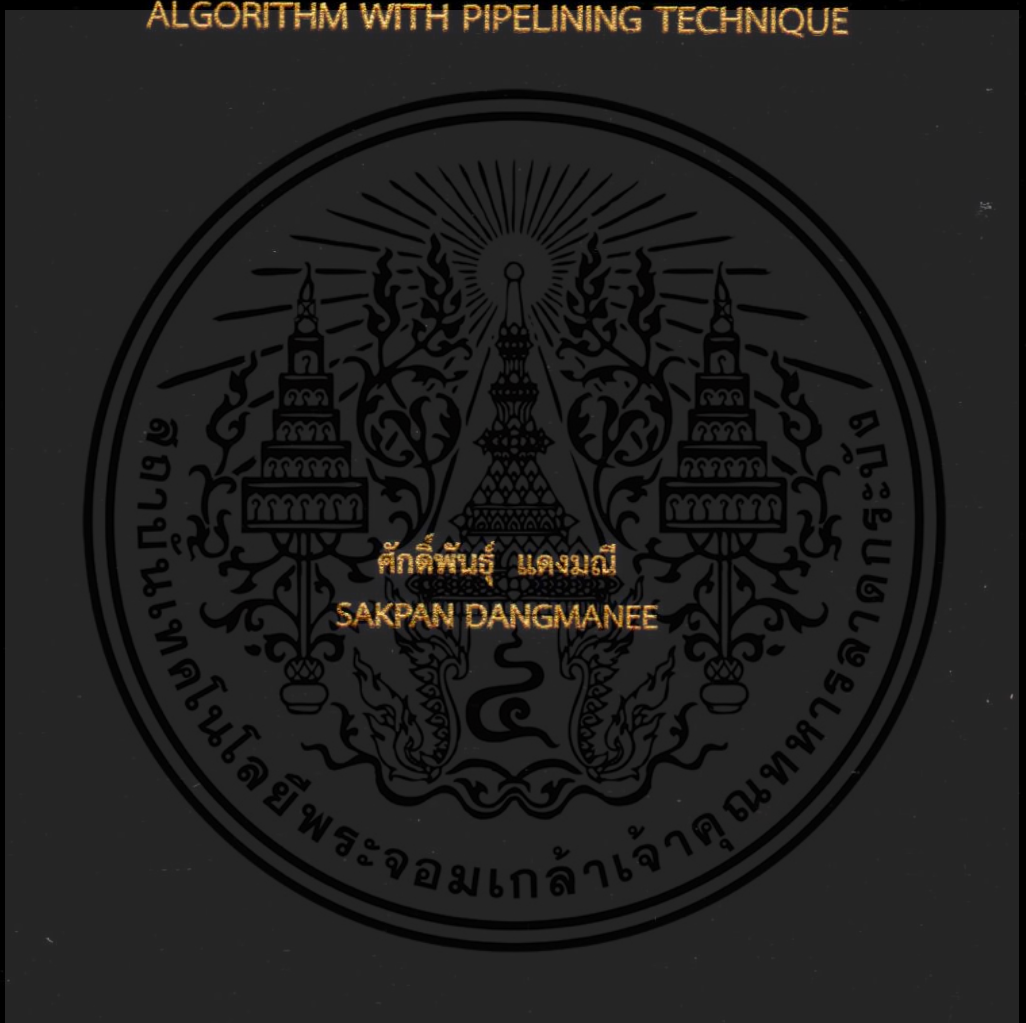
ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง

EFFICIENT PARALLEL LONGEST COMMON SUBSEQUENCE ALGORITHM WITH PIPELINING TECHNIQUE



ศักดิ์พันธุ์ แดงมณี
SAKPAN DANGMANEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2560
KMITL-2017-SC-M-002-002

ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง

EFFICIENT PARALLEL LONGEST COMMON SUBSEQUENCE ALGORITHM WITH PIPELINING TECHNIQUE

ศักดิ์พันธุ์ แดงมณี

SAKPAN DANGMANEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2560

KMITL-2017-SC-M-002-002

EFFICIENT PARALLEL LONGEST COMMON SUBSEQUENCE ALGORITHM WITH PIPELINING TECHNIQUE

SAKPAN DANGMANEE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2017
KMITL-2017-SC-M-002-002

คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ "ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง"
(EFFICIENT PARALLEL LONGEST COMMON SUBSEQUENCE ALGORITHM WITH PIPELINING TECHNIQUE)

ชื่อนักศึกษา นายศักดิ์พันธุ์ แดงมณี
รหัสประจำตัว 56605024
ปริญญา วิทยาศาสตรมหาบัณฑิต (สาขาวิทยาการคอมพิวเตอร์)
ภาควิชา วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร.จีรพร วีระพันธุ์

คณะกรรมการสอบวิทยานิพนธ์	ลายมือชื่อ
ผศ.ดร.อนันตพร หรรษคุณาฒัย ประธานกรรมการ	อนันตพร หรรษคุณาฒัย
ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์ อาจารย์บัณฑิตประจำ (ในสาขาวิชาที่เกี่ยวข้อง)	
ดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร ผู้ทรงคุณวุฒิจากภายนอกสถาบันฯ	
รศ.ดร.จีรพร วีระพันธุ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์	

วัน/ เดือน/ ปี ที่สอบ 23 มีนาคม พ.ศ. 2560 เวลา 16.00 - 19.00 น.
สถานที่สอบ ณ ห้อง 301 อาคารพระจอมเกล้า

คณะวิทยาศาสตร์รับรองแล้ว

(รองศาสตราจารย์ ดร.ดุษณี ธนะบริพัฒน์)
คณบดีคณะวิทยาศาสตร์
วันที่ 29 **เดือน** มี.ค. **พ.ศ.** 60

หัวข้อวิทยานิพนธ์	ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง
ชื่อนักศึกษา	นายศักดิ์พันธุ์ แดงมณี
รหัสประจำตัว	56605024
ปริญญา	วิทยาศาสตรมหาบัณฑิต
ภาควิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2560
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.จีรพร วีระพันธุ์

บทคัดย่อ

การหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS: Longest Common Subsequence) ของข้อมูล 2 ชุด (X, Y) ขนาด m และ n ที่เร็วและใช้พื้นที่จัดเก็บข้อมูลน้อยๆ มีบทบาทสำคัญอย่างมากในยุคปัจจุบันที่ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) อันเป็นผลเนื่องมาจากการใช้งานอุปกรณ์อิเล็กทรอนิกส์ของยุค IoT (Internet of Things) และการใช้งานแอปพลิเคชันสังคมออนไลน์ งานวิจัยเพื่อพัฒนาขั้นตอนวิธี LCS ที่ผ่านมา มุ่งเน้นที่การลดพื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำที่น้อยกว่าแบบเดิมที่เป็นอาร์เรย์ 2 มิติขนาด nxn ซึ่งต้องใช้พื้นที่ติดกันและเป็นข้อจำกัดในการประมวลผล เมื่อ n มีค่ามาก แต่ยังคงมีความซับซ้อนด้านเวลาของขั้นตอนวิธี LCS เป็น $O(n^2)$ ดังนั้นงานวิจัยนี้จึงนำเสนอขั้นตอนวิธี LCS แบบขนาน (Parallel LCS: PLCS) ที่จะประมวลผลด้วยเทคนิคไปป์ไลน์นิงที่มีประสิทธิภาพเร็วขึ้นเป็น O(n) และใช้พื้นที่ในการจัดเก็บข้อมูลได้อย่างเหมาะสมในขั้นตอนการจัดเตรียมข้อมูลที่จัดเก็บลงในอาร์เรย์ 1 มิติของลิสต์ พร้อมด้วยการค้นหา LCS หลายค่าแบบขนาน ซึ่งทำให้สามารถประมวลผลได้เร็วขึ้นและใช้พื้นที่ในการเก็บข้อมูลได้อย่างเต็มประสิทธิภาพ

คำสำคัญ : ลำดับร่วมเหมือนที่ยาวที่สุด (LCS), ขั้นตอนวิธี LCS แบบขนาน (PLCS), เทคนิคไปป์ไลน์-นิงสำหรับการประมวลผล PLCS

Thesis Title	Efficient Parallel Longest Common Subsequence Algorithm with Pipelining Technique
Student Name	Sakpan Dangmanee
Student ID	56605024
Degree	Master of Science
Department	Computer Science
Year	2017
Thesis Advisor	Assoc.Prof.Dr.Jeeraporn Weerapun

Abstract

Fast LCS (Longest Common Subsequence) of X(m) and Y(n) with space reduction really effects the Big Data Era and IoT (Internet of Things), where recently Big Data is increasing still very fast and unlimited by consuming of various electronic devices and social applications. A crucial limitation of the existing LCS approaches is the contiguous-space requirement for the nxn 2D-array in pre-processing. Existing LCS algorithms focuses on the improved memory-space by storing only important data in the pre-processing with the 2D-array processing in $O(n^2)$. Therefore, this work proposes the new parallel LCS (PLCS) algorithm for the efficient parallel pre-processing data in O(n) using a pipelining technique. Our efficient PLCS with space reduction can store large n with an array of multiple lists in the pre-processing step. In addition, our parallel searching time for finding the LCSs is also efficient.

Keywords : Longest Common Subsequence (LCS), Parallel LCS (PLCS), Pipelining-technique for PLCS.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มิอาจจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้และความเอาใจใส่จาก รศ.ดร.จีรพร วีระพันธุ์ อาจารย์ที่ปรึกษา ที่ท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ ผศ.ดร.อนันตพร หรรษคุณาฒัย ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์และดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบวิทยานิพนธ์ ที่ให้ความกรุณาให้คำแนะนำและชี้แนะจนวิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์

ขอขอบพระคุณบิดา มารดา ที่ให้การสนับสนุนในด้านต่างๆ และเป็นกำลังใจในระหว่างการศึกษาเล่าเรียนเป็นอย่างดี

ขอขอบคุณมหาวิทยาลัยพะเยา ที่ให้โอกาสในการทำงานและทุนสนับสนุนในการเรียนต่อในระดับปริญญาโทครั้งนี้

ขอบคุณเพื่อนๆ ที่ลาดกระบัง สำหรับมิตรภาพ ความช่วยเหลือ คำแนะนำต่างๆ คอยผลักดันให้ก้าวข้ามปัญหาและอุปสรรคต่างๆทั้งในการเรียนและการจัดทำวิทยานิพนธ์นี้

ขอบคุณรุ่นพี่ และรุ่นน้อง สำหรับความช่วยเหลือ และคำแนะนำในการดำเนินการจัดทำวิทยานิพนธ์ฉบับนี้

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับทุกคนที่กล่าวถึงในวิทยานิพนธ์นี้

นายศักดิ์พันธุ์ แดงมณี

สารบัญ

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในยุค IoT (Internet of Things หรือ Internet of Everything) มนุษย์นำอุปกรณ์อิเล็กทรอนิกส์มาใช้ในชีวิตประจำวันมากขึ้นเป็นทวีคูณ อุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ ถูกเพิ่มความสามารถให้สามารถประมวลผลอย่างง่ายตามความสามารถที่พึงมีของอุปกรณ์และยังสามารถเชื่อมต่อกับเครือข่ายต่าง ๆ เพื่อสามารถแลกเปลี่ยนข้อมูลกับอุปกรณ์อื่น อีกทั้งบนเครือข่ายอินเตอร์เน็ตมีการใช้งานแอปพลิเคชันสังคมออนไลน์ ข้อมูลที่เกิดจากการใช้งานอุปกรณ์อิเล็กทรอนิกส์และแอปพลิเคชันเหล่านี้ทำให้ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) ขั้นตอนวิธีต่างๆ ที่มีอยู่เดิมอาจไม่มีประสิทธิภาพที่เร็วเพียงพอกับการประมวลผลข้อมูลที่มีขนาดใหญ่ดังกล่าว ตัวอย่างเช่น ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS: Longest Common Subsequence) ซึ่งเป็นขั้นตอนวิธีที่ถูกนำมาประยุกต์ใช้ในงานมากมายหลายประเภท เช่น การเปรียบเทียบการเรียงตัวของ DNA หรือโปรตีนของผู้ป่วยกับคนปกติเพื่อตรวจหาความผิดปกติอย่างละเอียด[1] การตรวจสอบการคัดลอกโปรแกรมของนักศึกษา[2] การตรวจสอบเว็บไซต์ว่า เป็นเว็บไซต์ Phishing หรือไม่ [3] การตรวจสอบรูปแบบการป้อนอักขระด้วยท่าทาง (Gesture Character Input) [4] การตรวจสอบเสียงไซเรนของรถพยาบาล [5] เป็นต้น จากตัวอย่างการนำขั้นตอนวิธี LCS มาใช้ในงานตรวจสอบความเหมือนของข้อมูลชนิดต่างๆ ดังกล่าวอย่างแพร่หลาย เป็นผลอันเนื่องมาจากข้อดีของขั้นตอนวิธี LCS คือสามารถประมวลผลได้รวดเร็วกว่าและแม่นยำกว่าขั้นตอนวิธีแบบฮิวริสติกอื่น นอกจากนั้นยังมีกระบวนการและฟังก์ชันการคำนวณค่าที่ไม่ซับซ้อนจนเกินไป แต่หากยังต้องการพื้นที่จัดเก็บข้อมูลเพื่อการประมวลผลค่อนข้างมาก และในยุคที่ข้อมูลมีขนาดใหญ่มากๆ นี้ การใช้ขั้นตอนวิธี LCS แบบเดิมอาจส่งผลให้การประมวลผลไม่เสร็จทันในเวลาที่ต้องการ

การอิมพลีเมนท์ขั้นตอนวิธี LCS ของงานวิจัยในอดีตที่ผ่านมา เน้นเรื่องการลดพื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำเพราะจะต้องมีการสร้างเมตริกซ์ขนาด nxn โดยใช้โครงสร้างข้อมูลแบบอาร์เรย์ 2 มิติสำหรับใช้เก็บข้อมูลความยาวร่วมของข้อมูล 2 ชุด X และ Y (ขนาด m และ n) ซึ่งทางปฏิบัติพบว่า จะไม่สามารถประมวลผลเมื่อ n มีค่ามากได้ เนื่องจากอาร์เรย์ต้องใช้หน่วยความจำเก็บค่าที่ต่อเนื่องกัน (nxn ไม่เกิน 800x800) และจากการค้นคว้าของผู้วิจัยพบว่า ขั้นตอนวิธี LCS แบบใหม่ของ X. Xiang [6] เสนอการลดพื้นที่โดยมีการเก็บข้อมูลจุดเหมือน (Matching points) ลงในอาร์เรย์มิติเดียว ต่อมาขั้นตอนวิธี MLCS ของ J. Yang [7] จัดเก็บข้อมูลจุดเหมือนในรูปแบบข้อมูลโครงสร้างต้นไม้ที่เหมาะสำหรับข้อมูลหลายชุด และสุดท้ายขั้นตอนวิธี IDPSR-LCS [8] ของผู้วิจัยเองที่ลดการเก็บข้อมูลแบบอาร์เรย์ 2 มิติ ลงในอาร์เรย์ของลิสต์โดยไม่เก็บจุดที่เป็นข้อมูลซ้ำในแต่ละคอลัมน์ จากการเปรียบเทียบขั้นตอนวิธี LCS ของผู้วิจัยพบว่าทุกวิธียังคงมีความซับซ้อนของขั้นตอนวิธี LCS ในส่วนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) เป็น $O(n^2)$ นอกจากนั้นยังพบว่า มีงานวิจัยแบบขนานที่เน้นการประมวลผลที่เร็วขึ้น เพื่อรองรับความต้องการประมวลผลข้อมูลที่มีจำนวนมหาศาลให้ได้ผลลัพธ์ที่รวดเร็วยิ่งขึ้น ดังนั้นขั้นตอนวิธี LCS แบบขนาน (Parallel LCS) จึงถูกนำเสนอขึ้น เช่น ขั้นตอนวิธี Quick-DPPAR ของ Q. Wang [9] และขั้นตอนวิธี

ProMLCS ของ J.Yang [10] แต่อย่างไรก็ตามประสิทธิภาพของ MLCS แบบขนานในงานวิจัยที่ผ่านมามุ่งเน้นไปที่การประมวลผลข้อมูลหลายชุด (X_1 X_2, ..., X_s) ส่วนการออกแบบขั้นตอนวิธี LCS แบบขนานสำหรับข้อมูล 2 ชุด (X, Y) ยังมีข้อจำกัดในส่วนของขั้นตอนการจัดเตรียมข้อมูล ซึ่งทำได้ค่อนข้างยาก เพราะค่าของแต่ละสมาชิกจะต้องคำนวณจากผลลัพธ์ที่ต่อเนื่องกัน

จากปัญหาข้างต้น ผู้วิจัยจึงได้เสนอขั้นตอนวิธี PLCS-SR ที่เป็นขั้นตอนวิธี LCS แบบขนานที่มีประสิทธิภาพเร็วขึ้นเป็น O(n) ด้วยการลดเวลาส่วนใหญ่ที่ใช้ในการประมวลผล คือขั้นตอนการจัดเตรียมข้อมูล (Pre-processing) ด้วยเทคนิคไปป์ไลน์นิง พร้อมด้วยการลดพื้นที่ในการจัดเก็บค่าความยาวของ LCS โดยปรับโครงสร้างการจัดเก็บที่เสนอในงานวิจัยเดิม [8] ให้เหมาะสมเพื่อการประมวลผลแบบขนานที่มีประสิทธิภาพ และในส่วนของขั้นตอนการค้นหา LCS หลายค่า ก็จะสามารถค้นหาได้เร็วขึ้นด้วยการประมวลผลแบบขนาน (หนึ่งหน่วยประมวลผลต่อหนึ่ง LCS) เพื่อลดเวลาในการค้นหา LCS ทั้งหมดด้วยความซับซ้อนด้านเวลาเท่ากับ O(n)

1.2 วัตถุประสงค์ของงานวิจัย

งานวิจัยนี้มีวัตถุประสงค์ เพื่อลดพื้นที่ในการจัดเก็บข้อมูล และเพื่อลดเวลาในการประมวลผลของขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุด ดังนี้

1. นำเสนอขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดด้วยวิธีไดนามิคโปรแกรมมิงแบบลดพื้นที่ เพื่อลดพื้นที่ในการจัดเก็บข้อมูล ด้วยอาร์เรย์ 1 มิติของลิสต์แบบใหม่ แทนอาร์เรย์ 2 มิติแบบเดิม
2. ออกแบบขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนาน เพื่อลดเวลาของการประมวลผล ด้วยเทคนิคไปป์ไลน์นิงบนโมเดล CREW-PRAM ร่วมกับการลดพื้นที่ที่ใช้ในการประมวลผลของขั้นตอนวิธีให้เหมาะกับการนำมาประมวลผลกับข้อมูลที่มีขนาดใหญ่ในยุคปัจจุบัน (Big Data)

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้ได้นำเสนอขั้นตอนวิธีขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดที่ปรับปรุงจากวิธีไดนามิคโปรแกรมมิง เพื่อลดพื้นที่ในการจัดเก็บข้อมูล พร้อมด้วยการออกแบบขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานด้วยเทคนิคไปป์ไลน์นิงบนโมเดล CREW-PRAM เพื่อลดเวลาในการประมวลผล โดยทำการทดสอบด้วยการจำลองชุดข้อมูลจากการสุ่มเลือกตัวอักษรภาษาอังกฤษตามความยาวของลำดับอักษรที่กำหนดเพื่อแสดงถึงประสิทธิภาพของความเร็วที่เพิ่มขึ้นพร้อมกับใช้พื้นที่ในการประมวลผลที่ลดลง

1.4 ส่วนประกอบของวิทยานิพนธ์

ส่วนประกอบของวิทยานิพนธ์ฉบับนี้ประกอบด้วย

บทที่ 2 กล่าวถึง ทฤษฎีและงานวิจัยที่เกี่ยวข้อง ของขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุด ซึ่งประกอบด้วย ขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดแบบไดนามิคโปรแกรมมิง, ขั้นตอนวิธีค้นหาลำดับร่วมเหมือนที่ยาวที่สุดในงานวิจัยอื่นๆ

บทที่ 3 กล่าวถึงการหาลำดับร่วมเหมือนที่ยาวที่สุดด้วยวิธีไดนามิคโปรแกรมมิงแบบลดพื้นที่และผลการทดลองในการลดพื้นที่

บทที่ 4 กล่าวถึงขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง ซึ่งประกอบด้วย ขั้นตอนวิธี PLCS, ขั้นตอนวิธี PLCS-SR และ ประสิทธิภาพของขั้นตอนวิธี PLCS และ PLCS-SR

บทที่ 5 สรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงปัญหาของการค้นหาสายอักษรร่วมที่ยาวที่สุด วิธีการแก้ปัญหาแบบไดนามิคโปรแกรมมิง และงานวิจัยที่เกี่ยวข้องขั้นตอนวิธีสำหรับการแก้ปัญหาการค้นหาสายอักษรร่วมที่ยาวที่สุด

2.1 ปัญหาการค้นหาสายอักษรร่วมที่ยาวที่สุด

ปัญหาการค้นหาสายอักษรร่วมที่ยาวที่สุด เป็นการเปรียบเทียบสายอักษร ที่อยู่ในลักษณะของข้อมูลสายอักษร จำนวน 2 ชุด กำหนดให้เป็น X และ Y โดยที่ $X = <x_1, x_2, \ldots, x_m>$ เป็นสายอักษรชุดหนึ่งที่มีความยาว m ตัวอักษร และ $Y = <y_1, y_2, \ldots, y_n>$ เป็นสายอักษรอีกชุดอีกหนึ่งที่มีความยาว n ตัวอักษร เช่น

กำหนดให้ สายอักษร X = BECECBCCBE และ Y = EBDEDCBEEA

X = B E C E C B C C B E

Y = E B D E D C B E E A

สายอักษรร่วมที่ยาวที่สุด ชุดที่ 1 (LCS_1) คือ BECBE

X = B E C E C B C C B E

Y = E B D E D C B E E A

สายอักษรร่วมที่ยาวที่สุด ชุดที่ 2 (LCS_2) คือ BECEE

2.2 วิธีการค้นหา LCS แบบไดนามิคโปรแกรมมิง

ในการหาสายอักษรร่วมที่ยาวที่สุด (LCS) $Z = <z_1, z_2, \ldots, z_k>$ (ขนาด k ตัวอักษร) ของสายอักษร X_m และ Y_n ขั้นตอนวิธีในรูปที่ 2.1 จะดำเนินการค้นหา LCS แบบ Backward โดยพิจารณาจากค่าความยาวร่วมที่ประมวลผลไว้ก่อน (Pre-processing) ในเมตริกซ์ C ขนาด mxn โดยเริ่มค้นหาจาก (i, j) แรกที่มีค่า $c_{i,j}=k$

ในขั้นตอน Pre-processing จะจัดเตรียมเมตริกซ์ C_{mxn} ที่เก็บค่าของความยาวร่วมของสายอักษร (LCS) เมื่อ $c_{i,j}$ แทนค่าความยาวของ LCS จากสายอักษร X_m และ Y_n ที่คำนวณจากทุกค่าของ x_i และ y_j สำหรับ i = 1, 2, ..., m และ j = 1, 2, ..., n

เมื่อ $c_{i,j} = c_{i-1,j-1} + 1$ ถ้า $x_i = y_i$

$= \max(c_{i-1,j}, c_{i,j-1})$ ถ้า $x_i \neq y_i$

จากรูปที่ 2.1 แสดงถึงขั้นตอนวิธีการสร้างเมตริกซ์ C_{mxn} โดยในขั้นตอน Pre-processing มีรายละเอียดของการดำเนินขั้นตอนวิธีดังต่อไปนี้

1) นำสายอักษรที่ต้องการเปรียบเทียบ 2 ชุด กำหนดให้ชุดแรก มีค่า X และชุดที่สองมีค่า Y

2) ให้ความยาวของสายอักษรชุดแรกแทนค่าด้วย m และความยาวของสายอักษรชุดที่สองแทนด้วยค่า n

3) กำหนดค่า i เริ่มต้นเป็น 1

4) กำหนดค่า j เริ่มต้นเป็น 1

5) ทำการวนลูปตรวจสอบจากตัวอักษรที่ j จากสายอักษร Y ตั้งแต่ j มีค่าเท่ากับ 1 ไปจนถึง J

6) ทำการเปรียบเทียบตัวอักษรตัวที่ i จากสายอักษร X กับตัวอักษร j จากสายอักษร Y ว่ามีค่าเหมือนกันหรือไม่

ถ้าเหมือนกัน ให้กำหนดค่าในตำแหน่งที่ i,j มีค่าเท่ากับค่าจากตำแหน่ง i-1,j-1 ทำการบวกค่า 1

ถ้าไม่เหมือน ให้กำหนดค่าในตำแหน่งที่ i,j มีค่าเท่ากับค่าจากตำแหน่ง i-1,j หรือ i,j-1 โดยเลือกค่าที่มากกว่า

7) เมื่อตรวจสอบจากตัวอักษรที่ j จากสายอักษร Y จนถึง n ให้เพิ่มค่า i ไปอีก 1 และกำหนดให้ค่า j กลับมาเริ่มต้นที่ 1 ไปจนถึง n จนกว่า i จะมีค่าเท่ากับ m

หมายเหตุ กำหนดให้ $c_{i,0}$ และ $c_{0,j}$ มีค่าเป็น 0

```
for (i=1; i<=m; i++)
  for(j=1; j<=n; j++)
    if(xi=yj) ci,j = ci-1,j-1 +1;
    else ci,j = max(ci-1,j , ci,j-1);
 end for j, i;
```

รูปที่ 2.1 ขั้นตอนวิธีการสร้างเมตริกซ์ C_{mxn}

เมื่อขั้นตอน Pre-processing ทำการจัดเตรียมเมตริกซ์ C_{mxn} ที่เก็บค่าของความยาวร่วมของสายอักษร (LCS) เสร็จสิ้นจะได้เมตริกซ์ C_{mxn} ดังตัวอย่างในรูปที่ 2.2 แสดงการคำนวณค่าเมตริกซ์ C_{10x10} ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA

X \ Y	E	B	D	E	D	C	B	E	E	A	
B	0	1	1	1	1	1	1	1	1	1	1
E	1	1	1	2	2	2	2	2	2	2	2
C	1	1	1	2	2	3	3	3	3	3	3
E	1	1	1	2	2	3	3	4	4	4	4
C	1	1	1	2	2	3	3	4	4	4	5
B	1	2	2	2	2	3	4	4	4	4	6
C	1	2	2	2	2	3	4	4	4	4	7
C	1	2	2	2	2	3	4	4	4	4	8
B	1	2	2	2	2	3	4	4	4	4	9
E	1	2	2	3	3	3	4	5	5	5	10
	1	2	3	4	5	6	7	8	9	10	

รูปที่ 2.2 ตัวอย่างเมตริกซ์ C

จากรูปที่ 2.3 แสดงถึงขั้นตอนวิธีการค้นหา LCS มีรายละเอียดของการดำเนินขั้นตอนวิธีดังต่อไปนี้

1) เริ่มต้นจากจุดที่ค่า k มากที่สุด (ค่า k เป็นค่า c_{ij} ที่มีค่ามากที่สุด) กำหนดให้ค่า I มีค่าเท่ากับตำแหน่งของตัวอักษรบนสายอักษร X และค่า j มีค่าเท่ากับตำแหน่งของตัวอักษรบนสายอักษร Y ที่มีค่า k มากที่สุด

2) ทำเปรียบเทียบตัวอักษร x ที่ตำแหน่งที่ i และตัวอักษร y ที่ตำแหน่งที่ j

ถ้าเหมือนกัน ให้กำหนดให้ตัวอักษร z ที่ตำแหน่งที่ k เป็นตัวอักษร x ที่ตำแหน่ง i จากนั้นนำค่า k, i และ j มาลบออก 1

ถ้าไม่เหมือนกัน โดยที่ $c_{i-1,j}<c_{i,j-1}$ นำค่า j มาลบออก 1

นอกจากกรณีข้างต้น นำค่า i มาลบออก 1

```
BackwardLCS (Cmn, Xm, Yn, Zk, i, j, k)
    while(k>0) do
        if(xi=yj) zk=xi; k=k-1; i=i-1; j=j-1;
        else if(ci-1,j<ci,j-1) j=j-1; else i=i-1;
    end while;
end LCS;
```

รูปที่ 2.3 ขั้นตอนวิธีการค้นหา LCS

ตัวอย่างในรูปที่ 2.4 แสดงการหา LCS จากเมตริกซ์ C_{mxn} แบบ Backward จากค่า c_{ij} ที่ยาวที่สุด (ในที่นี้คือ k=5) และในตัวอย่างนี้จะได้ LCS คือ BECBE และ BECEE

X \ Y	E	B	D	E	D	C	B	E	E	A		LCS_1	LCS_2
B	0	(1)	1	1	1	1	1	1	1	1	1	B	B
E	1	1	1	[2]	2	2	2	2	2	2	2	E	E
C	1	1	1	2	2	[3]	3	3	3	3	3		C
E	1	1	1	2	2	3	3	[4]	4	4	4		E
C	1	1	1	2	2	3	3	4	4	4	5		
B	1	2	2	2	2	3	4	4	4	4	6		
C	1	2	2	2	2	3	4	4	4	4	7		
C	1	2	2	2	2	(3)	4	4	4	4	8	C	
B	1	2	2	2	2	3	(4)	4	4	4	9	B	
E	1	2	2	3	3	3	4	(5)	[5]	5	10	E	E
	1	2	3	4	5	6	7	8	9	10			

รูปที่ 2.4 ตัวอย่างการค้นหา LCS ด้วยขั้นตอนวิธี DP เดิม

2.3 งานวิจัยที่เกี่ยวข้องกับการแก้ปัญหา

2.3.1 วิธีการค้นหา New LCS

1. construct P = {(i,j): $i_i = j_j$ in the order of I}
2. construct P_1, P_2, ... , P_l,
where l =number of matches between I and J
3. LCS(I,J) = max{P_i ; i = 1, 2, ..., l}

รูปที่ 2.5 ขั้นตอนวิธี New LCS (I, J) ของ X. Xiang [6]

ในส่วนของขั้นตอนวิธีของ X. Xiang และคณะ [6] การจัดเตรียมข้อมูลก่อนทำการค้นหา LCS จะทำการคำนวณข้อมูล (ค่าความยาวร่วม) ทั้งหมดก่อน เช่นเดียวกับขั้นตอนวิธี DP แต่วิธีนี้จะจัดเก็บเฉพาะบางคู่ลำดับที่เป็น Pre-sequence point ที่จะใช้ในการค้นหา LCS โดยเก็บค่าในอาเรย์ 1 มิติ (P) ซึ่งเป็นคู่ลำดับ (i, j) จากตำแหน่งของตัวอักษรที่เหมือนกัน ที่ปรากฏอยู่ในสายอักษรแต่ละชุด ดังแสดงเป็นตัวอย่างในรูปที่ 2.6

I \ J	T	G	C	A	T	A	
A	0	0	0	1	1	1	1
T	1	1	1	1	2	2	2
C	1	1	2	2	2	2	3
T	1	1	2	2	3	3	4
G	1	2	2	2	3	3	5
A	1	2	2	3	3	4	6
T	1	2	2	3	4	4	7
	1	2	3	4	5	6	

$P = \{(i, j) : i_i = j_j\}$.
$P = \{ (1,4), (1,6),$
$(2,1), (2,5),$
$(3,3),$
$(4,1), (4,5),$
$(5,2),$
$(6,4), (6,6),$
$(7,1), (7,5) \}$

รูปที่ 2.6 ตัวอย่างการเก็บข้อมูลเพื่อค้นหา New LCS ของ X. Xiang

กำหนดให้ P = {(i, j) : i_i=j_j} แล้ว (i, j) เป็นคู่ลำดับตำแหน่งของตัวอักษรที่เหมือนกัน (Match pair) จากสายอักษร I และ J ซึ่งอาจเป็น p_x ที่สอดคล้องกับตัวอักษรที่เกิดขึ้นทั้งหมด โดยแสดงในรูปแบบของกลุ่มคู่ลำดับ P = {p_1, p_2, ... , p_l} โดยที่แต่ละ p_k เป็นคู่อันดับของ (i_k, j_k) การสร้างเซ็ตของคู่ลำดับ จะกำหนดให้คู่ลำดับของอักษรที่เหมือนกันจำนวนสองคู่ p_x, p_y, ถ้า $i_x < i_y$, $j_x < j_y$ เมื่อกำหนดให้ $p_x < p_y$, p_y เป็นลำดับที่อยู่ตามหลัง (sub-sequence) ของลำดับ p_x หรือ p_x เป็นลำดับที่อยู่ก่อนหน้า (pre-sequence) ของ p_y ตัวอย่างเช่น รูปที่ 2.6 แสดงการหา MLCS ของข้อมูล I = ATCTGAT และ J = TGCATA

1. การเตรียมข้อมูล (Pre-processing) เก็บในอาเรย์ 1 มิติ

P = {(1,4), (1,6), (2,1), (2,5), (3,3), (4,1), (4,5), (5,2), (6,4), (6,6), (7,1), (7,5)}

2. การหา LCS จาก P

P_1 = {(1,4), (2,5), (6,6)}

P_2 = {(1,6)}

P_3 = {(2,1), (3,3), (4,5), (6,6)}

P_4 = {(2,1), (3,3), (6,4), (7.5)}

P_5 = {(4,1), (5,2), (6,4), (7,5)}

P_6 = {(4,1), (5,2), (6,6)}

P_7 = {(7,1)}

ในที่นี้ จะได้ LCS = max{P_i} คือ P_3, P_4, P_5

วิธีการค้นหา LCS ในขั้น 2 จะเริ่มโดยสร้างเซตคู่ลำดับ P{(i, j): i_i=j_j} ตามลำดับตัวอักษรของ I จากนั้นสร้างชุดลำดับตั้งแต่ P_1 จนไปถึง P_l โดยที่ l แสดงถึงจำนวนคู่ระหว่างสายอักษรทั้งสองลำดับ และ P_k={p_z: p_z<*p_{z+1}, z = k, k+1, k+2, ... , l} จากนั้น LCS คือเซตคู่ลำดับที่มีความยาวที่สุด LCS(I, J) = max{P_i}

2.3.2 วิธีการค้นหา SA-MLCS

1. add source S into New_0 (in level 0).
2. set init_layer =0 and cur_layer =0.
3. for all nodes in New_{cur_layer} generate their all children and add to $New_{cur\_layer+1}$.
   if cur_layer≠0 then
   - mark Boolean_array of parents with 1.
   - if all children are expanded, then
     move parent to $Closed_{cur\_layer-1}$;
     else resort parent node in $Open_{cur\_layer-1}$.
4. for first i nodes in $Open_{cur\_layer}$ where i=min(c,|$Open_{cur\_layer}$|); c= widened beam generate unexpanded children in $New_{cur\_layer+1}$.
5. move all nodes in New_{cur_yaler} to $Open_{cur\_layer}$.
6. reserve first i nodes in $New_{cur\_layer+1}$, where i=min(c,|$New_{cur\_layer+1}$|) and delete others.
7. if cur_layer is not the last layer, then
   - if init_layer = cur_layer and |$Open_{cur\_layer}$| =|New_{cur_layer}|=0, then init_layer=init_layer+1.
   - cur_layer=cur_layer+1 and go to step 3 to continue this iteration.
8. if cur_layer is the last layer, then
   - output new solution by tracking back nodes in cur_layer if cur_layer > length of $current_{solution}$.
   - if init_layer=cur_layer and |$Open_{cur\_layer}$| =|New_{cur_layer}|=0, then terminate.
   - otherwise, cur_layer = init_layer and go to step 3 to start new iteration.

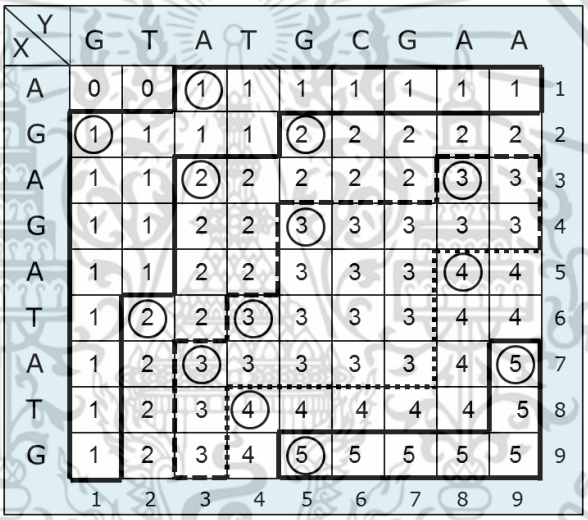
รูปที่ 2.7 ขั้นตอนวิธี SA-MLCS ของ J.Yang [7]

ขั้นตอนวิธี SA-MLCS ของ J.Yang และคณะ [7] จะจัดเตรียมข้อมูลก่อนค้นหา LCS โดยการตรวจสอบข้อมูลทั้งหมด (Pre-processing) เช่นเดียวกับขั้นตอนวิธี DP ตัวอย่างเช่น สำหรับข้อมูลสายอักษร I = AGAGATATG และ J = GTATGCGAA รูปที่ 2.8 แสดงการคำนวณค่าความยาวร่วมทั้งหมด และการค้นหาจุดที่เป็น Dominant point คือจุดที่มีอักษรเหมือนกันและเป็นจุดที่เกิดขึ้นก่อนในแนวแกน X และแกน Y จากนั้นจึงนำจุดดังกล่าวที่ได้มาสร้างเป็นกราฟ (ดังแสดงในรูปที่ 2.9) แล้วจึงทำการค้นหา LCS หลายชุด (MLCS) ในเส้นทางของกราฟที่ถูกนำมาสร้างขึ้น โดยโหนดของกราฟที่ใช้ในการค้นหา จะถูกจัดกลุ่มเป็นระดับชั้น (layers) ด้วยระยะห่างจากโหนดต้นทาง สำหรับแต่ละระดับชั้น ในขั้นตอนวิธี SA-MLCS ดังกล่าว จะมีการใช้โครงสร้างข้อมูลแบบคิว 3 คิว (New, Open, และ Closed queues) ซึ่งมีนิยามการสร้างดังนี้

New queue คือ คิวที่จัดเก็บโหนดที่สามารถสร้างโหนดลูกได้

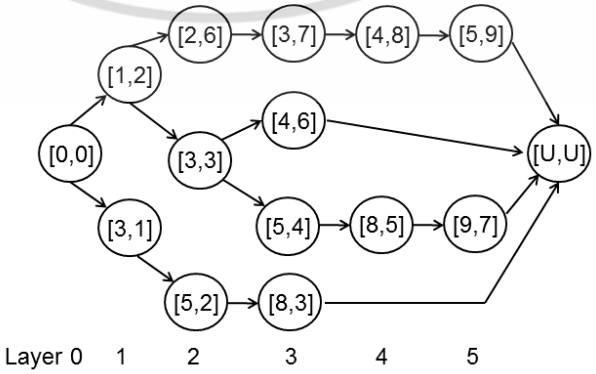
Open queue คือ คิวของโหนดที่ยังไม่ได้สร้างลูกทั้งหมด

Closed queue คือ คิวของโหนดที่ทำการสร้างลูกแล้วทั้งหมด



รูปที่ 2.8 ตัวอย่างการเก็บข้อมูลเพื่อค้นหา SA-MLCS ของ J.Yang

รูปที่ 2.9 แสดงการเก็บข้อมูลในรูปแบบกราฟที่ได้จากการประมวลผลขั้นตอน Pre-processing เพื่อลดพื้นที่ในการจัดเก็บข้อมูลในการประมวลผลขั้นตอนวิธี



รูปที่ 2.9 ตัวอย่างการเก็บข้อมูลในลักษณะของกราฟของ J.Yang

รูปที่ 2.10 แสดงการค้นหา Multiple LCS (MLCS) จากโครงสร้างข้อมูลกราฟ (รูปที่ 2.9) ด้วยการสร้างโครงสร้างข้อมูลแบบคิว 3 คิวคือ New, Open, และ Closed ซึ่งดำเนินการด้วยขั้นตอนวิธี SA-MLCS เป็นจำนวน 4 iterations และได้ LCS 2 ชุด คือ GAGAA (จาก Iteration 1: (1,2),(3,3),(5,4),(8,5),(9,7),) และ GTATG (จาก Iteration 3: (1,2),(2,6),(3,7),(4,8),(5,9))

Iteration 1

	Closed	Open	New
Layer 0		[0,0]	~~[0,0]~~
Layer 1		[1,2]	~~[1,2]~~ ~~[3,1]~~
Layer 2		[3,3]	~~[3,3]~~ ~~[2,6]~~
Layer 3	[5,4]		~~[5,4]~~ ~~[4,6]~~
Layer 4	[8,5]		~~[8,5]~~
Layer 5	[9,7]		~~[9,7]~~

Iteration 2

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1]	[1,2]	~~[3,1]~~
Layer 2	[3,3]	[5,2]	~~[5,2]~~
Layer 3	[5,4] [4,6]		~~[4,6]~~ ~~[8,3]~~
Layer 4	[8,5]		
Layer 5	[9,7]		

Iteration 3

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1] [1,2]		
Layer 2	[3,3] [2,6]	[5,2]	~~[2,6]~~
Layer 3	[5,4] [4,6] [3,7]		~~[3,7]~~
Layer 4	[8,5] [4,8]		~~[4,8]~~
Layer 5	[9,7] [5,9]		~~[5,9]~~

Iteration 4

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1] [1,2]		
Layer 2	[3,3] [2,6] [5,2]		
Layer 3	[5,4] [4,6] [3,7] [8,3]		~~[8,3]~~
Layer 4	[8,5] [4,8]		
Layer 5	[9,7] [5,9]		

รูปที่ 2.10 ตัวอย่างการค้นหา MLCS ในขั้นตอนวิธีของ J.Yang

2.3.3 ขั้นตอนวิธี Quick-DP และ Quick-DPPAR [9]

ขั้นตอนวิธี Quick-DP เป็นขั้นตอนวิธีที่ทำการปรับปรุงจากขั้นตอนวิธี Dominant point เดิม ซึ่งประกอบด้วยการประมวลผล 2 ส่วน ในส่วนแรกเซตของ dominant points ทั้งหมดจะถูกคำนวณในแต่ละ iteration โดยเริ่มจาก dominant point ที่ 0 (มีอยู่ 1 สมาชิก) เซตของ dominants ที่ k+1 หรือ $D^{(k+1)}$ จะสามารถหาได้จากเซตของ dominants ที่ k หรือ $D^{(k)}$ และในส่วนที่ 2 จะค้นหาเส้นทางที่เป็น LCS ด้วยการคำนวณแบบย้อนกลับ (โดยจะเริ่มจากสมาชิกสุดท้าย) ผ่านเซตของ dominant points ที่ได้มาจากส่วนแรกของขั้นตอนวิธี

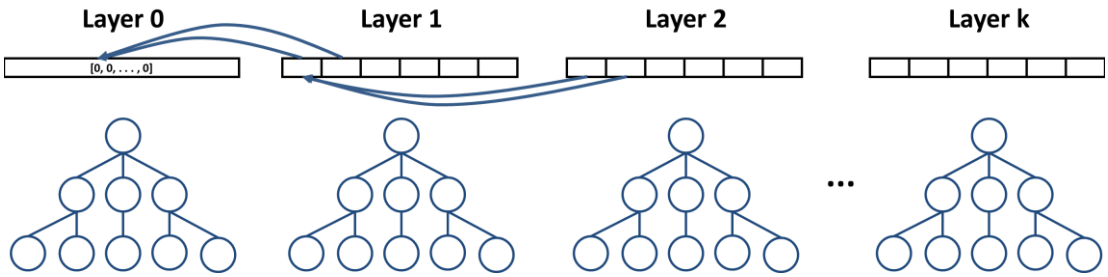
```
Algorithm Quick-DPPAR({a1, a2, . . . , ad}, Σ, Np)
1  Proc0 : Preprocessing; D0={|-1,-1,…,-1|};k=0;
2  while Dk not empty do {
3     Proc0 : distribute element of Dk
      Each processor, Proci, 1 ≤ i ≤ Np, performs :
4        get D_i^k from Proc0;
5        for q ∈ D_i^k  do {
6           B = Minima(Par(q, Σ));
7           for s ∈ Σ do{
8               Paris = Paris ∪ {q(s) | q(s) ∈ B};}}
9        Send Paris, s ∈ Σ, to Proc0;
10 Proc0 : calculate Pars = ∪_{1≤i≤Np} Paris, s ∈ Σ;
11 Proc0 : distribute Pars, s ∈ Σ;
     Each processor, Proci, 1 ≤ i ≤ Np, perform :
12    get Pars, s ∈ Σ;
13    D_i^{k+1}= Minima(Pars);
14    send D_i^{k+1} to Proc0;
15 Proc0 : defines Dk+1 = ∪_{1≤i≤Np} D_i^{k+1};
16 k = k-1; }
```

รูปที่ 2.11 ขั้นตอนวิธี Quick-DPAR แบบขนาน [9]

ขั้นตอนวิธีแบบขนาน Quick-DPPAR [9] ได้ถูกพัฒนาขึ้นโดยการใช้เทคนิควิธี Divide-and-Conquer โดยให้โพรเซสเซอร์ $Proc_0$ แบ่งเซตของ N dominant points ออกเป็นสองซับเซตย่อย Q และ R จากนั้นทำการส่งมอบงานไปที่ $Proc_q$ และ $Proc_r$ เพื่อทำการประมวลผลหาค่า Minima โดย $Proc_q$ และ $Proc_r$ สามารถมีหน่วยประมวลผลที่แบ่งต่อไปได้เช่นเดียวกัน ดังนั้นในขณะที่ทำการประมวลผลซ้ำ โครงสร้างแบบ Binary tree จะถูกสร้างขึ้นจากความสัมพันธ์แบบแม่-ลูก และในกรณีที่กำหนดให้มีหน่วยประมวลผล p หน่วย ความลึกของโครงสร้างต้นไม้เป็น $\log_2 p$ โดยโหนดที่เป็นใบ (Leaf nodes) จะเป็นจุดเริ่มต้นในการค้นหา LCS แบบย้อนกลับจากฟังก์ชัน Divide-and-Conquer ที่จะทำได้พร้อมๆ กัน

2.3.4 ขั้นตอนวิธี PRO-MLCS และ DPRO-MLCS [10]

Pro-MLCS เป็นขั้นตอนวิธีที่ถูกปรับปรุงมาจากขั้นตอนวิธีแบบ Dominant point เดิม เช่นเดียวกับ Quick-DP ร่วมกับฮิวริสติกที่เป็นแบบ Best-first search เพื่อนำมาใช้ในการค้นหา LCS จากข้อมูลโครงสร้างต้นไม้ที่ถูกสร้างขึ้นจาก dominant points ในแต่ละเลเยอร์ (layer) จะทำการเก็บข้อมูลในคิวแบบ priority queue 1 คิวที่บรรจุโหนดในเลเยอร์เดียวกัน ในแต่ละ iteration ขั้นตอนวิธีจะทำการขยายโหนดในแต่ละคิวแบบ priority queue ในขั้นตอนสุดท้ายของแต่ละรอบขั้นตอนวิธีจะสร้างคำตอบในเบื้องต้นที่เป็น Approximate solution และเมื่อขั้นตอนวิธีดำเนินการจนเสร็จสิ้นจะเป็นคำตอบที่เป็น Optimal solution



รูปที่ 2.12 โครงสร้างข้อมูลที่ใช้ในขั้นตอนวิธี Pro-MLCS [10]

รูปที่ 2.12 แสดงโครงสร้างข้อมูลที่ใช้ในขั้นตอนวิธี Pro-MLCS ที่แต่ละเลเยอร์มีการเก็บรักษา priority queue และต้นไม้ d-index และด้วยโครงสร้างข้อมูลทั้งสองชนิดนี้ถูกใช้เก็บข้อมูล point ที่จะถูกขยายและ point ที่ถูกขยายแล้ว priority queue ของเลเยอร์ 0 จะถูกสร้างไว้ในขณะเริ่มต้นและเก็บจุด R = [0,0, ..., 0]

ขั้นตอนวิธีแบบขนานโดยทั่วไปมักมีข้อจำกัดในการประมวลผลข้อมูลเฉพาะในเลเยอร์เดียวกัน และมีการแบ่งภาระงานไปยังแต่ละหน่วยประมวลผลที่ไม่เท่ากัน ตลอดจนไม่สามารถใช้ประโยชน์จากการมีหน่วยประมวลผลเป็นจำนวนมากได้ ดังนั้นขั้นตอนวิธี DPro-MLCS [10] จะทำการกำหนดการประมวลผลเลเยอร์ให้กับแต่ละหน่วยประมวลผล เพื่อให้ได้รับภาระงาน (Workload) ที่เท่าๆ กัน และจากการพิจารณาโครงสร้างของ Dominant points ที่มีการกระจายตัวที่หลากหลาย งานวิจัยนี้จึงได้นำวิธีการกำหนดภาระงานแบบหมุนเวียน ที่เรียกว่า Cyclic assignment method ที่จะกำหนดให้เลเยอร์ที่ pk+i กับหน่วยประมวลผล i (P_i) เมื่อ p เป็นจำนวนหน่วยประมวลผลทั้งหมด

อย่างไรก็ตามขั้นตอนวิธีแบบขนานทั้งสองวิธี [9-10] มีการดำเนินการที่เร็วขึ้นสำหรับข้อมูลหลายชุด (X_1, X_2, ... , X_s) แต่จะเริ่มการประมวลผลแบบขนาน หลังจากขั้นตอนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) ในแต่ละคู่ของข้อมูล (X, Y) ซึ่งเป็นเวลาส่วนหนึ่งของการประมวลผล LCS ที่ใช้เวลานานเท่ากับ $O(n^2)$ เนื่องจากขั้นตอนเตรียมข้อมูล จะทำการประมวลผลแบบขนาน ได้ค่อนข้างยาก เพราะค่าของแต่ละสมาชิก จะต้องคำนวณจากผลลัพธ์ที่ต่อเนื่องกัน

บทที่ 3
การหาลำดับร่วมเหมือนที่ยาวที่สุดด้วยวิธีไดนามิคโปรแกรมมิงแบบลดพื้นที่

ในบทที่ 3 นี้จะกล่าวถึงขั้นตอนวิธี DP ใหม่ ที่ใช้ในการค้นหา LCS (Longest Common Subsequence) ชื่อ IDPSR (Improved DP with Space Reduction) ที่ปรับปรุงมาจากขั้นตอนวิธี DP เดิมที่มีประสิทธิภาพ เนื่องจากในขั้นตอนของการจัดเตรียมข้อมูล (Pre-processing) ของวิธี DP เดิม จะใช้เมตริกซ์ขนาด mxn หรืออาเรย์ 2 มิติ ซึ่งเป็นข้อจำกัดในทางปฏิบัติเมื่อ m,n มีค่ามากๆ เพราะตัวแปรอาเรย์ขนาด mxn ต้องใช้พื้นที่จัดเก็บค่าติดกันในหน่วยความจำในขณะประมวลผลโปรแกรม ซึ่งปกติจะจองเนื้อที่อาเรย์ได้ไม่เกิน 800x800 (m, n < 800)

ดังนั้นงานวิจัยนี้ จึงนำเสนอขั้นตอนวิธี IDPSR ที่มีการใช้หน่วยความจำแบบ Dynamic เพื่อแก้ปัญหาดังกล่าว ด้วยโครงสร้างข้อมูลแบบอาเรย์ของลิสต์ชนิดพิเศษคือ Mlist L_i (i = 1, 2,..., n) ที่มีขนาดของลิสต์เป็นแบบ Dynamic (≤ k) เมื่อค่า k เป็นความยาวของ LCS และเสนอขั้นตอนวิธี Improved DP ที่มีประสิทธิภาพยิ่งขึ้น โดยปรับและเพิ่มฟังก์ชันให้สอดคล้องกับการประมวลผลข้อมูลที่จัดเก็บแบบ Mlist ซึ่งแบ่งเป็น 2 ส่วนคือ 1. การจัดเตรียมข้อมูล Mlist (เสนอในหัวข้อ 3.1) และ 2. การค้นหา LCS และ MLCS จาก Mist (เสนอในหัวข้อ 3.2)

3.1 ขั้นตอนวิธีการเตรียม Dynamic Mlist

ขั้นตอนวิธี 3.1 เป็นการสร้าง Mlist ที่เป็นการจัดเตรียมข้อมูล (Pre-processing) ด้วยวิธีการคำนวณค่าความยาวร่วมทั้งหมดเช่นเดียวกับขั้นตอน DP-LCS แต่ในที่นี้ จะจัดเก็บลงใน Mlist เฉพาะค่าใหม่ที่ไม่ซ้ำ (เพื่อลดพื้นที่ และเพื่อค้นหา LCS และ MLCS ได้เร็วขึ้น) พร้อมตำแหน่งของ X และตัวชี้ back

```
for (x=1; x<=m; x++)
  for(y=1; y<=n; y++)
    if(X[x]=Y[y]) c[y] = L[y-1]->c+1;
    else c[y] = max(c[y-1], L[y]->c);
  end for y
  UpdateMlist (Ln, cn, x);
end for x
return ImprovedMlist (Ln, n, k);
[Note: Each node of Mlist L has 3 fields (c, x, back)]
```

รูปที่ 3.1 ขั้นตอนวิธีการสร้าง Dynamic Mlist ของ IDPSR-LCS

การดำเนินขั้นตอนวิธีการสร้าง Dynamic Mlist ของ IDPSR-LCS ประกอบไปด้วยขั้นตอนดังต่อไปนี้

1) นำสายอักษรที่ต้องการเปรียบเทียบ 2 ชุด กำหนดให้ชุดแรก มีค่า X และชุดที่สองมีค่า Y

2) ให้ความยาวของสายอักษรชุดแรกแทนค่าด้วย m และความยาวของสายอักษร ชุดที่สองแทนด้วยค่า n

3) กำหนดค่า x เริ่มต้นเป็น 1

4) กำหนดค่า y เริ่มต้นเป็น 1

5) ทำการวนลูปตรวจสอบจากตัวอักษรที่ y จากสายอักษร Y ตั้งแต่ y มีค่าเท่ากับ 1 ไปจนถึง n

6) ทำการเปรียบเทียบตัวอักษรตัวที่ x จากสายอักษร X กับตัวอักษร y จากสายอักษร Y ว่ามีค่าเหมือนกันหรือไม่

ถ้าเหมือน ให้กำหนดค่าอาร์เรย์ c ในตำแหน่งที่ y มีค่าเท่ากับค่าจากตำแหน่ง x-1, y-1 ทำการบวกค่า 1

ถ้าไม่เหมือน ให้กำหนดค่าอาร์เรย์ c ในตำแหน่งที่ y มีค่าเท่ากับค่าจากตำแหน่ง x-1, y หรือ x, y-1 โดยเลือกค่าที่มากกว่า

7) เมื่อวนลูปตรวจสอบจากตัวอักษรจากสายอักษร Y จนครบแล้ว จึงทำการตรวจสอบเพื่ออัพเดทค่าใน Mlist

8) ทำซ้ำข้อ 5-7 โดยเพิ่มค่า x ไปอีก 1 และกำหนดให้ค่า y กลับมาเริ่มต้นที่ 1 ไปจนถึง n จนกว่า x จะมีค่าเท่ากับ m

9) ทำการปรับปรุง Mlist เพื่อลดพื้นที่ในการจัดเก็บข้อมูลลง

UpdateMlist (L_n, c_n, x)

```
for(y=1; y<=n; y++)
  if(c[y] != L[y]->c)          // add new node in L[y]
    allocate new node (N); // memory allocate
    N->c=c[y]; N->x=x; N->back=L[y]; L[y]=N;
  end if
end for y
```

รูปที่ 3.2 ขั้นตอนการอัพเดท Mlist ในแต่ละ Iteration

การดำเนินขั้นตอนการอัพเดท Mlist ในแต่ละ Iteration ประกอบไปด้วยขั้นตอนดังต่อไปนี้

1) กำหนดค่า y เริ่มต้นเป็น 1

2) ทำการเปรียบเทียบค่าอาร์เรย์ c ที่ตำแหน่ง y ว่ามีค่าเท่ากับค่า c ของของลิสต์ที่ตำแหน่ง y หรือไม่ ถ้ามีค่าไม่เท่ากัน ให้ทำการเพิ่มโหนดใหม่ลงไปในลิสต์ โดยให้

ค่า c ของโหนดใหม่มีค่าเท่ากับค่า c ในอาร์เรย์ที่ตำแหน่ง y

ค่า x ของโหนดมีค่าเท่ากับ x

ค่า back ชี้ไปที่ L[y] หรือโหนดก่อนหน้า

ค่า L[y] เป็นโหนดที่สร้างขึ้นใหม่

ImprovedMlist (L_n, f, k)

```
y=f; while(L[y]->c = k) y=y-1;  // min y with LCS=k
f=y; for(y=f; y>0; y--)
        c=L[y]; p=L[y+1]; // c=current, p=previous
        while(c->x > p->x) // adjust tail & free node
           L[y]=c->back; free(c); c=L[y];
        end while
      end for y;
return f+1;
```

รูปที่ 3.3 ขั้นตอนการปรับปรุง Mlist ก่อนทำการค้นหา LCS

การดำเนินขั้นตอนการปรับปรุง Mlist ประกอบไปด้วยขั้นตอนดังต่อไปนี้

1) กำหนดให้ค่า y เท่ากับค่า f(ค่า n)

2) วนลูปตรวจสอบปลายลิสต์ว่ามีค่าเท่ากับค่าที่มากที่สุดหรือไม่ ถ้าใช่ ให้ลบค่า y ออก 1 หยุดเมื่อเจอลิสต์ที่มีปลายลิสต์น้อยกว่า k

3) ให้ค่า f เท่ากับค่า y โดย y เป็นตำแหน่งที่พบค่า c ที่น้อยกว่า k ,

3.1) ให้ค่า y เริ่มต้นเท่ากับ f ทำซ้ำเมื่อค่า y ยังมากกว่า 0 เพื่อเริ่มตรวจสอบจากปลายลิสต์

3.2) ให้ค่า c ชี้ไปยังตำแหน่งของปลายโหนดลิสต์ปัจจุบัน (y)

3.3) ค่า p ชี้ไปยังตำแหน่งของปลายโหนดลิสต์ก่อนหน้า (y+1)

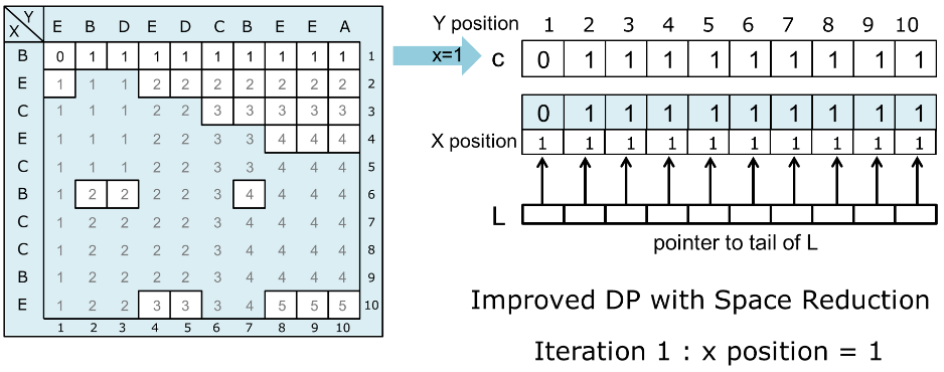
3.4) วนลูปตรวจสอบถ้าค่า x ของปลายโหนดลิสต์ปัจจุบันมีค่ามากกว่าค่า x ของปลายโหนดลิสต์ก่อนหน้า ให้ปลายโหนดของลิสต์ถูกเปลี่ยนเป็นโหนดก่อนหน้านั้นในลิสต์ และนำพื้นที่หน่วยความจำคืนให้กับระบบ

4) ทำซ้ำจนกว่าค่า y เท่ากับ 0

ให้ c[y] แทนค่าความยาวของสายอักษรร่วมจากสายอักษร X และ Y ดังนั้นสำหรับแต่ละค่าของ x (= 0,1,2,...,m) และทุกค่า y (= 0,1,2,..., n) คำนวณ c[y] ก่อนจัดเก็บบางค่าใน Mlist L[y]

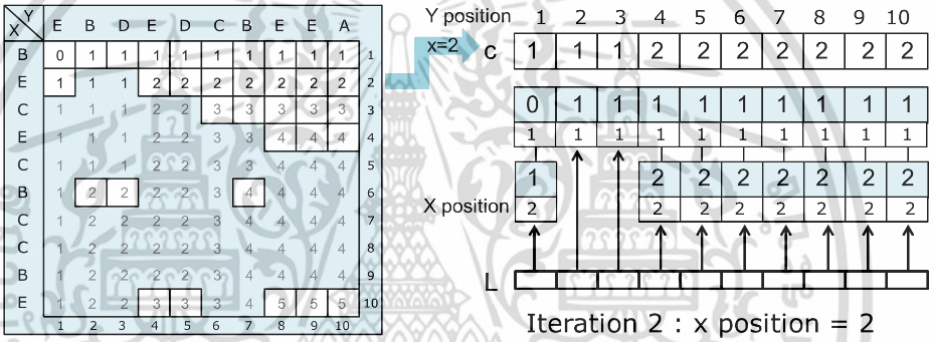
$$c[y] = \begin{cases} \text{ค่าใน } L[y-1]+1 & \text{ถ้า } X[x] = Y[y] \\ \max\{c[y-1], L[y]\} & \text{ถ้า } X[x] \neq Y[y] \end{cases}$$

ตัวอย่างเช่น รูปที่ 3.4-3.7 แสดงการสร้าง Mlist L ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA ดังนี้ ใน Iteration 1 (พิจารณา X ที่ตำแหน่ง x=1) ทุกค่าของ c[y], y=1,2,...,m จะถูกจัดเก็บใน L[y] ทุกค่าของ y พร้อมตำแหน่งแรกของ x=1



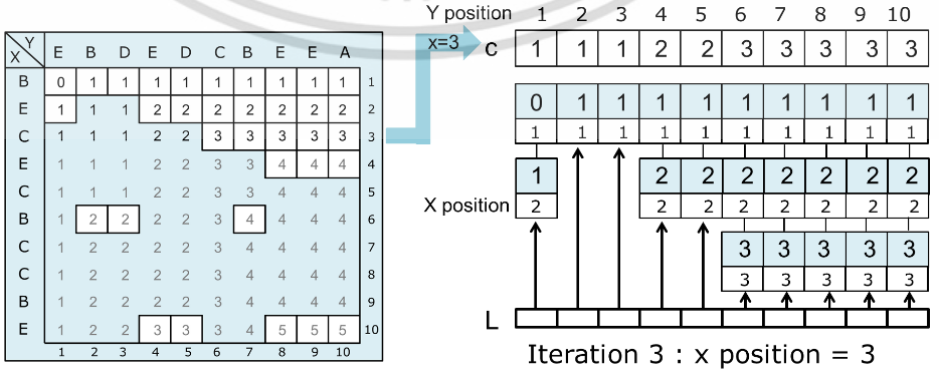
รูปที่ 3.4 การเก็บข้อมูลของขั้นตอนวิธี IDPSR-LCS ใน Iteration 1

ต่อมาใน Iteration 2 (x=2) จะมีเพียงบางค่าใหม่ของ c[y] ที่แตกต่าง และถูกจัดเก็บเพิ่มใน L[y] (คือ y=1,4-10)

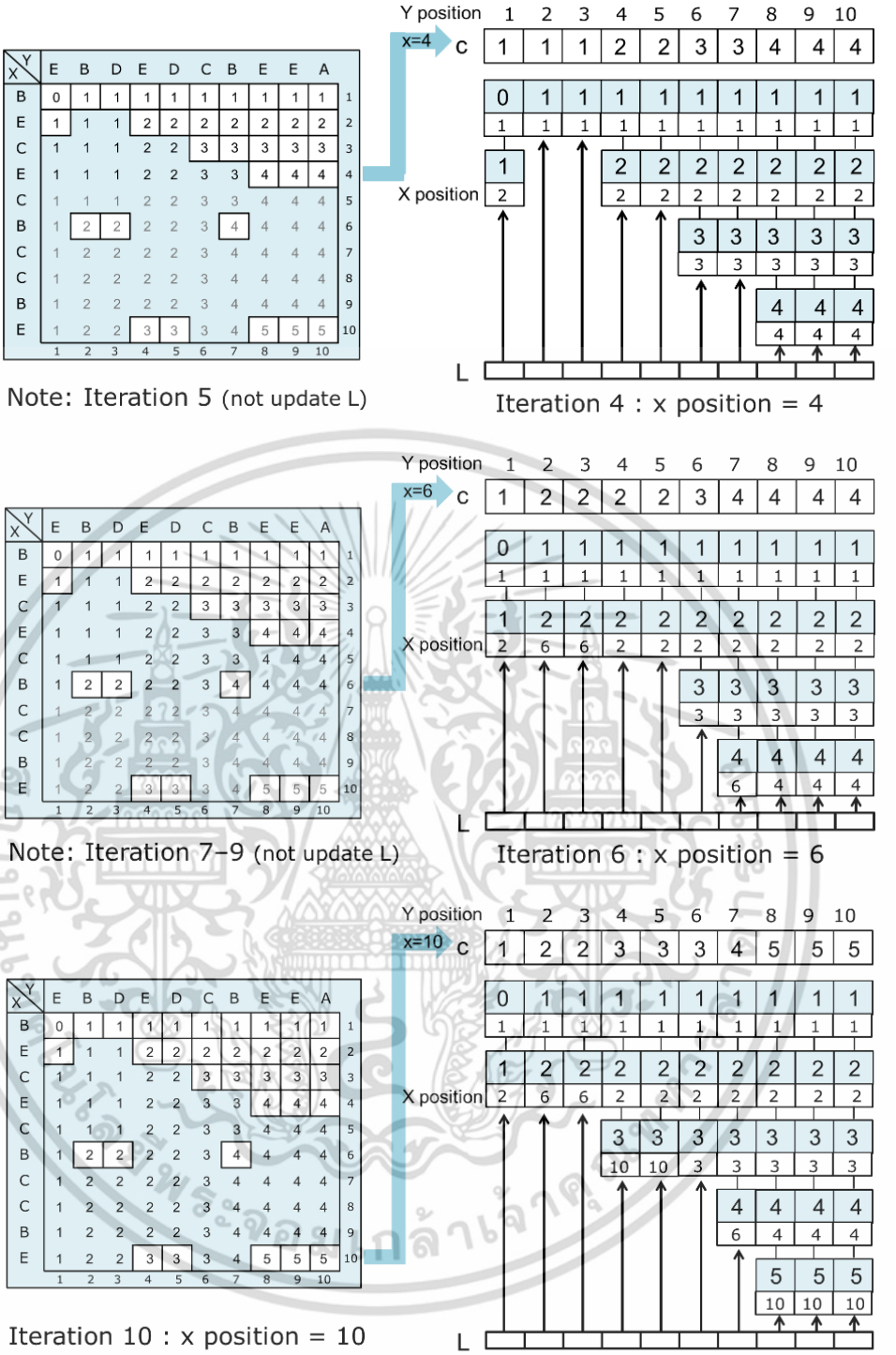


รูปที่ 3.5 การเก็บข้อมูลของขั้นตอนวิธี IDPSR-LCS ใน Iteration 2

และในทำนองเดียวกัน จะดำเนินการซ้ำใน Iteration 3-10 (ซึ่งโดยปกติ เมื่อ Iteration เพิ่มขึ้น ค่าไม่ซ้ำที่จัดเก็บในลิสต์จะมีน้อยลง เช่น Iteration 3 (เก็บเพิ่ม 5 ค่า), Iteration 4 (เก็บเพิ่ม 3 ค่า) และในบาง Iterations ไม่ต้องเก็บค่า(ใหม่)เพิ่ม เช่น Iterations 5, 7-9)

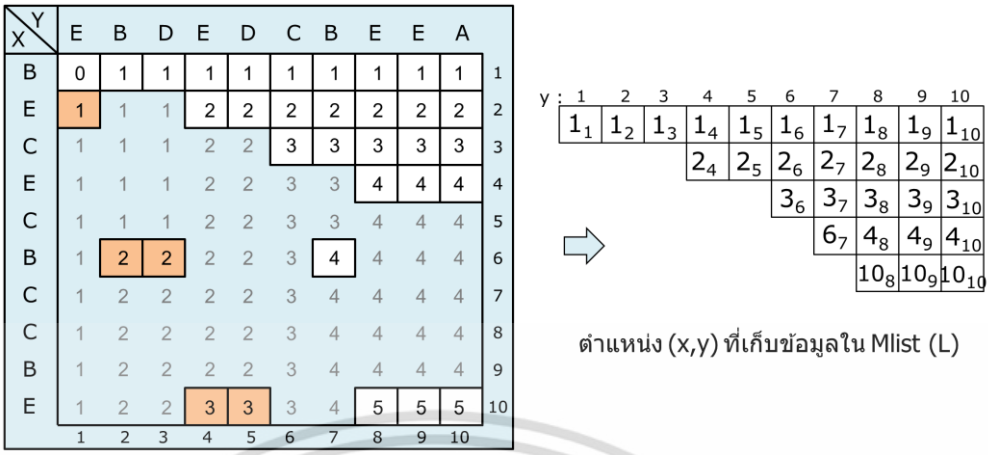


รูปที่ 3.6 การเก็บข้อมูลของขั้นตอนวิธี IDPSR-LCS ใน Iteration 3



รูปที่ 3.7 การเก็บข้อมูลของขั้นตอนวิธี IDPSR-LCS ใน Iteration 4-10

ดังนั้นการเก็บเฉพาะค่าที่ไม่ซ้ำดังกล่าว ทำให้สามารถลดพื้นที่จัดเก็บได้ และฟังก์ชันสุดท้าย (ImprovedMlist) คือการปรับ Mlits L เพื่อตัดบางโหนดที่ไม่ใช่ส่วนของ LCS ออก ดังแสดงในภาพที่ 3.8 เช่น โหนดในตำแหน่ง (x, y) = (10,5), (10,4), (6,3), (6,2), (2,1) ทำให้สามารถลดพื้นที่ได้ถึง 72% (ในตัวอย่างนี้) ซึ่งใช้พื้นที่จัดเก็บข้อมูลเพียง 28% และการเก็บแบบ Mlist ไม่จำเป็นต้องใช้พื้นที่ติดกันในหน่วยความจำ



รูปที่ 3.8 ตัวอย่างการเก็บข้อมูลแบบ Dynamic ใน Mlist ที่ลดลง 72%

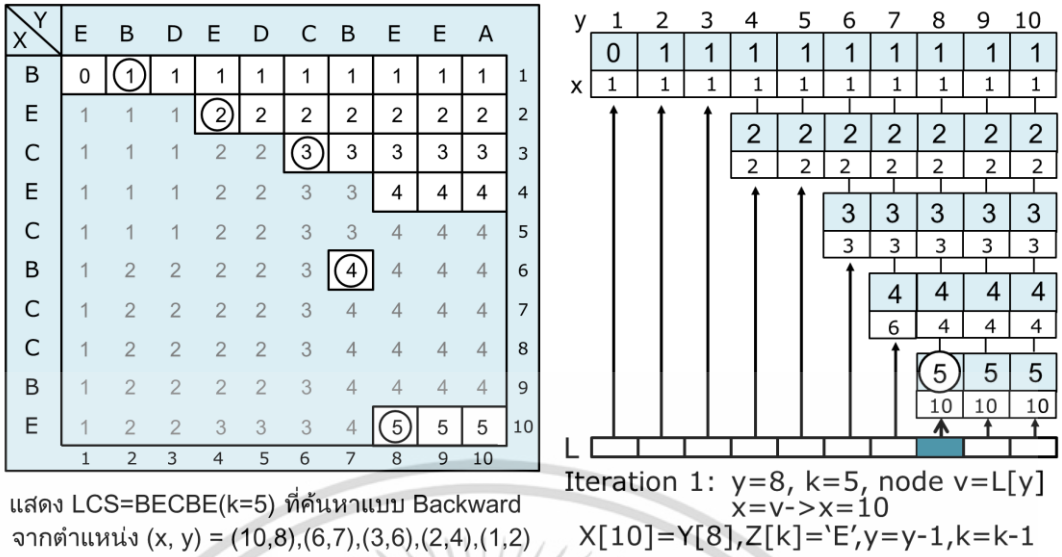
3.2 ขั้นตอนวิธีค้นหา LCS และ MLCS จาก Dynamic Mlist

ขั้นตอนวิธี 3.2 เป็นการค้นหา LCS จาก Mlist (ในขั้นตอนวิธี 3.1) โดยมี Input คือ list L_n, strings X_m, Y_n, ความยาว LCS = k พร้อมตำแหน่ง x, y และ Output คือ LCS Z_k ให้ v แทนโหนดใดๆ ที่เริ่มจาก tail ของ list L[y] (คือ v=L[y])

```
BackLCSfromMlist (Ln,Xm,Yn,Zk,x,y,k)
  while (k>0) do
    v=L[y]; x=v->x;
    if((X[x]=Y[y])&(v->c=k)) Z[k]=X[x];k=k-1;y=y-1;
    else // either Xx≠Y(of LCS1) or find Xx=Yy(of LCSother)
      while(v->c > k) v=v->back; // for other LCS
      x=v->x;
      if((X[x]=Y[y])&(v->c=k))Z[k]=X[x];k=k-1; y=y-1;
      else y=y-1;   // Xx≠Yy goto next L[y] for k
    end if-else;
  end while;
end.
```

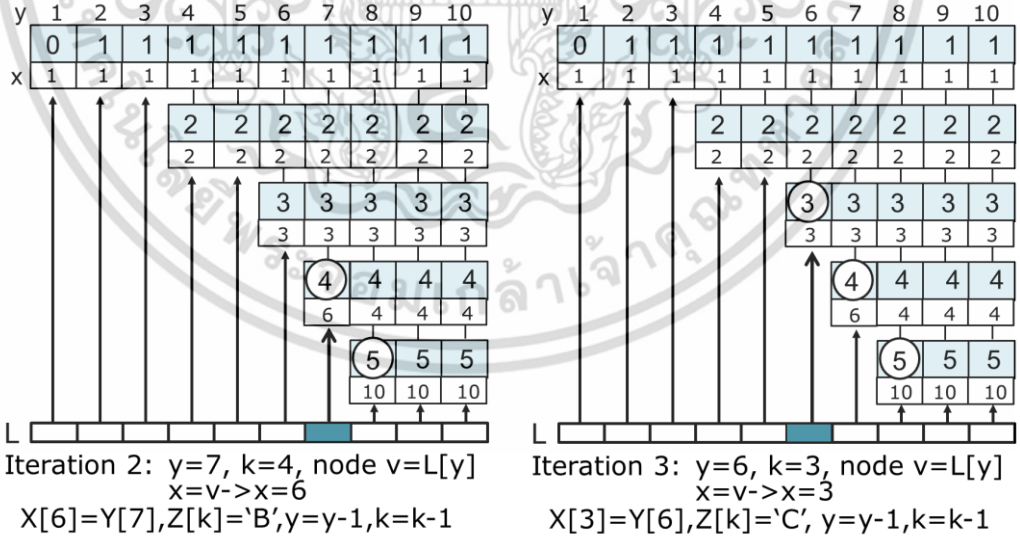
รูปที่ 3.9 การค้นหา LCS จาก Mlist ของวิธี IDPSR-LCS

ตัวอย่างเช่น รูปที่ 3.10 แสดงการค้นหา LCS ค่าแรก ที่เริ่มจาก (x, y)=(10, 8) ของ Mlist L_n ใน Iteration 1 (y=8, k=5) โหนดแรก v=L[y]=L[8] จะได้ x = v->x = 10 และ Z[5]='E'

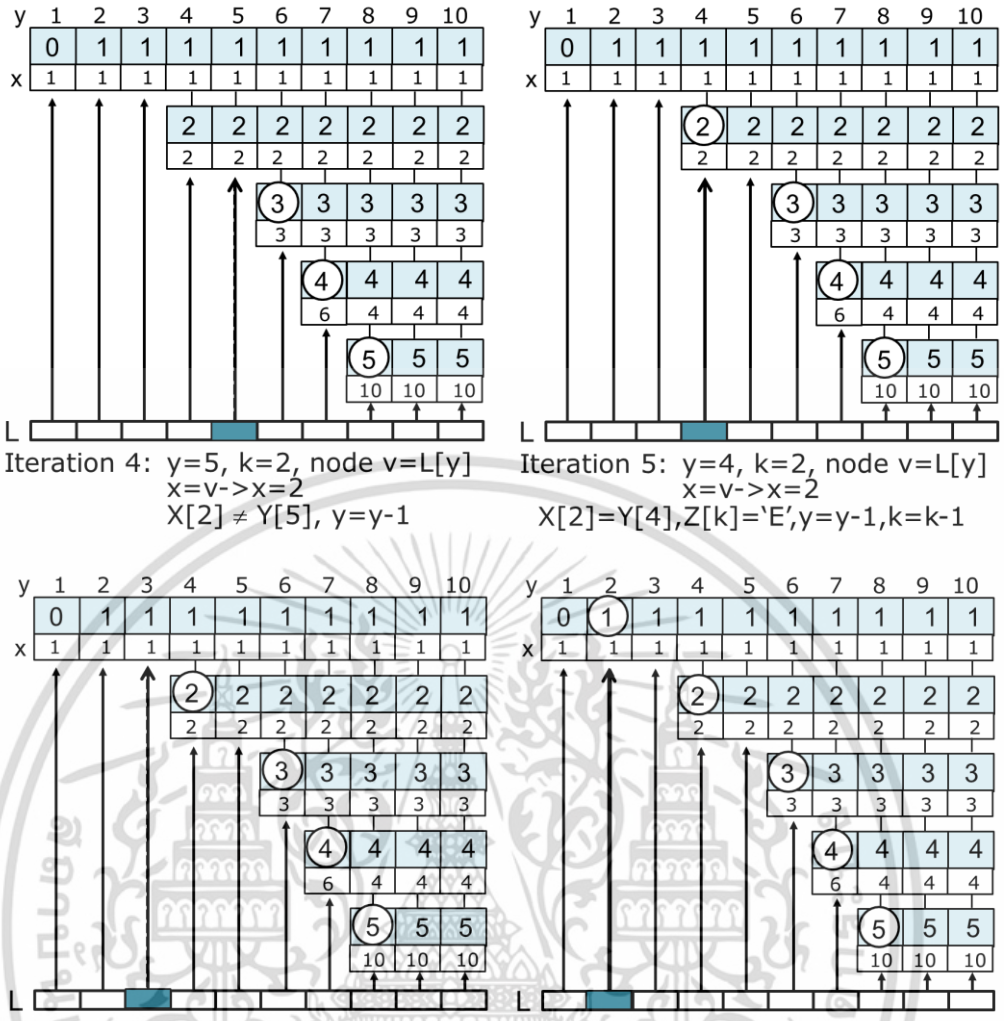


รูปที่ 3.10 การค้นหา LCS ใน Iteration 1

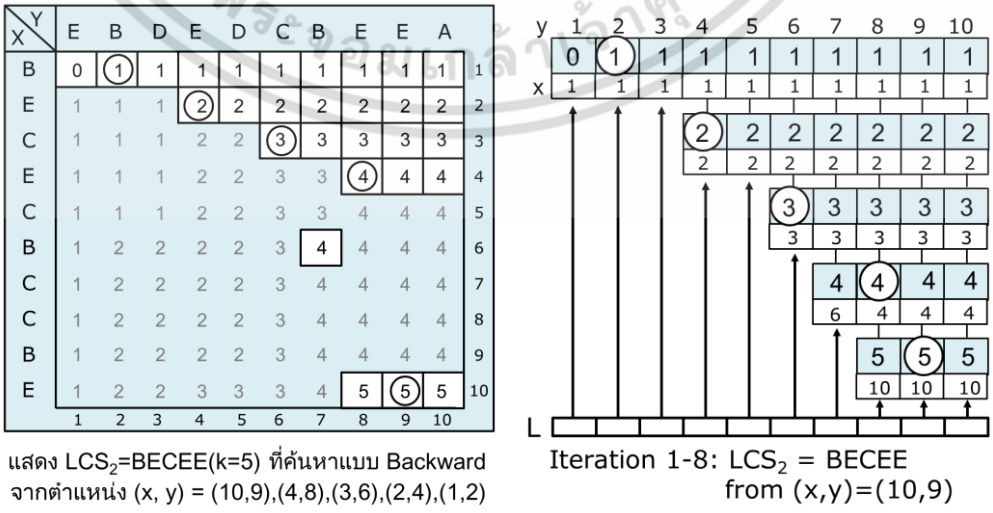
ในทำนองเดียวกัน Iteration 2 (y=7, k=4) จะได้ Z[4]='B' และ Iteration 3 (y=6, k=3) จะได้ Z[3]='C' ส่วน Iteration 4 (y=5, k=2) โหนด v=L[y]=L[5] และ x=v->x=2 แต่กรณีนี้ X[2]≠Y[5] จึงข้ามไปยังลิสต์ L[y-1] แต่ยังคงค่า k=2 และจะดำเนินการในทำนองเดียวกันใน Iterations ที่เหลือ ซึ่งจะได้ Z[2]='E' ใน Iteration 5 และ Z[1]='B' ใน Iteration 7 ส่วนรูปที่ 3.13 แสดงการค้นหา LCS อีกค่า ที่เริ่มจาก k=5 ที่ตำแหน่ง (x, y)=(10, 9)



รูปที่ 3.11 การค้นหา LCS ใน Iteration 2-3



รูปที่ 3.12 การค้นหา LCS ใน Iteration 4-7



รูปที่ 3.13 การค้นหา LCS_2

3.3 ผลการทดลองในการลดพื้นที่

ในขั้นตอนการเตรียมข้อมูลของวิธี IDPSR-LCS ของข้อมูล X (ขนาด m), Y (ขนาด n) ที่เสนอในหัวข้อ 3.1 คือ คำนวณทุกค่า mxn ของ c_{ij} (ใน DP เดิม) แต่เก็บชั่วคราวใน c_j และสร้างลิสต์ Mist L_j (ขนาด n) ทุกค่าของ Y คือ j = 1, 2, 3, ... , n เพื่อเก็บเฉพาะค่าที่แตกต่าง(หรือเมื่อพบค่าร่วมใหม่) เพื่อเป็น x-pointer jumping ในค่าของ X (ในขั้นตอนค้นหา LCS) นอกจากนั้นในขั้นนี้ จะเพิ่มฟังก์ชันที่ปรับลิสต์ (ImprovedMlist) ในส่วนท้ายๆ ของทุกลิสต์ ให้เหลือเฉพาะค่าที่เป็น LCS ทำให้เก็บค่าน้อยลง (ประมาณ cn ค่า ดังแสดงในรูปที่ 3.8 เมื่อ c เป็นค่าคงที่) ดังนั้นจึงมีความซับซ้อนด้านเวลา (Time Complexity) = O(mn) และความซับซ้อนด้านพื้นที่ (Space Complexity) = O(n)

ในส่วนการค้นหาสายอักษรร่วมที่ยาวที่สุด (LCSs) ด้วยวิธี IDPSR-LCS จะประมวลผลได้เช่นเดียวกับวิธี DP เดิมที่มีความซับซ้อนด้านเวลา O(n) ในกรณีดีที่สุด และ O(m+n) กรณีช้าที่สุด ส่วนวิธี IDPSR จะค้นหา LCS ด้วยความซับซ้อน O(n) เพราะ LCS ชุดแรก จะพบได้โดยตรงจากปลายลิสต์ ส่วน LCS อื่นๆ จะใช้ค่าต่อไปของแต่ละลิสต์ ที่จะไปถึงได้ด้วย x-pointer jumping ที่จัดเตรียมไว้ในลิสต์ ซึ่งแตกต่างจาก DP เดิม ที่จะต้องเลื่อนไปทีละค่าของอาเรย์ X และ Y

ตาราง 3.1 ความซับซ้อน Time และ Space ของขั้นตอนวิธี LCS

ขั้นตอนวิธี	Pre-processing		LCS searching	
	Time	Space	Time	Space
Original DP-LCS	O(mn)	O(mn)	O(m+n)	O(1)
New LCS 2007 [6]	O(mn)	O(n)	O(ckn)	O(ck)
SA-MLCS 2014 [7]	O(mn)	O(n)	O(kn)	O(ck)
IDPSR-LCS ที่เสนอ	O(mn)	O(n)	O(n)	O(1)

ตารางที่ 3.2 แสดงการเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธีต่างๆ ซึ่งประกอบด้วยขั้นตอนวิธีแบบไดนามิคโปรแกรมมิงเดิม, ขั้นตอนวิธี New LCS 2007, SA-MLCS 2014 และ ขั้นตอนวิธี IDPSR-LCS ซึ่งจะพบว่า ขั้นตอนวิธี IDPSR-LCS สามารถลดพื้นที่ในการจัดเก็บลงในขณะที่สามารถเพิ่มความเร็วในการค้นหา LCS ได้อีกด้วย

ตาราง 3.2 การเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธีต่างๆ

Original DP-LCS	
ข้อดี	สามารถหา LCS ได้โดยตรงจากค่า c_{ij} ซึ่งเร็วมาก O(m+n)
ข้อจำกัด	ใช้พื้นที่มากเพื่อเก็บค่าเมตริกซ์ c เป็นอาเรย์ 2 มิติ (mxn)
New LCS 2007[6]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Matching point ในอาเรย์ 1 มิติ
ข้อจำกัด	ใช้เวลาในการหา LCS O(ckn) นานกว่า DP เพราะต้องตรวจสอบเงื่อนไข pre/post ของ LCS ทุกค่าและ CS อื่นๆ

ตาราง 3.2 การเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธีต่างๆ (ต่อ)

SA-MLCS 2014[7]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Dominant point ในกราฟ
ข้อจำกัด	ใช้เวลาในการหา LCS O(kn) เร็วกว่า [6] เพราะใช้คิว 3 คิว (New, Open, Closed) ช่วยลดการตรวจความสัมพันธ์ที่ซ้ำซ้อน แต่นานกว่า DP เพราะต้องหาความสัมพันธ์ของ LCS ทุกค่า (MLCS) และ CS อื่นๆ ที่เกี่ยวข้อง
IDPSR-LCS ที่เสนอในงานวิจัยนี้	
ข้อดี	มีข้อดีเหมือน DP คือหา LCS ได้โดยตรง แต่เร็วกว่า O(n) มีข้อดีเหมือน [6,7] ที่สามารถลดพื้นที่เพราะเก็บข้อมูลเป็นอาเรย์ 1 มิติของลิสต์แทน (โดยเก็บเฉพาะค่า c_{ij} ที่แตกต่าง)
ข้อจำกัด	ยังคงใช้เวลาเตรียมข้อมูลในส่วน Pre-processing เป็น O(mn) เหมือน DP และวิธีอื่นๆ [6, 7]

ตารางที่ 3.3 แสดงผลการทดลองเปรียบเทียบการลดพื้นที่จัดเก็บข้อมูลของขั้นตอนวิธี IDPSR-LCS ที่สามารถลดพื้นที่ในการจัดเก็บข้อมูลได้ประมาณ 88% เมื่อทดสอบกับสายอักษรที่มีความยาวเท่ากัน (m=n)

ตาราง 3.3 ผลการทดลองเปรียบเทียบการลดพื้นที่จัดเก็บข้อมูล

m=n	k	Matrix C_{mxn}	Mlist	%พื้นทีเก็บ	%พื้นที่ลด
100	22	100x100 = 10,000	1,077	11%	89%
500	120	500x500 = 250,000	29,433	12%	88%
1,000	234	1,000x1,000 = 1,000,000	119,240	12%	88%
10,000	2401	10,000x10,000 = 100,000,000	11,913,820	12%	88%

บทที่ 4
ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดที่มีประสิทธิภาพด้วยเทคนิคไปป์ไลน์นิง

4. ขั้นตอนวิธี PLCS และ PLCS-SR แบบใหม่

งานวิจัยนี้นำเสนอการออกแบบขั้นตอนวิธี LCS แบบขนาน (Parallel LCS) ที่มีประสิทธิภาพทั้งในขั้นตอนการเตรียมข้อมูล และขั้นตอนการค้นหา LCS ซึ่งนำเสนอไว้ 2 ขั้นตอนวิธี คือ

1. ขั้นตอนวิธี PLCS (Parallel LCS) ด้วยเทคนิคไปป์ไลน์นิง (Pipelining Technique) ซึ่งจะนำเสนอในหัวข้อ 4.1 และ

2. ขั้นตอนวิธี PLCS-SR (Parallel LCS with Space Reduction) ที่ปรับปรุงมาจากขั้นตอนวิธี IDPSR-LCS [8] ที่ลดพื้นที่ในการจัดเก็บข้อมูลและเพิ่มเทคนิคไปป์ไลน์นิง ซึ่งจะนำเสนอในหัวข้อ 4.2

โดยทั้ง 2 วิธี เป็นการออกแบบขั้นตอนวิธีแบบขนานบนโมเดลของคอมพิวเตอร์แบบขนาน CREW-PRAM (Concurrent Read,Exclusive Write - Parallel Random Access Machine) และในการศึกษาเบื้องต้นจะสมมุติให้ p=n (p คือจำนวนหน่วยประมวลผล และ n คือความยาวของลำดับอักษรทั้งสองชุด) เพื่อความสะดวกในการอธิบายขั้นตอนวิธีแบบขนาน

4.1 ขั้นตอนวิธี PLCS

ขั้นตอนวิธี 4.1 เป็นการออกแบบขั้นตอนวิธี PLCS เพื่อการประมวลผลบนโมเดล CREW-PRAM ด้วยเทคนิคไปป์ไลน์นิงระหว่างหน่วยประมวลผลในแต่ละแถวในขั้นตอนการจัดเตรียมข้อมูล (Pre-processing) โดยมีหลักการทำงานคือ P_i จะเริ่มประมวลผลค่า c_{ij} ในแถวที่ i ใน clock j (โดยเริ่มที่แถวที่ i (i=1) หลักที่ j (j=1)) ต่อไปใน clock j+1 P_{i+1} จะเริ่มคำนวณค่า $c_{i+1,J}$ โดยที่ J=j-i+1 และใน clock ต่อไป (clock j+2) ก็จะเริ่มการทำงานของ P_{i+2} ไปเรื่อยๆ จนกระทั่งทุกหน่วยประมวลผล ทำการคำนวณค่า c_{IJ} เสร็จสิ้นภายใน 2n clocks

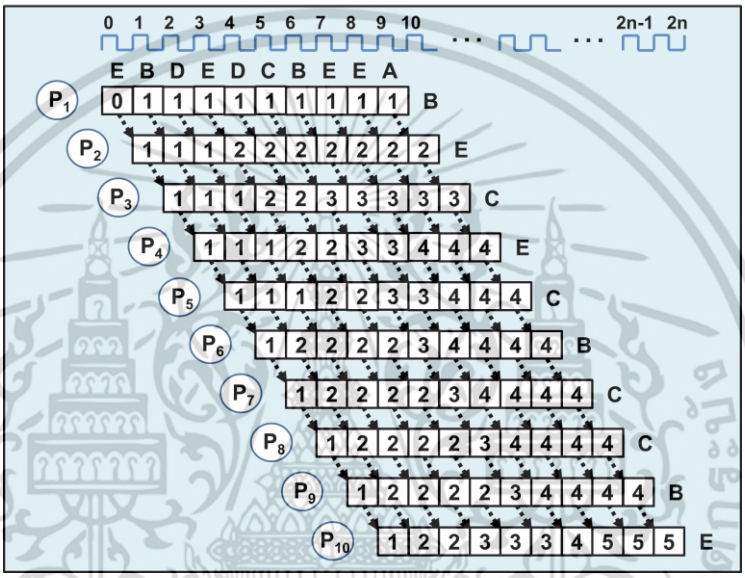
<u>หมายเหตุ</u> ในกรณีที่ p<n, n≠m จะใช้เทคนิคมอบภาระงาน (Workload) แบบ Cyclic เช่น ถ้า p=2 ดังนั้น P_1 จะประมวลผลแถว 1, 3, 5, ..., m-1 และ P_2 จะประมวลผลแถว 2, 4, 6, ... , m สลับกันไปจนครบทุกแถว

```
forall processors i=1 to n pardo
  for j = 1 to 2n do
    J = j–i+1;
    Pi: if (j≥i) and (J≤n) do
          if (xi=yJ) ci,J = ci-1,J-1+1;
          else ci,J = max(ci-1,J , ci,J-1);
        end if
  end for j
end forall
```

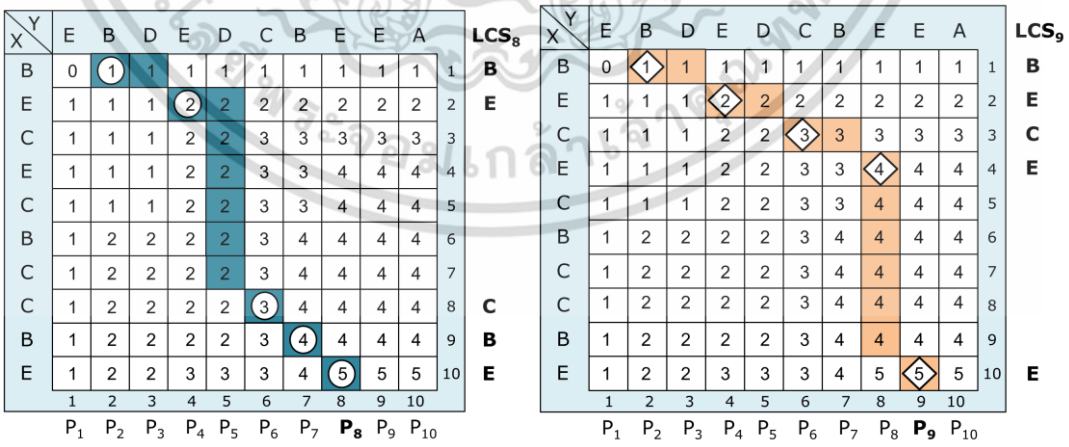
รูปที่ 4.1 ขั้นตอนวิธีการสร้าง Matrix C(nxn) ของ PLCS

```
forall processors j=1 to n pardo
   Ij=n; Jj=j; Kj=k;
   Pj: if (cIj,Jj<k) LCSj=Null;
       else do
         while (Kj>0) do
           if (xIj=yJj) zKj=xIj; Kj--; Ij--; Jj--;
           else if(cIj-1,Jj < cIj,Jj-1) Jj--; else Ij--;
         end while
end forall
```

รูปที่ 4.2 ขั้นตอนวิธีการค้นหา LCS แบบขนานของ PLCS



รูปที่ 4.3 การประมวลผลและเก็บค่าในเมตริกซ์ C ของขั้นตอนวิธี PLCS



รูปที่ 4.4 การค้นหา LCS ของขั้นตอนวิธี PLCS

รูปที่ 4.3 แสดงตัวอย่างการจัดเตรียมข้อมูล (Pre-processing) แบบขนานบนโมเดล CREW-PRAM ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA โดยใน clock 1 เริ่มการทำงานของ

P_1 เพื่อคำนวณค่า $c_{1,1}$ และใน clock 2 P_1 คำนวณค่า $c_{1,2}$ และเริ่มการทำงานของ P_2 เพื่อคำนวณค่า $c_{2,1}$ และจะดำเนินการซ้ำในทำนองเดียวกันนี้ไปเรื่อย ๆ ใน clock ต่อไป เช่น เมื่อประมวลผลถึง clock n P_1 จะประมวลผลเสร็จ n สมาชิก ส่วน P_2 จะประมวลผลได้ n-1 สมาชิก P_3 จะประมวลผลได้ n-2 สมาชิก และ P_n จะเริ่มประมวลผลได้เพียง 1 สมาชิก ดังนั้นขั้นตอนนี้จะยังคงทำซ้ำจนกระทั่งสุดท้าย P_n คำนวณค่า $c_{n,n}$ เสร็จสิ้นภายใน 2n clocks ซึ่งด้วยเทคนิคไพน์ไลน์นิงดังกล่าว (ทำให้สามารถคำนวณค่า c_{ij} ในแถวถัดไปใน clock ที่ต่อเนื่องได้อย่างเหมาะสม) ภายในเวลา 2n clocks หรือ O(n) ซึ่งเร็วกว่า $O(n^2)$ ที่ประมวลผลด้วย 1 หน่วยประมวลผล

ขั้นตอนวิธีจากรูปที่ 4.2 เป็นการค้นหา LCS แบบขนาน ด้วยเวลา O(n) โดยที่ทุกหน่วยประมวลผล P_j (j = 1, 2, 3, ... , n) จะสามารถเริ่มพร้อมกันด้วยการตรวจสอบเงื่อนไขเพื่อหา LCS ดังแสดงผลลัพธ์ในรูปที่ 4.4 ซึ่งมีเพียง 3 หน่วยประมวลผล (P_8, P_9, P_{10}) ที่จะค้นหา LCS ขนาด k=5 (ซึ่งในกรณีนี้ P_{10} จะได้ LCS_{10} เหมือน LCS_9) ส่วนหน่วยประมวลผลอื่นๆ (P_j ; j=1-7) ที่ $c_{10,j} < k$ (หรือไม่พบ LCS) ดังนั้นจะให้ค่าเป็น Null

4.2 ขั้นตอนวิธี PLCS-SR

ขั้นตอนวิธี PLCS-SR ประกอบไปด้วยการประมวลผล 2 ขั้นตอนคือ 1. การจัดเตรียมข้อมูล Mlist L(n) ด้วยเทคนิคไพน์ไลน์นิง (ขั้นตอนวิธีในรูปที่ 4.5) และ 2. การค้นหา LCS จาก Mlist (ขั้นตอนวิธีในรูปที่ 4.6) เมื่อ Mlist L(n) จัดเก็บข้อมูลค่าความยาวร่วมเฉพาะค่าใหม่ที่ไม่ซ้ำเพื่อลดเวลาในการคำนวณค่าความยาวร่วมด้วยขั้นตอนวิธีแบบขนานด้วยเทคนิคไพน์ไลน์นิงและลดพื้นที่การจัดเก็บข้อมูลในอาร์เรย์ของลิสต์

```
forall processors i=1 to n pardo
  for j=1 to 2n do
    J=j–i+1;
    Pi: if (j≥i) and (J≤n) do
          if (xi=yJ) cJ=LJ-1.c+1;
          else cJ=max(LJ.c, cJ-1);
          UpdateMlist(L(n),c(n),i,J)
        end if
  end for j
end forall
ImprovedMlist (L(n),n,Ln.c);
```

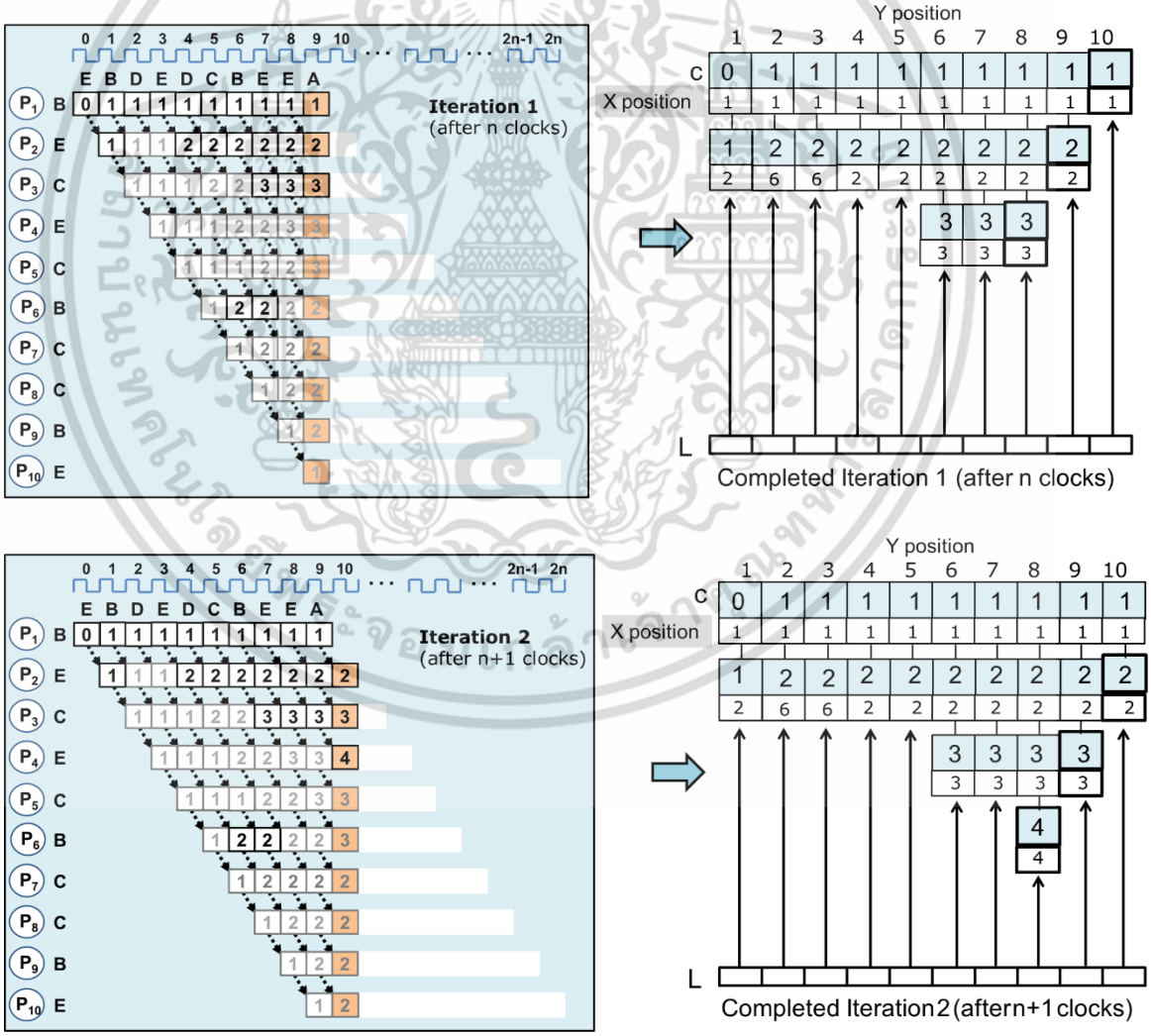
```
UpdateMlist (L(n),c(n),i,J) : O(1) by all processors
Pi: if(cJ ≠ LJ.c) do              // add new node in LJ
      allocate new node (N); // memory allocate
      N.c=cJ; N.x=i; N.back=LJ; LJ=N;
    end if
```

```
ImprovedMlist (L(n), f, k) : O(n) by P1
j=f; while(Lj.c=k) j=j-1;  // min j with LCS=k
f=j; for(j=f; j>0; j--)
        Cr=Lj; Pr=Lj+1;     // Cr=Current, Pr=Previous
        while(Cr.x > Pr.x) // adjust tail & free node
           Lj=c.back; free(Cr); Cr=Lj;
        end while
      end for j
return f+1
```

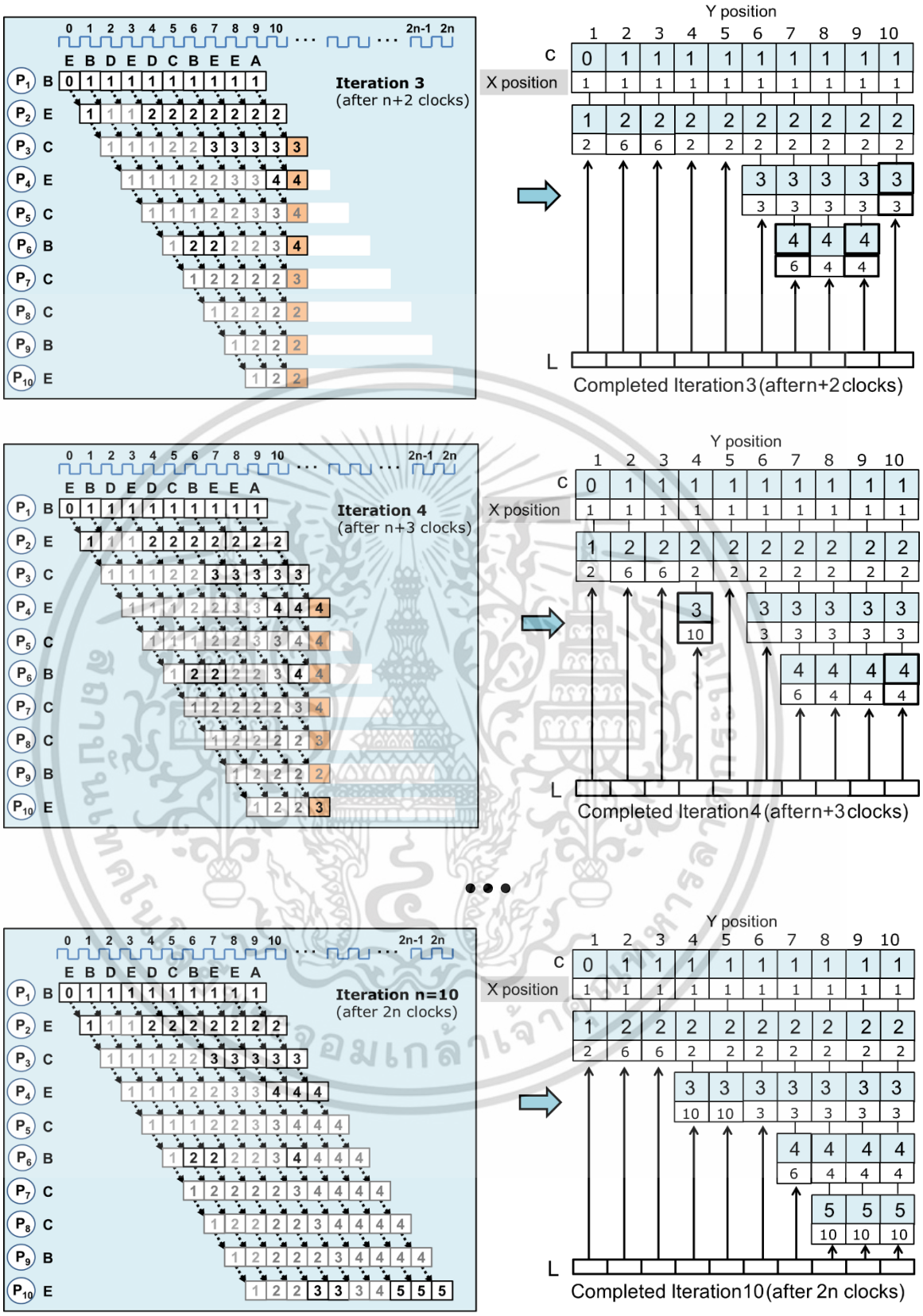
รูปที่ 4.5 ขั้นตอนวิธีการสร้าง Mlist L(n) [c,x,back] ของ PLCS-SR

```
forall processors j=1 to n pardo
  Ij=n; Jj=j; Kj=k;
  Pj: if (cIj,Jj<k) LCSj=Null;
      else do
        while (Kj>0) do
          vj=Lj; Ij=vj.x;
          if(xIj=yJj)and(vj.c=k) zKj=xIj; Kj--; Jj--;
          else do          // xIj≠yJj
            while(vj.c > k) vj=vj.back;
            Ij=vj.x;
            if (xIj=yJj)and(vj.c=k) zkj=xIj; Kj--; Jj--;
            else Jj--;   // XIj ≠YJj goto next LJj for k
          end if-else
        end while
      end if-else
end forall
```

รูปที่ 4.6 ขั้นตอนวิธีการค้นหา LCS แบบขนานของ PLCS-SR



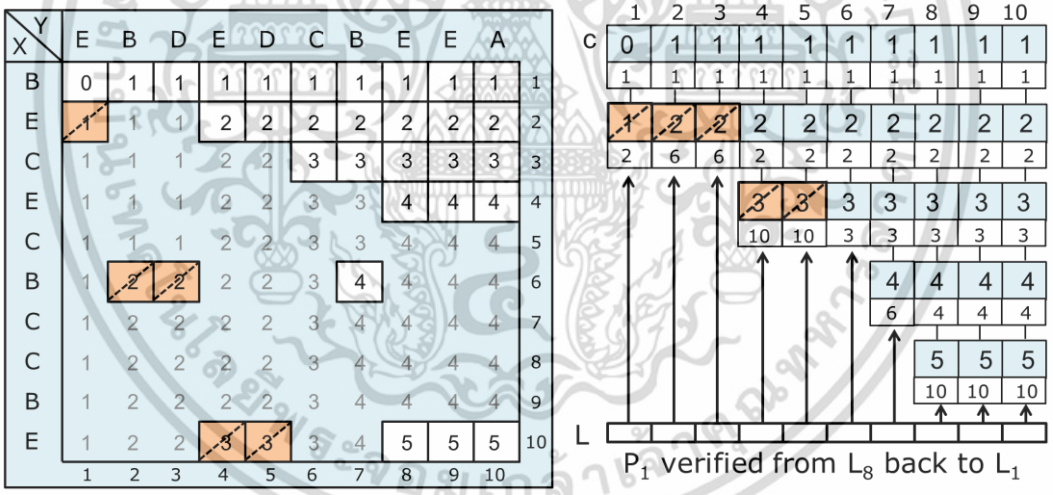
รูปที่ 4.7 การสร้าง Mlist L(n) จากขั้นตอนวิธี PLCS-SR ใน Iteration 1-2



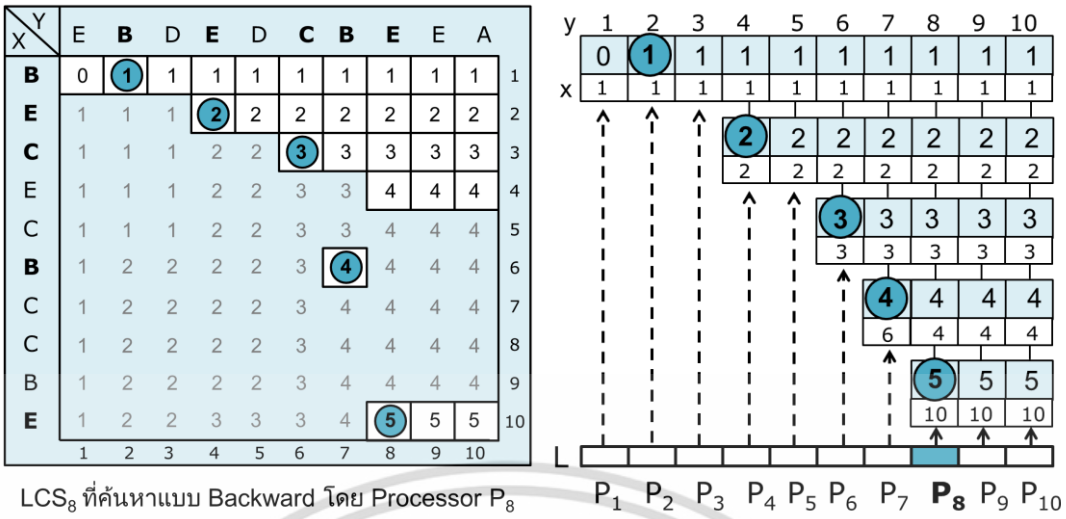
รูปที่ 4.8 การสร้าง Mlist L(n) จากขั้นตอนวิธี PLCS-SR

รูปที่ 4.7-4.8 แสดงการสร้าง Mlist L(n) ที่แต่ละโหนดเก็บค่า 3 ค่า [c, x, back] ในขั้นตอนการเตรียมข้อมูลของวิธี PLCS-SR จากข้อมูล X = BECECBCCBE และ Y=EBDEDCBEEA เมื่อทุกหน่วยประมวลผล P_i รับผิดชอบแถว i (=1, 2, ..., n) ของ X ซึ่งจะเริ่มเมื่อ j ≥ i (ด้วยเทคนิคไพน์ไลน์นิงตามค่า J=j-i+1) เช่นเดียวกับวิธี PLCS (ในรูปที่ 4.3) แต่ในกรณีนี้ P_i จะเก็บค่า c_J ลงในลิสต์ L_J เฉพาะค่าที่ไม่ซ้ำ และเมื่อดำเนินการถึง n clocks แรก จะได้ผลลัพธ์ดังนี้ ในรูปที่ 4.7 (รูปแรก หรือ Iteration1) P_1 ประมวลผล Iteration 1 เสร็จ (n สมาชิก คือ c_J; J=1, 2,...,n) พร้อมด้วยการเก็บค่าที่ไม่ซ้ำลงในลิสต์ ส่วน P_2 คำนวณได้ n-1 สมาชิกและเก็บเพียงบางค่า (i=2 และ J=1,4-9) ลงในลิสต์ในทำนองเดียวกับ P_3 – P_9 ซึ่งจะคำนวณได้ n-i+1 สมาชิก และสุดท้าย P_{10} คำนวณได้ 1 สมาชิก กล่าวโดยสรุปใน clock ที่ n=10 P_1–P_3 เก็บข้อมูลใหม่ลงในลิสต์ ส่วน P_4–P_{10} จะไม่เก็บข้อมูลเพราะเป็นค่าซ้ำที่มีอยู่ก่อนหน้า ต่อมาในรูปที่ 4.7 รูปที่สอง หรือ Iteration 2 แสดงผลลัพธ์หลังจาก n+1 clocks (ที่ประมวลผลโดย P_2 – P_{10}) โดย P_2 ประมวลผล Iteration 2 เสร็จ และ P_2 – P_4 เก็บข้อมูลใหม่ลงในลิสต์ ส่วน P_5–P_{10} จะไม่เก็บข้อมูลที่เป็นค่าซ้ำ และในทำนองเดียวกันจะดำเนินการประมวลผลซ้ำๆ ไปจนเสร็จ Iterations 3–10 โดยจะแสดงผลลัพธ์ Mlist ในภาพสุดท้าย

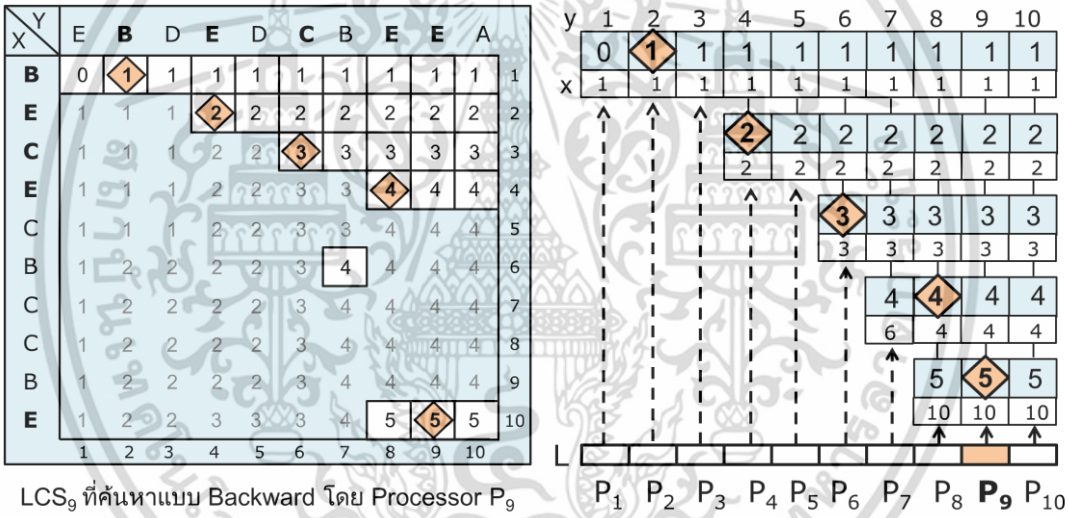
รูปที่ 4.9 แสดงผลลัพธ์ของฟังก์ชัน ImprovedMlist คือการปรับ Mlist L(n) เพื่อตัดบางโหนดที่ไม่ใช่ส่วนของ LCS ออก เช่น โหนดในตำแหน่ง (i, j) = (10,5), (10,4), (6,3), (6,2), (2,1) ทำให้สามารถลดพื้นที่ได้ถึง 72% และการเก็บค่าใน Mlist ไม่ต้องใช้พื้นที่ติดกันในหน่วยความจำ



รูปที่ 4.9 ตัวอย่างผลลัพธ์การปรับ Mlist ด้วยฟังก์ชัน ImprovedMlist



รูปที่ 4.10 การค้นหา LCS หลายค่า แบบ PLCS-SR จาก Mlist โดย P_8



รูปที่ 4.11 การค้นหา LCS หลายค่า แบบ PLCS-SR จาก Mlist โดย P_9

ในขั้นตอนการค้นหา LCS (ขั้นตอนวิธีจากรูปที่ 4.6) จาก Mlist ของวิธี PLCS-SR โดยมี Input คือ Mlist L(n), X, Y, ความยาว LCS = k โดยทุกหน่วยประมวลผล (P_j ; j=1, 2, ..., 10) สามารถประมวลผลได้พร้อมๆ กันด้วยเวลา O(n) โดยที่แต่ละหน่วยประมวลผลจะรับผิดชอบหา LCS 1 ชุด โดยในแต่ละ P_j จะกำหนดให้ v_j แทนโหนดใดๆ ที่เริ่มจาก tail ของ list L_j และ P_j จะหา LCS ได้ก็ต่อเมื่อค่า $v_j.c$ ของโหนดเริ่มต้นมีค่า=k ส่วนโหนดเริ่มต้นที่มีค่า $v_j.c < k$ จะกำหนดให้ LCS=Null (ไม่เป็นค่า LCS)

รูปที่ 4.10 และ 4.11 แสดงการค้นหา LCS แบบขนาน โดยที่ทุกหน่วยประมวลผล P_j (j = 1, 2, 3, ... , 10) จะต้องตรวจสอบเงื่อนไขก่อนการหา LCS พร้อมๆ กัน ซึ่งในตัวอย่างนี้จะมีเพียง 3 หน่วยประมวลผล (P_8, P_9, P_{10}) ที่จะทำการค้นหา LCS ขนาด k=5 ส่วนหน่วยประมวลผลอื่นๆ (P_j ; j = 1, 2, ... ,7) ที่มีค่าเริ่มต้นในโหนด v_j ของลิสต์ L_j คือ $v_j.c < k$ (แสดงว่าไม่เป็น LCS ซึ่งจะกำหนดให้ค่า LCS เป็น Null) สำหรับหน่วยประมวลผล $P_8 - P_{10}$ ที่มีค่าเริ่มต้นของโหนด $v_j.c$=k

ในลิสต์ L_8 – L_{10} จะทำการค้นหา LCS พร้อมๆ กัน รูปที่ 4.10 แสดงการค้นหา LCS_8 โดย P_8 ที่เริ่มจากโหนด v_j ของ L_8 ที่มีค่า $v_j.c=k$ ในตำแหน่ง (i, j)=(10, 8) ซึ่งจะได้ LCS_8 = BECBE และรูปที่ 4.11 แสดงการค้นหา LCS_9 โดย P_9 ที่เริ่มจากโหนด v_j ของ L_9 ที่มีค่า $v_j.c=k$ ในตำแหน่ง (i, j)=(10, 9) ซึ่งจะได้ LCS = BECEE ส่วน P_{10} จะได้ LCS_{10} เหมือนกับ LCS_9

4.3 ประสิทธิภาพของขั้นตอนวิธี PLCS และ PLCS-SR

การเปรียบเทียบประสิทธิภาพของขั้นตอนวิธี PLCS และ PLCS-SR (ที่นำเสนอในหัวข้อ 4.1 และ 4.2) ในที่นี้จะเป็นการแสดงประสิทธิภาพด้านเวลาเป็นหลัก คือความสามารถในการประมวลผลที่เร็วขึ้นด้วย n หน่วยประมวลผล ซึ่งจะเปรียบเทียบกับขั้นตอนวิธีแบบเดิม ที่ใช้หนึ่งหน่วยประมวลผล ดังแสดงเป็นค่าความซับซ้อน Time และ Space ในตาราง 4.1 (PLCS ที่มีข้อจำกัดเรื่องพื้นที่จัดเก็บข้อมูลแบบเมตริกซ์ nxn) และตาราง 4.2 (PLCS-SR ที่ลดพื้นที่จัดเก็บข้อมูลในส่วน Pre-processing) โดยความซับซ้อนด้านเวลาของการประมวลผลแบบขนานของทั้ง 2 วิธี เท่ากับ O(n) เทียบกับเดิม $O(n^2)$ ที่ประมวลผลด้วย 1 หน่วยประมวล ดังนั้น Speedup ($S_P=T_1/T_P$) ของทั้ง 2 วิธีมีค่า ≈n แต่วิธี PLCS เก็บข้อมูลในอาร์เรย์ 2 มิติ ที่เป็นแบบ shared matrix nxn ที่ไม่สามารถประมวลผลเมื่อ n มีค่ามาก ส่วนวิธี PLCS-SR เก็บข้อมูลใน shared Mlist ที่สามารถรองรับ n ทุกค่า

ตาราง 4.1 ความซับซ้อน Time และ Space ของขั้นตอนวิธี PLCS

ขั้นตอนวิธี	Sequential 1 PE		Parallel n PEs	
	Time (T_1)	Space	Time (T_P)	Shared Space
Pre-processing	$O(n^2)$	$O(n^2)$	O(n)	$O(n^2)$
MLCS searching	O(cn)	O(n)	O(n)	O(n)
Total	**$O(n^2)$**	**$O(n^2)$**	**O(n)**	**$O(n^2)$**

ตาราง 4.2 ความซับซ้อน Time และ Space ของขั้นตอนวิธี PLCS-SR

ขั้นตอนวิธี	Sequential 1 PE		Parallel n PEs	
	Time (T_1)	Space	Time (T_P)	Shared Space
Pre-processing	$O(n^2)$	O(kn)	O(n)	O(kn)
MLCS searching	O(cn)	O(n)	O(n)	O(n)
Total	**$O(n^2)$**	**O(kn)**	**O(n)**	**O(kn)**

บทที่ 5
สรุปผลการวิจัย

5.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอขั้นตอนวิธี IDPSR-LCS ซึ่งพัฒนามาจากขั้นตอนวิธีไดนามิคโปรแกรมมิง (DP) และขั้นตอนวิธี PLCS-SR ในการค้นหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS) ของข้อมูลสายอักษร 2 ชุด (X_m, Y_n) เพื่อปรับปรุงการใช้พื้นที่หน่วยความจำรวมถึงลดเวลาที่ใช้ในการประมวลผลขั้นตอนวิธีลง

จากผลการทดลองขั้นตอนวิธี IDPSR-LCS ซึ่งเป็นขั้นตอนวิธีที่นำเสนอใหม่นั้น ขั้นตอนวิธีสามารถลดพื้นที่ที่ใช้ในการเก็บข้อมูลเพื่อค้นหา LCS = O(n) ด้วยการใช้อาเรย์ของลิสต์ที่เก็บเฉพาะข้อมูลที่ไม่ซ้ำและเก็บสมาชิกแบบไดนามิค แทนอาเรย์ 2 มิติแบบเดิมที่ใช้พื้นที่ในการเก็บข้อมูล = O(mn) ซึ่งในทางปฏิบัติจะประมวลผลได้ไม่เกิน m, n = 800 เพราะอาเรย์ต้องใช้หน่วยความจำเก็บค่าที่ต่อเนื่องกัน (< 800 x 800) ขั้นตอนวิธี IDPSR สามารถประมวลผลกับข้อมูลสายอักษรที่มีความยาวได้มากกว่า m, n = 800 ซึ่งเป็นผลมาจากการปรับปรุงวิธีการเก็บข้อมูลในการประมวลผลในรูปแบบของลิสต์ โดยเมื่อเปรียบเทียบกับขั้นตอนวิธีแบบไดนามิคโปรแกรมมิงจะพบว่ามีการเก็บข้อมูลที่น้อยลงประมาณ 88% และนอกจากนั้นการเก็บข้อมูลที่ลดลงยังส่งผลให้สามารถค้นหา LCS รวดเร็วขึ้นเป็น O(n) จากเดิม O(m+n) จากการทำ pointer jumping ในขณะที่ขั้นตอนวิธี New LCS 2007 ที่เก็บข้อมูลในรูปแบบของลิสต์และ SA-MLCS 2014 ที่เก็บข้อมูลในรูปแบบกราฟจะมีขั้นตอนวิธีในการค้นหา LCS ที่ซับซ้อนกว่าส่งผลให้ต้องมีการใช้พื้นที่และเวลาในการค้นหา LCS เพิ่มมากขึ้น

ในขณะที่ขั้นตอนวิธี PLCS-SR เป็นขั้นตอนวิธีการหาลำดับร่วมเหมือนที่ยาวที่สุด LCS แบบขนานบนโมเดล Shared memory CREW-PRAM ที่มีประสิทธิภาพด้านความเร็วเท่ากับ O(n) ทั้งในส่วนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) และการค้นหา LCS ที่ประกอบด้วย 2 ขั้นตอนวิธี โดยวิธีแรก PLCS พัฒนามาจากขั้นตอนวิธี DP-LCS ดั้งเดิม ด้วยเทคนิคไปป์ไลน์นิงแบบขนาน และวิธีที่ 2 PLCS-SR ที่พัฒนามาจากขั้นตอนวิธี IDPSR-LCS ที่ลดพื้นที่จัดเก็บข้อมูล [8] ร่วมกับการประมวลผลแบบขนานด้วยเทคนิคไพน์ไลน์นิง ซึ่งในการเปรียบเทียบประสิทธิภาพของทั้งสองวิธีพบว่า ค่า Speedup ของทั้งสองวิธีมีค่า ≈n แต่วิธี PLCS มีข้อจำกัดที่ขนาดของ Shared matrix เช่นเดียวกับวิธี DP-LCS ส่วนวิธี PLCS-SR เก็บข้อมูลใน Shared Mlist ที่รองรับกรณีที่ n มีค่ามากได้ สอดคล้องกับยุคปัจจุบันที่ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) และต้องการความรวดเร็วในการประมวลผล

5.2 ข้อเสนอแนะ

จากการปรับปรุงขั้นตอนวิธีการหาลำดับร่วมเหมือนยาวที่สุดที่ผู้วิจัยได้นำเสนอได้มุ่งประเด็นการปรับปรุงขั้นตอนวิธีในการหาคำตอบจากสายอักษรเพียง 2 ชุดเท่านั้น ในขณะที่ปัญหาการหาลำดับร่วมเหมือนยาวที่สุดสามารถนำสายอักษรมาเปรียบเทียบครั้งละมากกว่า 2 ชุดขึ้นไป ที่เรียกว่า multiple longest common subsequence หรือ MLCS รวมถึงในงานวิจัยอื่นๆ ได้มีการนำขั้นตอนวิธี LCS ไปประยุกต์ใช้ในงานต่างๆ เช่น การเปรียบเทียบการเรียงตัวของ DNA หรือโปรตีนของผู้ป่วยกับคนปกติ, การตรวจสอบการคัดลอกโปรแกรมของนักศึกษา, การตรวจสอบรูปแบบการป้อนอักขระด้วยท่าทาง และอื่นๆ ดังนั้นในการวิจัยต่อไปที่น่าสนใจจะเป็นการออกแบบขั้นตอนวิธี MLCS แบบขนาน สำหรับข้อมูลหลายชุด (X_1, X_2, ... , X_s) เนื่องจากเป็นปัญหา NP-hard ชนิดหนึ่ง และการนำขั้นตอนวิธี PLCS-SR ไปประยุกต์ใช้ในงานต่างๆ

เอกสารอ้างอิง

[1] K. Ning, H. Kee and H. W. Leong. 2006. “Finding Patterns in Biological Sequences by Longest Common Subsequences and Shortest Common Supersequence” 53-60. **Sixth IEEE Symposium on BioInformatics and BioEngineering.**

[2] R.A.C. Campos and F.J.Z Martinez. 2012. “Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem” 1–4. **9th International Conference on Electrical Engineering, Computing Science and Automatic Control.**

[3] D. Abraham and N.S. Raj. 2014. “Approximate string matching algorithm for phishing detection” 2285–2290. **2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI).**

[4] D. Frolova, H. Stern and Berman, S. 2013. "Most Probable Longest Common Subsequence for Recognition of Gesture Character Input" 871–880. **IEEE Trans. on Cybernetics.** vol. 43. no.3.

[5] J. Liaw and et al. 2013. “Recognition of the Ambulance Siren Sound in Taiwan by the Longest Common Subsequence” 3824-3828. **2013 IEEE international conference on Systems, Man, and Cybernetics.**

[6] X. Xiang, D. Zhang, and J. Qin. 2007. “An New Algorithm for the Longest Common Subsequence Problem” 112–115. **2007 International Conference on Computational Intelligence and Security Workshops.**

[7] J. Yang, Y. Xu, Y. Shang, and G. Chen. 2014. “A Space-Bounded Anytime Algorithm for the Multiple Longest Common Subsequence Problem” 2599–2609. **IEEE Trans. on Knowledge and Data Engineering.** vol. 26. no. 11.

[8] S. Dangmanee and J. Werapun. 2016. “Longest Common Subsequence by New Dynamic Programming with Space Reduction” 286–291. **The 12th National Conference on Computing and Information Technology.**

[9] Q. Wang, D. Korkin, and Y. Shang. 2011. “A Fast Multiple Longest Common Subsequence (MLCS) Algorithm” 321–334. **IEEE Transactions on Knowledge and Data Engineering.** vol.23, no.3.

[10] J. Yang and et al. 2013. “A New Progressive Algorithm for a Multiple Longest Common Subsequences Problem and Its Efficient Parallelization” 862–870 **IEEE Transactions on Parallel and Distributed Systems.** vol.24. no.5.

ภาคผนวก

งานวิจัยที่ตีพิมพ์

1. S. Dangmanee and J. Werapun. 2016. “Longest Common Subsequence by New Dynamic Programming with Space Reduction” 286–291. **The 12th National Conference on Computing and Information Technology.**
2. S. Dangmanee and J. Werapun. 2016. “Efficient Parallel Longest Common Subsequence with Pipelining Technique” 5 3 1 –5 3 6 . **The 20th International Computer Science and Engineering Conference 2016.**

NCCIT 2016

The 12th National Conference on Computing and Information Technology

Proceedings of NCCIT 2016
The 12th National Conference on Computing and Information Technology
7th - 8th July 2016
At Centara Hotel & Convention Centre Khon Kaen, Thailand
www.nccit.net

Faculty of Informatics
Mahasarakham University

Faculty of Information Technology
King Mongkut's University of Technology North Bangkok

บทความวิจัย

การประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 12
7-8 กรกฎาคม 2559
โรงแรมเซ็นทาราและคอนเวนชันเซ็นเตอร์ ขอนแก่น

คณะวิทยาการสารสนเทศ
มหาวิทยาลัยมหาสารคาม

คณะเทคโนโลยีสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

การหาลำดับร่วมเหมือนที่ยาวที่สุดด้วยขั้นตอนวิธีไดนามิคโปรแกรมมิงแบบลดพื้นที่
Longest Common Subsequence by New Dynamic Programming with Space Reduction

ศักดิ์พันธุ์ แดงมณี (Sakpan Dangmanee)[1] และจีรพร วีระพันธุ์ (Jeeraporn Werapun)[2]
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
sakpan.da@up.ac.th[1], jeeraporn.we@kmitl.ac.th[2]

บทคัดย่อ

บทความวิจัยนี้ ได้นำเสนอขั้นตอนวิธีไดนามิคโปรแกรมมิง (Dynamic Programming (DP)) แบบใหม่ (IDPSR) ที่ลดพื้นที่ ในการประมวลผลลำดับร่วมเหมือนที่ยาวที่สุด (Longest Common Subsequence (LCS)) ของข้อมูลสายอักษร 2 ชุด ปัญหาการค้นหา LCS ดังกล่าว ถูกนำมาประยุกต์ใช้ในงานหลากหลายประเภท อาทิ DNA matching และการตรวจจับ Web Phishing โดยขั้นตอนวิธี DP เดิม ใช้พื้นที่ในหน่วยความจำแบบเมตริกซ์ขนาด mxn ซึ่งเป็นอาเรย์ 2 มิติ ที่ต้องใช้พื้นที่ติดกันในหน่วยความจำ ซึ่งเป็นข้อจำกัดในการประมวลผล m, n ขนาดใหญ่ๆ งานวิจัยที่ผ่านมา จึงเน้นขั้นตอนวิธีที่ใช้พื้นที่จัดเก็บเฉพาะข้อมูลที่สำคัญๆ แต่อาจประมวลผลนานขึ้น ดังนั้นเพื่อแก้ปัญหาดังกล่าว ประกอบกับข้อมูลที่มีอยู่ในยุคปัจจุบันมีการขยายตัวขึ้นอย่างรวดเร็ว ในงานวิจัยนี้ ผู้วิจัยจึงได้เสนอขั้นตอนวิธี IDPSR-LCS ที่ลดขนาดพื้นที่จัดเก็บข้อมูลด้วยอาเรย์ 1 มิติของลิสต์ Mlist ในส่วนขั้นตอนการจัดเตรียมข้อมูล ซึ่งยังช่วยลดเวลาการค้นหา LCS ด้วย

คำสำคัญ: ลำดับร่วมเหมือนที่ยาวที่สุด (LCS), ขั้นตอนวิธี IDPSR แบบใหม่ที่ลดพื้นที่เก็บข้อมูล, LCS สำหรับ Web Phishing

Abstract

This paper presents a new dynamic programming algorithm with space reduction (IDPSR) for solving LCS (Longest Common Subsequence) for some applications such as DNA matching, Web Phishing, etc. A limitation of the existing dynamic programming for LCS is the contiguous-space requirement for the mxn-matrix or 2D-array in pre-processing. Recent LCS algorithms focus on improved memory space by storing only important data in pre-processing but more storage and time are required in searching step for the LCS. Therefore, we introduce the new LCS algorithm for large m and n with the array of multiple lists (Mlist) in the pre-processing, which can also improve searching time for the LCS.

Keyword: longest common subsequence (LCS), new IDPSR with space reduction, LCS for Web Phishing.

1. บทนำ

ขั้นตอนวิธีลำดับร่วมเหมือนที่ยาวที่สุด (Longest Common Subsequence (LCS)) เป็นขั้นตอนวิธีสำหรับหาตัวอักษรหรือข้อมูลที่มีลำดับที่ตรงกันและมีความยาวมากที่สุด ในปัจจุบันขั้นตอนวิธีดังกล่าวถูกนำมาประยุกต์ใช้ในงานหลากหลายชนิด เช่น การเปรียบเทียบการเรียงตัว DNA หรือโปรตีนของผู้ป่วยและคนปกติเพื่อตรวจหาความผิดปกติอย่างละเอียด [1], การตรวจสอบการคัดลอกผลงานตีพิมพ์ที่เป็นงานเขียน หรือการคัดลอกโปรแกรมของนักศึกษาในการเรียนการสอน [2], การตรวจสอบ URL ของเว็บไซต์ว่าเป็นเว็บไซต์ Phishing หรือไม่ [3] การตรวจสอบรูปแบบการป้อนอักขระด้วยท่าทาง (Gesture Character Input) [4] และการตรวจสอบรูปแบบความเหมือนของเสียง [5] เป็นต้น การนำขั้นตอนวิธี LCS ไปประยุกต์ใช้ในงานต่างๆ อย่างแพร่หลาย ทั้งนี้เนื่องมาจากข้อดีของขั้นตอนวิธี คือสามารถประมวลผลได้รวดเร็วและแม่นยำกว่าขั้นตอนวิธีแบบฮิวริสติกอื่นๆ และเนื่องจากมีกระบวนการที่ไม่ซับซ้อนจนเกินไป แต่เมื่อทำการอิมพลีเมนต์ขั้นตอนวิธี LCS ด้วยการโปรแกรมแบบพลวัต (Dynamic Programming (DP)) พบว่าขั้นตอนวิธี DP-LCS ใช้พื้นที่จัดเก็บข้อมูลในหน่วยความจำค่อนข้างมาก ซึ่งไม่สอดคล้องกับข้อมูลที่มีอยู่ในยุคปัจจุบันที่มีการขยายตัวขึ้นอย่างรวดเร็ว เพื่อรองรับการใช้งานแอพพลิเคชันที่เป็นเครือข่ายสังคม (Social Network) หรือที่เรียกว่ายุคข้อมูลขนาดใหญ่ (Big Data) ทำให้ขั้นตอนวิธี DP-LCS เดิม ที่มีประสิทธิภาพ อาจไม่สามารถนำมาใช้ในการประมวลผลข้อมูลที่มีขนาดใหญ่ๆ ได้ การปรับปรุงให้ขั้นตอนวิธีใช้พื้นที่ในการประมวลผลที่ลดลงจึงเป็นสิ่งจำเป็นในการนำขั้นตอนวิธี DP ดังกล่าว มาใช้ได้อย่างมีประสิทธิภาพในยุคปัจจุบัน

ในทางทฤษฎี ขั้นตอนวิธี DP-LCS เดิมที่มีประสิทธิภาพ จะถูกพัฒนาเป็น 2 ส่วนคือ 1. การจัดเตรียมข้อมูลก่อนประมวลผล (Pre-processing) โดยสร้างตารางเก็บข้อมูลขนาด mxn เพื่อระบุตำแหน่งที่มีตัวอักษรที่เหมือนกัน และ 2. นำข้อมูลในขั้นแรกไปใช้ประมวลผลเพื่อหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS) ซึ่งใช้เวลาประมวลผล O(mn) และพื้นที่เก็บข้อมูล O(mn) ในปี ค.ศ. 2007 X.Xiang และคณะ [6] ได้เสนอขั้นตอนวิธี LCS แบบใหม่ที่ลดพื้นที่ด้วยการเก็บข้อมูลจุดที่เป็น Pre-sequence เท่านั้น ในส่วนของการจัดเตรียมข้อมูล ด้วยเวลา O(mn) เท่าเดิม แต่ลดพื้นที่ลงเหลือ O(Ln) ต่อมาในปี ค.ศ.2014 J.Yang และคณะ [7] เสนอขั้นตอนวิธี SA-MLCS ที่ลดพื้นที่ ด้วยการจัดเก็บข้อมูลที่เป็น Dominant Point (จุดที่เป็นตัวอักษรร่วมของข้อมูลลำดับ) ก่อนการประมวผล LCS ด้วยพื้นที่ O(n) แต่ใช้เวลา O(mn) เพื่อสร้างกราฟที่ครอบคลุมจุดเหมือนในลำดับถัดไป แต่ในขั้นตอนการหา Multiple LCS จะมีการใช้พื้นที่เพิ่ม O(n) สำหรับข้อมูลคิว 3 คิว (New, Open, and Closed Queues) เพื่อหา MLCS

ในงานวิจัยนี้ ผู้วิจัยได้เสนอขั้นตอนวิธี DP แบบใหม่ ที่ลดพื้นที่จัดