666666666666666666666665554444
212	2222222222222221111111111111111111111111111111110000000000000000000000000000000000000000077777777777777766666777776666677777555555555533333444444444444444444444444444444444444444444444444444444444444444 2222222222222222111111111111111110000011110000000000000000000000000000000000000777777777777777777777777777777766666555444444444444444444444444444444444444444444444444444444444444444444444444444
213	3333444444444422221111111111111111111111111110000000000000000077777000000000777711112222222221110000000066666666666666666666666666666666666666666444444444443333444444444444433334444444444444444444445555 3334444444444444422222211122211111111111111111110000000000000000000000000000000000077777000111222222222222000000666666666666666666666666666666666666664444444444444444444444444444444444444444444555
214	3333333311111111111111111111111111111100000000000077771111222211112222111100000000000000007777666677776666555566666555566666666677766666666666644444444444433334444444444444333344444444444444444444445555 3333222222221111111111111111110000000000000000000000000001111222211111110000000000000077776666666666666555556666777766667777666666666644444443333444444444444444443333444444444444444444444444445555
215	3333322221111111111111110000000000000000000001111111111111666666666677770000011111222211117777766666666666666666666666666666666664444444444444444443333344444444444444444444444444444444444444444444 333222222111111111111110000000000000000000000011122222222222200006666666666666777666777111122222222222222222111777666666666666666666666666666666666666444444444333444444433344444444444444444444444444555
216	2222223333332222222222222222222333333344444444444433333333333322222222222222222211111111111000000000000000000000000077777766666777776666666666666666666666666666666666666666666666666666666666655555544444 2222222222222222222233333334444444444444433333333222222222222222222222222222222222221111110000000000000000077777777777777666666666666666666666666666666666666666666666666666666666666666666644444
217	3333333222222223334444333222222222222222211110000000000000077776666666666666666666666666777700001112222222222222222222222222222222200001111777666666666666666666666666666666666666665554444444444444444444 3333333333333322222222222222333355555555522222222211110000000000000000000777766666666666666666777777777111111111222222222222222222222222000000000000066666666666666666666666555556666555444444444444444444444
224	3333333333222333222222222222222222222222222222222222222222222222222222222222222221117700011177666666666666666666666666666666666666666666666677777700000007777776666666666666666666655444444554444444 2223332222222222222222222222222222222222222222222222222222222222222222222221117770000006666666666666666666666666666666666666666666666666666666777000777777776666666666666666555555555444444444333444
226	2233222222222222222222222222222222222222222222222222223333444444433222211100110000000000770000000006644444444444333444444556677770000077666666666666666666666666666666666666666666677777777777555564444443344 2233222222222222222222222222222222222222222222222233333344333332222111111100000000000077777776666666644332222223344334445555555666770077000666666666666666666666666666666666666677667700000776666666655444444444
227	3322222222222222222222222222222222222222222222222222222222233222223333556677666666555544334433222222211111111110000077777766666666666666666666666666666666666666666666666666677771007777666666666555444443344 2332222222222222222222222222222222222222222222222233222222222222222233443456670076666666665544444433322222221111111110000000700777667666666666666666666666666666666666666666666667707777666666555644444334
228	3323322222222222222222222222222222222222222222222222224446666666666666555433222233222222223332220007667666666000222122211100077666666666666666666666666666666666666666666666666677700000666666666666444444444 2232222222222222222222222222222222222222222222222222224446666666666666555555332232222222222333222000766766666600022222112220110777666666666666666666666666666666666666666666666677700000666666666665444444444
229	2222222222222222222222222222222222222222222222222222222222223332233333334444455444455555566444442222222111000110000000000000000777777766777666666666666666666666666666666666666666666666666666666666655544 2222222222222222222222222222222222222222222222222222222222332222333334433444554444555555664444332221111111000110000000000000000777777777777776666666666666666666666666666666666666666666666666666666666655544
230	2222222222222222222222222222222222222222222233222333344555333344355555555566666777701122220776666555444433443222222222221111111111100777001110077007777776666666666666666666666666666666666666666666666664444 3322222222222222222222222222222222222222222222223333355544333344455555566666667777112222200066666666555444444333322222221122211001110000777111100000777776677666666666666666666666666666666666666666665544
231	3333222222222222222221111000000000000000000110000111111110000000000666555555555544444445544444444444444445555566666677771122222110000000777711110000000000777776666555555554433443344433334444555566666644444443344 3333332222222222222222211100001100000000000000000111111110001100006666555555555555555544444444444444444455444445566666666770011222222110077007700001111100007777666666666655444444443344433443333556666666554444
232	2222222222222222222222222222222222222222222222222222222222222222222222222220000000000011111111111777777777777666666666666666666666666666666666666666666666666666666666666666664444444444444444444444444444444444 2222222222222222222222222222222222222222222222222222222222222222222000000000000000000000011111111116666666666666666666666666666666666666666666666666666666666666666666666644444444444444444444444444444444444
233	2222222000022222222222333322222222222222111111110000000777777766666666666666666666666666600000000001111111111111112221111122222222000000006666666666666666555555555555555555555555555544444554444444444444444444444444444444 2222222000011122223332222222222222222221111111100000007777777666666666666666555555666600000000011111111111111111111111222222221110000777666555555666666655555555555555555555544444445554444444444444444444444444444444
234	3333222222222222222222211000000066770011111000000777667766660000022222222222222211777666666666666666666655555555555555444422222211222222222223333445555544333355555666666666666667700112222211007766666665544444444 333222222222222222222222211111100077777771110001110007770006666666666666666666777111111222222222222222111777666666666666666555566655555554444445555444444422222211111112222222233333355555555555666666666655555544444
235	333332222222222222222223333344444444444444442222222222222222000000000000000011111222222222222222111110000077777666666666666666677777700000000000000000666666666666666444444444444444455555566666666666666666555554444 3333333333333333333333444444444444444422222222222222220000000000000000111111111111112222222220000000000000000007777777666666666777777700000000000000000066666666666666664444444444444455555555666666666665555555554444444
236	3333333333333322222222222222222222111112222111110000000001111100000000000000000000000011111111111117777766665555555555555444445555444444444444444444444444444666666666000007777666666666666666666665555556666644444444444 33333333333333222222222222222222222222211111111111000011111000000000000000000000000000000001111111111111777777666665555555555555444444444455554444444444444444455557777777777766666666666666666666666555555555444444444

ภาคผนวก ข.

ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

1. ณัฐฉัฐ ปานตระการ, กฤตวัน ศิริบูรณ์ และ บุญธีร์ เครือตราชู "การสร้างเสตทแมทชีนแบบอัตโนมัติเพื่อการรู้จำสตริง" **การประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 3 (The 3rd National Conference on Computing and Information Technology – NCCIT'07).** กรุงเทพฯ : คณะเทคโนโลยีสารสนเทศ, สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ. 25-26 พฤษภาคม 2550.

2. Kruatrachue B., Pantrakarn N. and Siriboon K. "Automatic State Machine Induction for String Recognition." **Proceeding of the 2007 International Conference of Computational Intelligence and Intelligent Systems (ICCIIS 2007), World Congress on Engineering 2007 (WCE 2007), London, UK,** July 2-4, 2007.

NCCIT'07

The 3rd National Conference
on Computing and Information Technology

ISBN : 978-974-19-3281-8

NCCIT'07

การประชุมทางวิชาการระดับชาติ ด้านคอมพิวเตอร์ ครั้งที่3 และเทคโนโลยีสารสนเทศ

25 - 26 พ.ค. 2550

คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

การสร้างเสตทแมทชีนแบบอัตโนมัติเพื่อการรู้จำสตริง

ณัฐฉัฐ ปานตระการ, ผศ.กฤตวัน ศิริบูรณ์ และ รศ.ดร.บุญธีร์ เครือตราชู

ภาควิชาวิศวกรรมกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

nattachatp@hotmail.com, kritawan@diamond.ce.kmitl.ac.th, booontee@yahoo.com

บทคัดย่อ

งานวิจัยนี้เสนอวิธีการสร้างแบบจำลอง ในรูปแบบเสตทแมทชีนแบบอัตโนมัติ เพื่อนำมาใช้รู้จำสตริงใด ๆ โดยใช้สตริงฝึกหัด ที่ได้จากข้อมูลทิศทางในรูปภาพของตัวอักษรภาษาไทย ในการสร้างเสตทแมทชีนจะแบ่งเป็น 2 ช่วง คือ ช่วงที่ 1 จะสร้างแมทชีนขึ้นมา จากสตริงของตัวอักษรเป้าหมายแต่ละตัว (Positive Training) และช่วงที่ 2 จะปรับแมทชีนให้ไม่ยอมรับสตริงอื่น ๆ ที่อยู่นอกเหนือจากตัวอักษรเป้าหมาย (Negative Training) วิธีนี้สามารถนำไปประยุกต์ใช้กับการรู้จำสตริงที่มีลักษณะเฉพาะ และสร้างโมเดลให้มีรูปแบบตามที่ต้องการได้

คำสำคัญ: การสร้างเสตทแมทชีน, การเรียนรู้ DFA, การรู้จำสตริง

Abstract

This research focus on generating a model in form of a state machine to recognize string derived from the direction information of Thai character's image. The state machine induction process has two steps. The first step is to generate the machine from the strings of each target character (Positive Training), and the second step is to adjust the machine to reject any other string (Negative Training). This method can be applied with specific string recognition. The generated model generalization can be controlled.

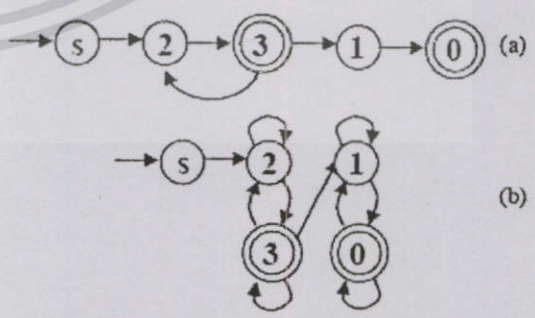
Keyword: State Machine Induction, DFA Learning, String Recognition.

1. บทนำ

ปัญหาในการสร้างโมเดลเพื่อรู้จำสตริงใด ๆ นั้น คือ จะทำอย่างไรให้โมเดลที่สร้างขึ้นมีลักษณะ generalized มากพอที่จะยอมรับสตริงที่มีรูปแบบคล้ายคลึงกันได้ แต่จะต้องเจาะจงมากพอที่จะไม่ยอมรับสตริงจากคลาสอื่น ๆ ด้วย โดยบทความนี้จะนำเสนอวิธีการสร้างเสตทแมทชีนแบบอัตโนมัติ ซึ่งได้จากสตริงตัวอย่างทั้งแบบ positive และ negative

จุดเด่นในงานวิจัยนี้ คือ เป็นวิธีการสร้างโมเดลที่สามารถควบคุมรูปแบบของโมเดล ให้มีโครงสร้างเป็นไปตามต้องการได้ นอกจากนั้น ยังนำตัวอย่างสตริงทั้งแบบ positive และ negative มาใช้สร้างโมเดล ซึ่งจะช่วยเพิ่มประสิทธิภาพในการรู้จำได้อีกด้วย

การสร้างเสตทแมทชีนโดยทั่วไปนั้น มักจะใช้วิธีการยุบ prefix tree ให้กลายเป็น DFA ที่เล็กที่สุด โดยที่ไม่ให้ขัดต่อตัวอย่างที่กำหนดไว้ เช่น EDSM [1], SAGE [2] เป็นต้น ซึ่งวิธีการเหล่านี้ อาจจะไม่เหมาะกับการรู้จำสตริงบางประเภทที่มีรูปแบบเฉพาะ



ภาพที่ 1 : โมเดลเปรียบเทียบระหว่างวิธีอื่นๆและวิธีในงานวิจัยนี้

ในงานวิจัยนี้ จะสามารถกำหนดรูปแบบในการสร้างโมเดลให้มีลักษณะตามต้องการได้ โดยการนำสตริงที่ได้จากตัวอักษรภาษาไทยมาเป็นต้นแบบ ซึ่งจะแบ่งได้เป็น 2 รูปแบบด้วยกัน คือ Loop และ Loop Pair [3] ดังนั้น โมเดลที่ได้จึงเหมาะสมกับสตริงตัวอย่างได้มากกว่าโมเดลทั่วไป ภาพที่ 1 จะแสดงความแตกต่างของโมเดลจากงานวิจัยนี้และวิธีอื่น ๆ ซึ่งสร้างมาจากสตริง "2310" และ "2323" ภาพที่ 1a เป็นโมเดลที่ได้จากวิธีอื่นๆ ส่วนภาพที่ 1b เป็นโมเดลที่ได้จากวิธีในงานวิจัยนี้ ถ้านำโมเดลมารู้จำสตริงใหม่ "23310" โมเดลแรกจะปฏิเสธสตริง ในขณะที่โมเดลที่ 2 จะสามารถรับสตริงได้ จะเห็นว่า อินพุตที่เกินมาเพียงตัวเดียวนั้น อาจจะมีผลกระทบให้โมเดลนั้นยอมรับหรือปฏิเสธสตริงได้ ทั้งที่ตามปกติแล้ว เราจะไม่สนใจว่าอินพุตมีจำนวนที่แน่นอนกี่ตัว เพียงแค่อินพุตนั้นมีตำแหน่งอยู่ภายในช่วงที่กำหนดก็เพียงพอแล้ว

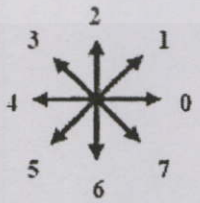
นอกจากนั้น ขั้นตอนวิธีนี้ ยังได้นำข้อมูลตำแหน่งและความยาวของอินพุตมาช่วยในการตัดสินใจ เพื่อเลือกว่าเสตทใดที่เหมาะสมในการเพิ่มทรานสิชั่นเข้าไปในโมเดล และยังนำมาเป็นข้อมูลในการรู้จำตัวอักษรในภายหลังอีกด้วย

งานวิจัยก่อนหน้านี้บางส่วน จะสร้างโมเดลขึ้นมาจากสตริงของคลาสเป้าหมายเท่านั้น เช่น CFG [3] HMM [4], [5] แม้ว่าจะสามารถยอมรับสตริงในคลาสนั้นๆได้ แต่อาจจะไม่สามารถปฏิเสธสตริงจากคลาสอื่นได้ ในงานวิจัยนี้ จึงใช้สตริงจากคลาสอื่นมาเป็นข้อมูลในการสร้าง trap state ซึ่งจะช่วยป้องกันไม่ให้โมเดลยอมรับสตริงอื่นได้นั่นเอง

2. นิยามและความรู้เบื้องต้น

ในส่วนนี้จะอธิบายถึงนิยามบางส่วนที่ใช้ในการสร้างเสตทแมทชิน ซึ่งจะกล่าวถึงในบทความนี้

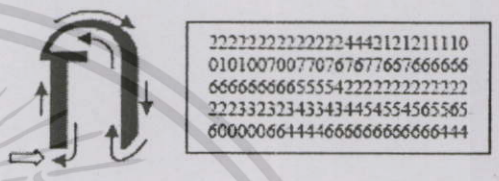
2.1 เชนโค้ด (Chain Code)



ภาพที่ 2 : ทิศทางทั้ง 8 ในเชนโค้ด

อักขระตัวพิมพ์ภาษาไทยแต่ละตัว จะถูกนำมาแสกนเป็นรูปภาพ จากนั้นนำมาแปลงเป็นสตริงที่เรียกว่า เชนโค้ด (Chain Code) โดยพิจารณาจากทิศทางในเส้นขอบของรูปภาพอักขระ เชนโค้ดจะแบ่งทิศทางออกเป็น 8 ทิศ ดังภาพที่ 2

เชนโค้ดเป็นลำดับของอินพุตที่แทนทิศทางทั้ง 8 ในการวนเส้นขอบอักขระ โดยเริ่มจากมุมซ้ายล่างวนไปจนบรรจบกับจุดเริ่มต้นอีกครั้งหนึ่ง ภาพที่ 3 แสดงตัวอย่างเชนโค้ดที่ได้จากอักษร "ก"



ภาพที่ 3 : ตัวอย่างเชนโค้ดของตัวอักษร "ก"

2.2 Deterministic Finite Automata (DFA)

Deterministic Finite Automata (DFA) จะนิยามโดยแมทชิน $M = (\Sigma, Q, \delta, q_0, F)$ โดย Σ เป็นเซตของตัวอักษรที่เป็นอินพุต, Q คือ เซตของเสตท, δ: $Q \times \Sigma \rightarrow Q$ คือ Transition Function, $q_0 \in Q$ เป็นเสตทเริ่มต้น, $F \subseteq Q$ เป็นเซตของ Final State สำหรับเสตท $d_0 \in Q$ ซึ่ง $\forall a \in \Sigma$, $\delta(d_0, a) = d_0$ จะเรียกว่า *Trap State*

2.3 Positive และ Negative Training Set

ให้ S_+ และ S_- เป็นเซตที่ไม่ Intersect กัน ซึ่งต่างเป็นซับเซตของ Σ^* จะเรียก S_+ ว่า *Positive Training Set* และเรียก S_- ว่า *Negative Training Set* [6] โดย S_+ จะเป็นเซตของสตริงใดๆบน alphabet Σ ซึ่ง accept โดย DFA M ส่วน S_- จะเป็นเซตของสตริงใดๆบน alphabet Σ ซึ่ง reject โดย DFA M นั่นคือ DFA M จะรับเซตของสตริง $S = S_+ \cup S_-$ ซึ่งจะ accept Positive Training Set และ reject Negative Training Set

2.4 Generalization

แมทชินจะ *Generalized* ก็ต่อเมื่อสามารถยอมรับสตริงที่มีรูปแบบคล้ายคลึงกับสตริงตัวอย่างได้ แมทชินที่มีลักษณะ *Over Generalized* คือ แมทชินที่รับสตริงมากเกินไป รวมทั้งสตริงที่ไม่ได้อยู่ในคลาสเดียวกัน ถ้าแมทชินไม่สามารถยอมรับสตริงที่ควรจะอยู่ในคลาสนั้น ๆ แมทชินนั้นมีลักษณะ *Under Generalized*

3. งานวิจัยที่เกี่ยวข้อง

การสร้างเสตทแมทชีนเพื่อการรู้จำสตริงนั้นได้มีการวิจัยกันอย่างกว้างขวาง Lang และ Peralmutter [1] ได้จัดการแข่งขัน Abbadingo One DFA Learning เพื่อสร้าง DFA จากตัวอย่าง และมีผู้ชนะสองราย คือ วิธีของ Price [1] และ Juillé [2] ขั้นตอนวิธีเหล่านี้ แม้จะเป็นวิธีการสร้าง DFA จากตัวอย่างทั้ง positive และ negative ที่มีประสิทธิภาพ แต่โมเดลที่ได้อาจจะมีลักษณะ over หรือ under generalized ได้ เนื่องจากเป็นวิธีที่ใช้ในการรู้จำสตริงทั่วๆไป ซึ่งไม่ได้เจาะจงลักษณะของสตริงดังเช่นที่ใช้ในงานวิจัยนี้

Hidden Markov Model (HMM) เป็นวิธีการสร้างเสตทแมทชินอีกวิธีหนึ่ง ซึ่งจะใช้ความน่าจะเป็นในการสร้างโมเดล [4], [5] โดยจะต้องมีการกำหนดจำนวนของเสตททั้งหมดไว้ล่วงหน้า ก่อนที่จะเริ่มทำการ training นอกจากนั้นวิธี HMM จะไม่สามารถกำหนดรูปแบบโครงสร้างของเสตทในโมเดลได้ และใช้เวลาในการ training ค่อนข้างมากอีกด้วย

งานวิจัยนี้ได้ทำการพัฒนามาจากงานวิจัยก่อนที่ใช้คอนเท็กฟรีแกรมม่า (CFG) ในการรู้จำตัวอักษรภาษาไทย [3] ในงานวิจัยก่อนจะใช้เฉพาะสตริงแบบ positive training ในการสร้างโมเดล ตามรูปแบบที่กำหนด คือ loop, loop pair และ trap loop pair ซึ่งในงานวิจัยนี้จะใช้ DFA แทน CFG และปรับโมเดลโดยใช้ negative training เพื่อให้สามารถปฏิเสธสตริงที่ไม่ใช่สตริงเป้าหมายได้ด้วย นอกจากนั้น ยังใช้ข้อมูลตำแหน่งและความยาวของอินพุตในสตริง มาช่วยในการเลือกเสตทที่เหมาะสมในการสร้างโมเดล และเพิ่มประสิทธิภาพในการรู้จำ

วิธีการที่ใช้ในงานวิจัยนี้จะเป็นวิธีการสร้าง DFA เพื่อใช้ในการรู้จำสตริงที่ได้จากอักษรภาษาไทย ซึ่งมีข้อดีหลายประการกล่าวคือ (1) โมเดลสามารถรับสตริงเป้าหมาย และปฏิเสธสตริงอื่นๆได้พร้อมกัน (2) สามารถควบคุมรูปแบบของโมเดลให้เป็นไปตามต้องการได้ โดยใช้ loop และ loop pair จาก [3] (3) ใช้ข้อมูลตำแหน่งและความยาว ช่วยในการสร้างและรู้จำสตริง (4) จะไม่มีทรานสิชั่นย้อนกลับภายในโมเดล ซึ่งอาจจะทำให้โมเดลเกิดลักษณะ over generalized ได้

4. ภาพรวมของระบบการทำงาน

ในการออกแบบระบบ เพื่อจะรู้จำสตริงในรูปแบบใด ๆ นั้น เป้าหมายหลัก คือ จะพยายามสร้างโมเดลสำหรับแต่ละคลาสของแพทเทิร์น ซึ่งจะต้องยอมรับสตริงในคลาสเดียวกัน แต่ปฏิเสธสตริงจากคลาสอื่น ๆ

ในการสร้างเสตทแมทชีนเพื่อการรู้จำตัวอักษรภาษาไทย จะมี 3 ขั้นตอนหลักด้วยกัน ดังนี้

1) *Preprocessing phase* จะทำการแปลงภาพตัวอย่างของตัวอักขระให้กลายเป็นสตริงในรูปแบบเชนโค้ด
2) *State Machine Induction phase* จะทำการสร้างโมเดลที่ยอมรับสตริงในแต่ละคลาส
   a. *Positive Training* จะสร้างโมเดลที่ยอมรับสตริงภายในคลาสเดียวกัน
   b. *Negative Training* จะปรับโมเดลที่ได้ เพื่อให้ปฏิเสธสตริงจากคลาสอื่น ๆ
3) *Competition phase* จะทดสอบสตริงที่ไม่ทราบค่ากับทุก ๆ แมทชีน เพื่อหาอักขระที่เป็นไปได้มากที่สุด

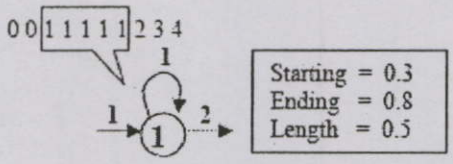
5. ขั้นตอนวิธีการสร้างเสตทแมทชีนแบบอัตโนมัติ

เสตทแมทชีนที่ใช้ในงานวิจัยนี้จะเป็น DFA ที่สร้างจากสตริงตัวอย่าง โดยทุก ๆ คลาสจะมี DFA 1 โมเดลต่อ 1 คลาส จากนั้น จะทำการรู้จำสตริงที่ไม่ทราบค่าว่าควรจะเป็นคลาสใด

5.1 Representation

เสตทที่อยู่ใน DFA จะมีลักษณะดังนี้ (1) ทุกเสตทจะถูกกำหนดค่าเป็นทิศทางในเชนโค้ด (0-7) ต้องมีทรานสิชั่นที่วนลูปเข้าที่เสตทเดิมสำหรับอินพุตที่ตรงกับค่านั้น ซึ่งแทนทิศทางเดียวกันของอินพุต (2) ทรานสิชั่นที่เข้าไปยังเสตทใดๆจะต้องเป็นทรานสิชั่นสำหรับอินพุตที่ตรงกับค่าของเสตทนั้น ๆ

ตัวอย่างของเสตทที่อยู่ใน DFA แสดงในภาพที่ 4 หลังอินพุตตัวที่ 2 ที่มีค่าเป็น "0" อินพุตตัวถัดไปจะเป็น "1" ดังนั้นจะมีทรานสิชั่นจากเสตทก่อนหน้านี้ สำหรับอินพุตเท่ากับ "1" ไปยังเสตทใหม่ที่มีค่าเป็น "1" ถ้ามีอินพุต "1" ซ้ำอีก ก็จะวนซ้ำอยู่ที่เสตทเดิมไปจนกว่าจะมีอินพุตค่าอื่น หลังจากวนซ้ำที่เสตทเดิม 5 ครั้งแล้ว ก็จะเปลี่ยนไปยังเสตทต่อไป เมื่อมีอินพุตใหม่เท่ากับ "2"

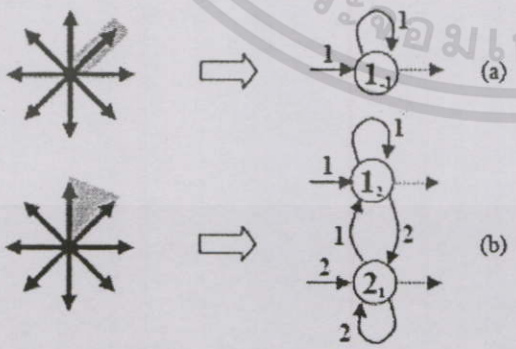


ภาพที่ 4 : รูปแบบเสตทที่ใช้ในเสตทแมทชิน

สำหรับในงานวิจัยนี้ จะเพิ่มข้อมูลในส่วนของตำแหน่งและความยาวอินพุตเข้าไป ซึ่งตำแหน่งทั้งหมดจะถูกแปลงให้อยู่ในช่วง [0, 1] ดังภาพที่ 4 ข้อมูลที่เพิ่มเข้ามาในแต่ละเสตท มีดังต่อไปนี้

- *Starting point* เป็นตำแหน่งน้อยที่สุดของอินพุตตัวแรกที่จะเข้ามายังเสตทนั้น
- *Ending point* เป็นตำแหน่งมากที่สุดของอินพุตตัวสุดท้าย ก่อนที่ย้ายไปยังเสตทถัดไป
- *Length* เป็นช่วงความยาวของอินพุตที่มากที่สุด ที่มีอินพุตวนอยู่ภายในเสตทเดียวกัน

เสตทที่สร้างขึ้นใน DFA จะมี 2 รูปแบบด้วยกัน คือ *Loop* และ *Loop Pair* โดย Loop จะแทนลำดับของอินพุตที่มีทิศทางเดียวกันซ้ำ ๆ หรือ *d** อินพุตค่าเดียวกันจะวนที่เสตทเดิม ส่วน Loop Pair จะแทนลำดับของอินพุตที่มีทิศทางอยู่ติดกัน หรือ *(d1*d2*)** โดยจะมีทรานสิชั่นไปและย้อนกลับสำหรับทั้งสองเสตทที่มีค่าทิศทางติดกัน ภาพที่ 5 แสดงตัวอย่างของ Loop และ Loop Pair โดยภาพที่ 5a จะแสดงเสตทเดี่ยวที่ใช้แทนลำดับของอินพุต "1" เช่น 1, 111, 1111111 ส่วนภาพที่ 5b จะแสดงคู่เสตทซึ่งมีค่าอยู่ติดกัน ในที่นี้คือ "1" และ "2" ซึ่งจะแทนอินพุตที่ได้จาก "1" และ "2" เช่น 12, 111212, 12122211



ภาพที่ 5 : Loop และ Loop Pair

ตัวเลขที่ห้อยภายในแต่ละเสตท จะใช้แทนค่าของคู่เสตทที่เหมาะสม ตัวอย่างเช่น เสตทที่มีค่า "1" จะสามารถมีคู่เสตทที่มีค่า "0" หรือ "2" ได้ ส่วนค่าคู่เสตทที่เท่ากับ "-1" นั้นหมายถึงไม่มีคู่เสตทสำหรับเสตทนั้น ค่าคู่เสตทนี้ จะใช้ช่วยให้เพิ่มคู่เสตทได้ง่ายขึ้น ถ้ามีค่าเป็น "-1" หรือไม่ได้กำหนดคู่เสตทไว้ จะสามารถเพิ่มคู่เสตทใด ๆ ให้เชื่อมกับเสตทนั้นได้ แต่ถ้ามีการกำหนดคู่เสตทไว้แล้ว จะสามารถเชื่อมได้เฉพาะเสตทที่มีค่าเท่ากับค่าคู่เสตทนั้นเท่านั้น

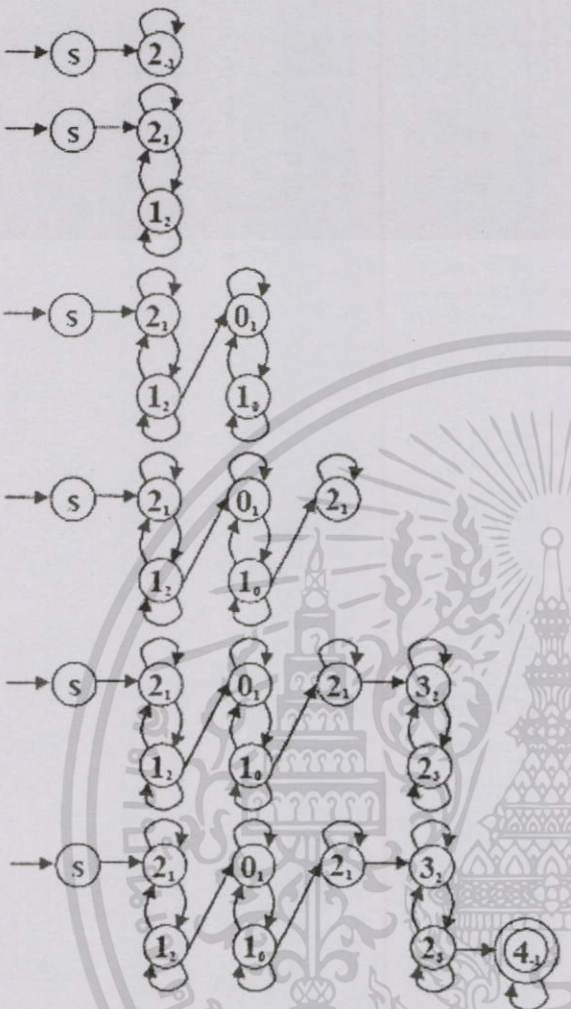
5.2 Induction phase

ในขั้นตอนนี้ การสร้าง DFA จะแบ่งเป็น 2 ช่วง เริ่มจากนำสตริงของคลาสเป้าหมายมาสร้าง DFA ที่ยอมรับสตริงเหล่านั้น ขั้นตอนนี้เรียกว่า *Positive Training* จากนั้น จะนำสตริงอื่น ๆ มาเทรนและเก็บข้อมูล แล้วสร้าง trap state เพื่อป้องกันไม่ให้สตริงอื่น ๆ ไปถึง final state ได้ ขั้นตอนนี้เรียกว่า *Negative Training*

ก่อนอื่น จะทำการสร้างเสตทเริ่มต้นสำหรับแมทชินนั้น ๆ ก่อน จากนั้น นำแพทเทิร์นแรกมาสร้างโมเดล โดยการอ่านอินพุตไปทีละตัว ถ้าไม่มีทรานสิชั่นสำหรับอินพุตนั้น ๆ ออกจากเสตทใด จะเพิ่มทรานสิชั่นไปยังเสตทที่เหมาะสม ถ้าไม่สามารถเพิ่มทรานสิชั่นได้ ก็จะสร้างเสตทขึ้นมาใหม่ โดยพยายามใช้ทรานสิชั่นและเสตทเดิมที่มีอยู่ให้มากที่สุด ทำซ้ำไปเรื่อย ๆ จนกว่าจะหมดสตริงนั้น เสตทสุดท้ายจะถูกกำหนดให้เป็น final state จากนั้น จะเริ่มทำซ้ำทั้งหมดอีกครั้งหนึ่งกับแพทเทิร์นอื่นๆต่อไป

การหาเสตทที่เหมาะสมในการเพิ่มทรานสิชั่น จะต้องเป็นเสตทที่มีตำแหน่งอยู่ในช่วงเดียวกัน หรือใกล้เคียงกับตำแหน่งของอินพุตปัจจุบัน และจะต้องเป็นเสตทที่อยู่ข้างหน้าถัดไปเท่านั้น ไม่สามารถเพิ่มทรานสิชั่นย้อนกลับไปยังเสตทที่อยู่ก่อนหน้าได้ ถ้าพบเสตทที่มีเงื่อนไขตรงตามข้อกำหนด จะเพิ่มทรานสิชั่นเข้าไปยังเสตทที่มีอยู่ แต่ถ้าไม่พบ จะสร้างเสตทใหม่ เงื่อนไขนี้จะช่วยลดการเกิดลูปภายในเสตท ซึ่งทำให้แมทชินยอมรับอินพุตมากเกินกว่าที่ควรจะเป็น

ภาพที่ 6 จะแสดงตัวอย่างการสร้าง DFA จากสตริงตัวอย่างตัวแรก ที่มีค่า "2210123244"

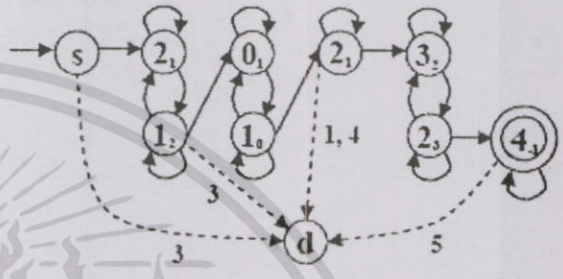


ภาพที่ 6 : ตัวอย่าง DFA ที่ได้จากขั้นตอน Positive Training

หลังจากสร้าง DFA จาก positive sample แล้ว ก็จะเก็บข้อมูลจากแพทเทิร์นของตัวอักษรจากคลาสอื่น ซึ่งเรียกว่า negative sample ถ้า DFA สามารถรับแพทเทิร์นใดได้ ก็จะเก็บข้อมูลการเกิดทรานสิชั่นที่ผิดพลาดบางส่วนไว้ เพื่อนำมาสร้าง trap state ในภายหลัง ความผิดพลาดจะเกิดก็ต่อเมื่อไม่มีทรานสิชั่นออกจากเสตทปัจจุบันสำหรับอินพุตนั้น ๆ ก็จะนับจำนวนความผิดพลาดที่เกิดขึ้น และยังคงวนอยู่ ณ เสตทเดิม จากนั้นจึงอ่านอินพุตตัวถัดไป

หลังจากเก็บข้อมูลของแพทเทิร์นทั้งหมดแล้ว ก็จะสร้างเสตทใหม่ trap state ขึ้นมา เพื่อใช้ป้องกันไม่ให้แพทเทิร์นที่เป็น negative เหล่านี้ ไปถึง final state ได้ โดยการสร้างทรานสิชั่นจากเสตทที่เกิดความผิดพลาดไปยัง trap state ซึ่งจะทำให้ DFA สามารถปฏิเสธสตริงที่เป็น negative ได้

ตัวอย่างของ DFA ที่ถูกปรับแล้ว หลังจากสร้างทรานสิชั่นไปยัง trap state แสดงในภาพที่ 7 จาก DFA ในภาพที่ 6 จะใช้ negative sample ลองทดสอบกับโมเดล 2 แพทเทิร์น คือ "32130223243" และ "2102143245" ซึ่งสามารถจบที่ final state ได้ จึงจะมีทรานสิชั่นบางส่วนที่สร้างจากเสตทที่เกิดความผิดพลาดไปยัง trap state



ภาพที่ 7 : ตัวอย่าง DFA ที่ได้จากขั้นตอน Negative Training

5.3 Competition phase

หลังจากขั้นตอน Training ทั้งหมดแล้ว ก็จะนำสตริงที่ไม่ทราบค่ามาทดสอบกับแมทชินทุกตัว เพื่อหาตัวอักษรที่ถูกต้อง โดยการทดสอบจะเริ่มใส่จากเสตทเริ่มต้น แล้วอ่านอินพุตเข้าไปทีละตัว ถ้ามีทรานสิชั่นสำหรับอินพุตตัวนั้น ก็จะย้ายไปเสตทใหม่ตามทรานสิชั่นที่มี ทำแบบนี้ไปเรื่อย ๆ จนครบทั้งสตริง ถ้าเสตทสุดท้ายเป็น final state แสดงว่า DFA รับสตริงนั้น จากนั้นนำไปทดสอบกับ DFA ของทุกคลาส

ในการทดสอบสตริงหนึ่ง ๆ นั้น อาจจะมี DFA หลายตัวที่สามารถยอมรับสตริงเดียวกันได้ ดังนั้น การเลือกว่าควรจะเป็น DFA ตัวใด จะใช้ข้อมูลความผิดพลาดที่เกิดขึ้นระหว่างการทดสอบมาประกอบการพิจารณา ความผิดพลาดมี 3 รูปแบบ ดังนี้

- *Transition error* จำนวนอินพุตทั้งหมดในเสตทใด ๆ ซึ่งไม่มีทรานสิชั่นสำหรับอินพุตนั้น ๆ
- *Position error* จำนวนอินพุตทั้งหมด ซึ่งตำแหน่งของอินพุตนั้น ไม่ได้อยู่ภายในช่วงตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายของแต่ละเสตท
- *Length error* จำนวนอินพุตทั้งหมด ซึ่งมีความยาวเกินกว่าความยาวอินพุตที่กำหนดในแต่ละเสตท

ถ้าไม่มีทรานสิชั่นสำหรับอินพุตตัวใด ก็จะคงอยู่ที่เสตทเดิมไปจนกว่าจะมีทรานสิชั่นออกจากเสตทนั้น ตามอินพุตที่เข้ามาใหม่ โดยระหว่างนั้น จะนับเป็น Transition error การเลือก

DFA ที่เหมาะสม จะเลือกจาก DFA ที่ยอมรับสตริงนั้น และมีความผิดพลาดรวมน้อยที่สุด

6. ผลการทดลอง

ในการทดสอบความถูกต้องในการรู้จำ จะนำขั้นตอนวิธีนี้มาใช้ทดสอบการรู้จำตัวอักษรตัวพิมพ์ภาษาไทย โดยใช้สตริงที่ได้จากเชนโค้ดทั้งหมด 5906 แพทเทิร์น จาก 66 ตัวอักขระ แบ่งเป็น training set ทั้งหมด 4920 แพทเทิร์น และ test set ทั้งหมด 986 แพทเทิร์น จำนวนเสตทเฉลี่ยของ DFA ที่สร้างขึ้นจะได้ 76 และ 77 เสตท สำหรับวิธีที่ใช้ positive training อย่างเดียว และวิธีที่ใช้ทั้ง positive และ negative training ตามลำดับ

ตารางที่ 1 : ผลลัพธ์ในการรู้จำเปรียบเทียบกับวิธีต่างๆ

Approach	Training set		Test set	
Hidden Markov Model	85.61%		82.56%	
Positive Training	98.70% *	1.30% **	79.41% *	1.62% **
Positive & Negative Training	98.70% *	1.30% **	83.67% *	2.33% **

* Percent of correct patterns
** Percent of unidentified patterns

ตารางที่ 1 จะแสดงผลลัพธ์ที่ได้การขั้นตอนการแข่งขันจากทั้ง training และ test set ซึ่งจะเทียบขั้นตอนวิธีในงานวิจัยนี้กับ Hidden Markov Model (HMM) โดยใช้ Left-Right-Left topology ที่กำหนดจำนวน 20 เสตท และเพิ่ม final state เข้าไป [5] วิธี HMM จะใช้เฉพาะตัวอย่างแบบ positive เท่านั้นในการ training ส่วนวิธีในงานวิจัยนี้จะใช้ตัวอย่างทั้ง positive และ negative ผู้ชนะที่ได้ คือ มีเลเบลเป็นตัวอักษรตัวเดียวกับที่ควรจะเป็น แสดงว่าสตริงนั้นสามารถรู้จำได้ถูกต้อง แต่มีบางแพทเทิร์นที่มีผู้ชนะมากกว่าหนึ่งตัวอักษร จะถือว่าแพทเทิร์นนั้นไม่สามารถระบุค่าได้ (Unidentified pattern) ในกรณีที่ใช้เฉพาะ Positive Training เท่านั้น ผลการรู้จำสำหรับ test set จะต่ำกว่าวิธี HMM แต่ถ้าใช้ Negative Training ร่วมด้วย จะมีผลการรู้จำพอๆกับวิธี HMM (1.11%) ส่วนการรู้จำ training set วิธีในงานวิจัยนี้จะได้ผลลัพธ์ที่ดีกว่า HMM ค่อนข้างมาก (13.09%)

7. บทสรุป

บทความนี้จะนำเสนอวิธีในการสร้าง DFA โดยอัตโนมัติจากตัวอย่างทั้งแบบ positive และ negative โดยจะทดสอบกับสตริงที่ได้จากเชนโค้ดของอักษรภาษาไทย วิธีนี้จะง่ายต่อการเข้าใจ และสามารถสร้างโมเดลและรู้จำสตริงได้อย่างรวดเร็ว ผลลัพธ์ในการรู้จำนั้นมีค่าไม่แตกต่างกับวิธี HMM

ปัญหาที่พบในการรู้จำตัวอักษร คือบางแพทเทิร์นมีลักษณะคล้ายคลึงกันมากเกินไป เช่น "ข" และ "ฃ" ซึ่งอาจจะไม่สามารถแยกแยะระหว่างรอยหยักที่หัว กับ noise ได้ สตริงที่มี noise อาจจะทำให้ได้ DFA ที่ไม่เหมาะสม เช่น บางทรานสิชั่นอาจจะทำให้ข้ามเสตทที่สำคัญไป ดังนั้น ก่อนจะนำสตริงมาใช้จึงควรขจัด noise ออกให้มากที่สุด

นอกจากนั้น อาจจะใช้คุณลักษณะอื่น ๆ มาช่วยในขั้นตอนการสร้างและการรู้จำได้อีกด้วย อย่างเช่น ในการรู้จำตัวอักษรแบบออฟไลน์ อาจนำสถิติการใช้เสตทและทรานสิชั่นมาช่วยในการตัดสินใจ ในการตัดเสตทหรือทรานสิชั่นบางส่วนที่ไม่จำเป็นออกไป เพื่อให้โมเดลเหมาะสมมากขึ้น

8. เอกสารอ้างอิง

[1] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the Abbadingo one DFA learning competition and a new Evidence-Driven State Merging algorithm," Lecture Notes in Computer Science, 1433: 1998, pp. 1-12.

[2] H. Juillé and J. B. Pollack, "A sampling-based heuristic for tree search applied to grammar induction," in Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98), Madison, Wisconsin, USA, 1998, pp. 26-30.

[3] B. Kruatrachue, P. Polsuntikul, and K. Siriboon, "Dynamic construction of context free grammar from sample for on-line Thai handwriting recognition," AISTA 2004: International Conference, Luxembourg, Nov. 15 – 18, 2004.

[4] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications inspeech recognition," Proceeding of the IEEE, Vol.77, 1989, pp. 257-286.

[5] K. Siriboon, A. Jirayusakul, and B. Kruatrachue, "HMM topology selection for on-line Thai handwritten recognition," in Proceeding of the first International Symposium on Cyber Worlds, 2002.

[6] O. Cicchello and S. C. Kremer, "Inducing grammars from sparse data sets: A survey of algorithms and results," Journal of Machine Learning Research 4, 2003, pp. 603-632.

Lecture Notes in Engineering and Computer Science

WCE 2007

World Congress on Engineering 2007

London, U.K.
2-4 July, 2007

S. I. Ao
Leonid Gelman
David WL Hukins
Andrew Hunter
A. M. Korsunsky (Eds.)

IA ENG
International Association of Engineers
ISBN: 978-988-98671-5-7

Automatic State Machine Induction for String Recognition

Boontee Kruatrachue, Nattachat Pantrakarn, and Kritawan Siriboon

Abstract—**One problem of generating a model to recognize any string is how to generate one that is generalized enough to accept strings with similar patterns and, at the same time, is specific enough to reject the non-target strings. This research focus on generating a model in the form of a state machine to recognize strings derived from the direction information of character's images. The state machine induction process has two steps. The first step is to generate the machine from the strings of each target character (Positive Training), and the second step is to adjust the machine to reject any other string (Negative Training). This automatic state machine induction method can also be applied with any string sequence recognition in other applications.**

Index Terms—State Machine Induction, DFA Learning, Character Recognition.

I. Introduction

This paper provides a novel state machine induction method for string recognition. The constraint in constructing model is to generate a model that is generalized enough to accept similar strings in the same class and is specific enough to reject any strings of other classes. The model proposed in this research is a state machine generated automatically from positive and negative training string samples.

The main advantage of our approach is that the generated models generalization can be controlled to suit the training strings. These usages of both positive and negative samples in generalization control helps in improving recognition.

Many previous researches use state machine or grammar induction to generate models from samples. Some DFA induction methods build the prefix tree acceptor from labeled sample and merge systematically equivalent states to construct the smallest DFA. These algorithms, such as Trakhtenbrot-Barzdin [1], EDSM [2], SAGE [3], or ALGERIA [4], etc., may not be suitable for recognizing strings in some specific domains. Usually, the generated models can either be over or under generalization. This is due to the algorithm trying to reduce number of states of the generated machine by preserving training pattern labels.

Our approach is more suitable for generating state machine that is not over or under generalized. In this case, the string contains direction alphabets derived from contour following of a character image. This string sequence has specific patterns that we can force in generating the state model. Hence the model generalization can be controlled.

For the directional chain code strings, there are 2 main patterns [5]. The first one, called *loop*, is the repeating of the same direction in a stroke, d^*. The second one, called *loop pair*, is the direction repetition between two adjacent directions in the chain code, $(d1^*d2^*)^*$.

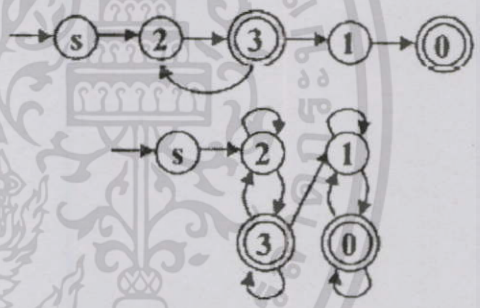


Figure 1 The models generated by other DFA induction and our algorithm.

The difference between model from other approaches and from ours is shown in Fig.1. Both models are generated from training string '2310', and '2323'. Fig.1a shows the model generated by other DFA learning algorithm that is under generalized. Fig.1b shows the suitable model from our approach. If the unknown string '23310' is tested with both models, the first one rejects, but the second one accepts. Actually, the string derived from direction is not concerned about the exact number of inputs. One input and two inputs of the same value are not different. In other DFA induction, only one repeating input can cause that string to be rejected, instead to be accepted.

In order to improve generalized control of the induction process, this research also uses position information of a stroke to decide whether to reuse existing state or to add a new state. This position information along with their length is also used in unknown string recognition.

Some preceding works use only samples in the targeted classes to generate models (positive training), such as CFG [5]

Manuscript received March 22, 2007.

Boontee Kruatrachue is Assoc. Prof. Dr. in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrubang, Bangkok 10520 Thailand (e-mail: booontee@yahoo.com).

Nattachat Pantrakarn is a master student in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrubang, Bangkok 10520 Thailand (e-mail: nattachatp@hotmail.com).

Kritawan Siriboon is Asst. Prof. in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrubang, Bangkok 10520 Thailand (e-mail: kritawan@diamond.ce.kmitl.ac.th).

or HMM [6], [7]. These models can accept all strings in the classes, but may not distinguish strings from any other class. In this research, we use information from negative samples to create some transitions to trap state. This method prevents the negative strings to reach the final states, in the other word; the models can reject strings from non-targeted classes.

We begin by presenting principle knowledge and some definitions of state machine induction in Section 2. Section 3 defines the previous researches corresponded to this work. We provide an overview of the system in Section 4, and the detail of the state machine induction algorithm in Section 5. Section 6 shows how to setup the experiment and the results for Thai character recognition. Finally, the conclusion and analysis is performed in Section 7.

II. Preliminaries

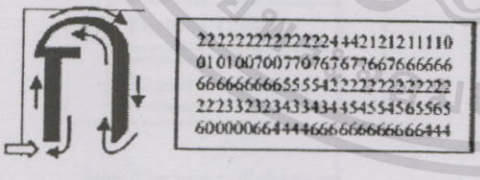
In this section, we describe some definitions of state machine induction problem that is used throughout this paper.

A. Chain Code

Each printed Thai character is scanned as an image. Each image is encoded as a string, called *chain code*, by using directional information of image contour. Chain codes used in this experiment are based on an eight-way directional system as shown in Fig.2

Figure 2 Chain code in 8-way directional system.

Chain code is a sequence of inputs generated from the contour following in 8-way direction of character image. Fig.3 shows the chain code derived from character 'ก'. The string starts from the left-bottom of image, and follows the contour



until meets the initial point.

Figure 3 Chain code example of character 'ก'.

B. Deterministic Finite Automata (DFA)

The Deterministic Finite Automata or Deterministic Finite State Machine is a finite state machine which has one and only one transition to a next state for each pair of state and input symbol. DFA recognizes the set of regular languages. DFA is defined as quintuple, $M = \{\Sigma, Q, \delta, q_0, F\}$, where

- Σ is a finite input alphabets,
- Q is a finite non-empty set of states,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final state.

A state $d_0 \in Q$ such that $\forall a \in \Sigma, \delta(d_0, a) = d_0$ is called *trap state*. Trap state has no transition out to other states.

C. Positive and Negative Training Set

Let the disjoint set S_+ and S_-, called *positive* and *negative training set* respectively [8], are subsets of Σ^*. S_+ is the set of any string over the alphabet Σ which is accepted by DFA M, and S_- is the set of any string over the alphabet Σ which is rejected by DFA M. Therefore, DFA M recognizes the set of strings $S = S_+ \cup S_-$ when it accepts positive training set and rejects negative training set.

D. Generalization

A machine is generalized when it accepts strings that have similar patterns to the samples. A machine is *over generalized* if it accepts too many strings that may not be in the class. If a machine is too restrict and loses the ability to accept strings that should be considered as the strings in the class, the machine is *under generalized*.

III. Related Works

There are many researches that applied automata induction approach with string recognition. Trakntenbrot and Barzdin introduced an algorithm for constructing the minimum DFA from complete labeled sample in polynomial time [1]. Lang and Pearlmutter [2] organized the Abbadingo One DFA Learning Competition to encourage work on DFA induction from sparse training data. Two algorithms from Price [2] and Juillé [3] are the winners. These algorithms are effective methods to minimize DFA from both positive and negative samples, but some transitions are too generalized or too specific for some strings because it was designed for any string in non-specific domain problem. There are other reviews in DFA learning algorithm in [8].

Hidden Markov Model (HMM) is another state model induction by using probabilistic information [6], [7]. The number of all states in model must be defined before the training stage begins. The structure of states in model cannot be controlled, while our approach can set the pattern of states to suit the strings. Moreover, HMM consumes much more time in the training step.

This research adopts from previous work that used context-free grammar (CFG) in Thai character recognition [5]. The previous research used only positive training strings to generate models using loop, loop pair, and trap loop pair policy. We represent all models as DFA instead of CFG, and adapt models to reject non-targeted strings by using negative samples. In addition, we use both the position and length information of inputs to determine appropriate state for each input.

Our approach is DFA induction method designed for string recognition derived from Thai characters. There are many

advantages in this method: (1) models can both accept positive strings and reject negative strings, (2) models can be controlled to accept similar strings by using loop and loop pair from [5], (3) models use the position and length information to distinguish each input in string, and (4) models have no loop back transition which results in over generalization.

IV. System Overview

To design a system to recognize any string pattern, the goal is to generate one model for each class of patterns such that the model has to accept any strings in the targeted class, and reject any strings in the non-targeted class.

In an automatic state machine induction for Thai character recognition, there are 3 main phases (Fig.4) as follows:-

1) *Preprocessing phase*: encoding each sample image of a character to string, called chain code, using directional information.
2) *State Machine Induction phase*: generating model to accept strings derived from each character class by using chain code strings as training set.
   a. *Positive Training*: generating model to accept any string derived from target character.
   b. *Negative Training*: adjusting model to reject any string derived from non-target character.
3) *Competition phase*: passing unknown string to each model and compare result to find the most possible character.

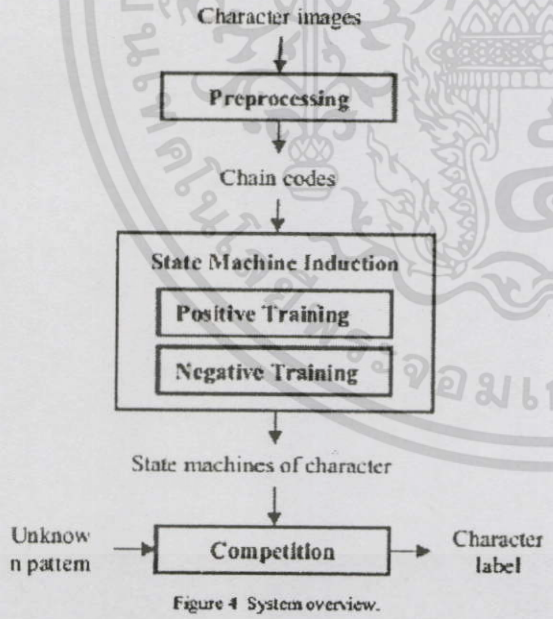


Figure 4 System overview.

V. Automatic State Machine Induction

State machines represented in this work are DFAs created from sample strings. We firstly begin with generating DFA from strings of targeted character class. Secondly, DFA is adjusted by creating new state, called trap state, and setting some transitions from the existing states to trap state. Each character class has one DFA model. After that, the unknown pattern is passed into each DFA to find which label it is.

A. Representation

All of the state machines represented in this research are Deterministic Finite Automata (DFA) with the following constraints. 1) Every state in DFA must have only one loop back to itself for the same input. This input is defined as its value. If there is a new input label at the same state, this input causes transition from this state to the next. The new input does not belong to the current state, since the state already has a loop. 2) The input that causes the transition will be the value of the next state (the next state will loop into that next state with the same input that cause the transition).

The representation of the state in DFA is demonstrated in Fig.5. After the second input '0', new '1' is coming. The transition from the previous state for input '1' is added to the new state valued '1'. Input '1' will repeat at the same state until the next value of input appears. After repeating '1' for five times, the next input '2' comes out and moves to the next state.

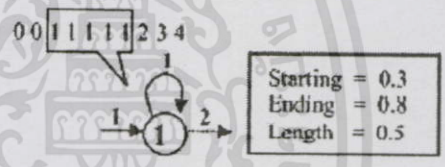


Figure 5 The representation in state machine.

Normally, the state has just the transition information but has no position or length information of each input that causes a transition in a state. In this research, we add position and length of input that cause the loop transition in the state. All positions are normalized in range [0, 1]. As shown in Fig.5, the additional information is used in each state as follows:-

- *Starting point* is the minimum position of the first input reaching that state,
- *Ending point* is the maximum position of the last input before moving to the next state,
- *Length* is the maximum interval of inputs that stay at that state.

Each state generated in DFA has two formats, called *loop* and *loop pair*. Loop state represents the sequence of identical direction inputs, d^*. The same input can repeat at the same state by looping into itself. Loop pair stands for the sequence of inputs that have adjacent direction, $(d1^*d2^*)^*$. If a new state represents input symbol that is adjacent to former input, forward and backward transitions are added for both states. Fig.6 shows the example of loop and loop pair states. Fig.6a shows a single state represented sequences of '1', such as 1, 111, 11111111, ... Fig.6b displays a pair of states that has adjacent value '1' and '2' represented any combination of '1' and '2', such as 12, 111212, 12122211, ...

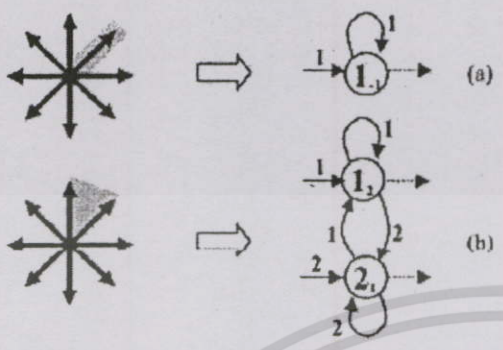


Figure 6 Loop and loop pair state.

The subscript number in each state represents the appropriated value of its pair state, for example, state value '1' can have pair state that has value '0' or '2'. The pair state value '-1' means that there is no expected pair state. This pair state value helps in adding new loop pair state easier. Any state with the subscript value '-1' or undefined loop pair can add any adjacent state as its loop pair. But any state with defined subscript can add a loop pair state that has a value equal to the defined subscript. Any state that already has a loop pair state can not add another loop pair state.

B. *Induction Phase*

In the induction phase, each DFA is generated by two steps. First, the strings from targeted class are used to generate a DFA that accepts strings in the targeted class. Second, all non-targeted strings are trained and a trap state is created to prevent negative samples reaching the final states of DFA. The former step is called *positive training*, and the latter is called *negative training*.

Initially, a starting state is created for every machine at the beginning. The first training string is used to create states and transitions of an initial model. Each input alphabet of the string is read. If there is no transition for that input, a new transition is added from the present state to the next proper state. If there is no proper state, a new state is generated and the transition for that specific input is added. These steps are repeated until the last input of the string. The current state at the end of the string is set as the final state. The next pattern is read, and all the processes are restarted at the starting state.

All additional transitions must be in the forwarding position, i.e., they cannot add the transition from current state to the previous state. Before creating a new state, all states in DFA are explored to find the state with the same value as the current input. The input position must also be inside that state's position. If that state exists and the transition is forwarded, the transition of that input is added to that state. Otherwise, a new state is created. This forward transition restriction is to avoid generating loop among many states. This causes the acceptance of extra sequences that does not occur in the trained pattern.

Fig.7 shows each step in generating DFA from the first training '2210123244' string pattern.

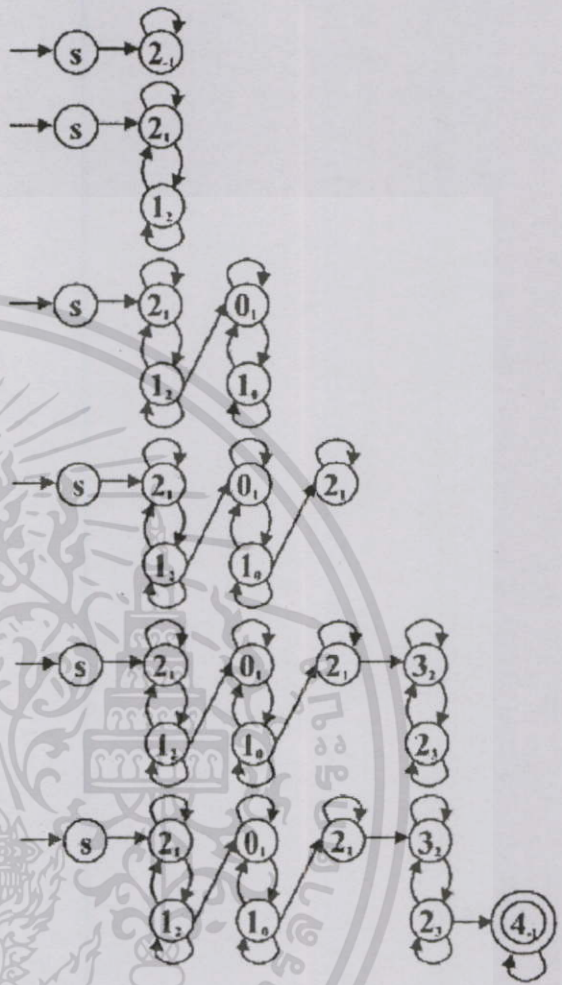


Figure 7 An example of DFA generated from positive training.

After DFA model is generated from positive samples, all patterns from any other character class, called negative samples, are passed into DFA to collect error information. If negative pattern can be accepted by DFA or reach the final state, some error information of that pattern is recorded to use as information to create trap state. The error can occur in any state where there is no transition defined for that input alphabet. Hence, if there is no defined next state for that alphabet, the next state remains in the same current state. The number of errors for each alphabet in each state is kept.

When all negative patterns have already passed, the new "*trap state*" are generated. Trap state is used to prevent negative patterns to reach the final states by adding transitions from error state to trap state. This forces DFA to reject negative strings.

An example of an adapted DFA after generating transitions to trap state is shown in Fig.8. From the final DFA generated in Fig.7, some negative patterns are passed into model. The

negative sequence strings, 32130223243 and 2102143245, can reach the final state. Some transitions are added from the states that have error to the trap state.

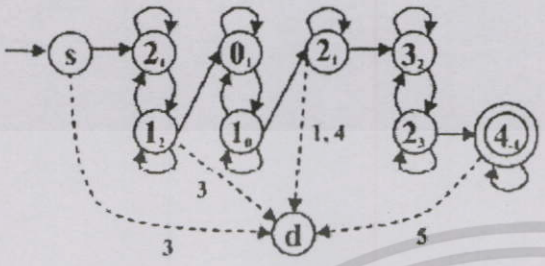


Figure 8 An example of DFA adapted from negative training.

C. Competition Phase

After all DFA models are generated from training samples, the unknown pattern can be identified in a competition phase. begin at the initial state, and then read each input one by one. If there is a transition for current input, then move to the next state. This process continues until all inputs have passed. If the last state is a final state, this means that DFA accepts this string. The unknown string is tested with every DFA.

There may be more than one DFA that accepts the input string. In order to choose the right one, the following parsing information is needed.

- *Transition error*. The total number of inputs in any state that have no transition.
- *Position error*. The total number of inputs that are not between starting and ending point of a current state. This includes all the inputs before starting and after ending position.
- *Length error*. The total number of inputs exceeding the length of a current state.

If there is no transition for current input, the current state does not change until there is transition out to the next state. To find the most aligned DFA, the minimum summation of all three errors is selected.

VI. Experiments

To test its accuracy, state machine induction method was applied with the recognition of Thai character images. All strings used in this experiment are derived from contour following of printed character's images in chain code direction. We use 5,906 patterns of both positive and negative samples from 66 character labels, including numbers, alphabets, vowels, and intonation marks, to generate 66 DFAs. There are 4,920 patterns for training set, and 986 patterns for test set. The average number of states for each generated DFA is 76 and 77 respectively for using only positive training, and using both positive and negative training. The negative training step used only one additional state, trap state, to improve recognition rate.

TABLE I
Competition Result

Approach	*Training set*		*Test set*	
Hidden Markov Model	85.61%		82.56%	
Positive Training	98.70% *	1.30% **	79.41% *	1.62% **
Positive & Negative Training	98.70% *	1.30% **	83.67% *	2.33% **

* Percent of correct pattern
** Percent of unidentified pattern

Table 1 shows the result in competition phase for training and test set. Our approach is compared to the well-known probabilistic state model approach, Hidden Markov Model (HMM). We used Left-Right-Left topology 30 states HMM with final state modification [7] for testing HMM. The HMM used only the positive training. The proposed state machine induction use both positive and negative training. If the winner in competition has the same label as expected label, that pattern is recognized correctly. The unidentified pattern is the pattern which has two or more winners when tested with all DFAs. This pattern cannot be identified which label it should be. The recognition accuracy for the test set of our approach is lower than HMM when positive training is used but about the same with HMM when used with both positive and negative training (1.11% better). In recognition the training set, the proposed state induction is much better than HMM (13.09%).

VII. Conclusion

This paper proposed a new approach to generate DFA automatically from both positive and negative samples by using position and length information. The algorithm was applied with strings taken from character's images in chain code direction. The generated models are simple and do not consume much more time in induction and competition phase. The recognition results are comparable to the HMM.

One problem found in character recognition is that some patterns are very similar, such as 'ข' and 'ฃ'. They cannot be differentiated between significant curve and noise. Noises of string cause many problems in recognition. Some noises cause the generated DFA to be over or under generalized. Some states are created but used only once or twice in training pattern. Some transitions are added but some significant states are skipped. These noises should be cleaned before the strings are used in training.

To improve efficiency, other features can be used in induction or competition phase. For example, in off-line character recognition, statistical usage of each state and transition can be used. Some unused states or transitions can be pruned to decrease restriction of models. Eliminating noise still performs the important role for string recognition in every specific domain.

REFERENCES

[1] B. Trakhtenbrot, and Ya. Barzdin, *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company, Amterdam, 1973.

[2] H. Juillé and J. B. Pollack, "A sampling-based heuristic for tree search applied to grammar induction," in Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98), Madison, Wisconsin, USA, 1998, pp. 26-30.

[3] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the Abbadingo one DFA learning competition and a new Evidence-Driven State Merging algorithm," Lecture Notes in Computer Science, 1433: 1998, pp. 1-12.

[4] R. C. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," In The 2nd Intl. Colllo. on Grammatical Inference and Applications, 1994, pp. 139-152.

[5] B. Kruatrachue, P. Polsuntikul, and K. Siriboon, "Dynamic construction of context free grammar from sample for on-line Thai handwriting recognition," AISTA 2004: International Conference, Luxembourg, Nov. 15 – 18, 2004.

[6] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications inspeech recognition," Proceeding of the IEEE, Vol.77, 1989, pp. 257-286.

[7] K. Siriboon, A. Jirayusakul, and B. Kruatrachue, "HMM topology selection for on-line Thai handwritten recognition," in Proceeding of the first International Symposium on Cyber Worlds, 2002.

[8] O. Cicchello and S. C. Kremer, "Inducing grammars from sparse data sets: A survey of algorithms and results," Journal of Machine Learning Research 4, 2003, pp. 603-632.

ประวัติผู้เขียน

ชื่อ – นามสกุล	นางสาว ณัฐฉัฐ ปานตระการ
วันเดือนปีเกิด	วันที่ 15 เดือนพฤษภาคม พ.ศ. 2526
ประวัติการศึกษา	พ.ศ. 2548 สำเร็จการศึกษาระดับปริญญาตรี
	หลักสูตรวิทยาศาสตรบัณฑิต เกียรตินิยมอันดับ 1
	จากคณะวิทยาศาสตร์ สาขาวิทยาการคอมพิวเตอร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง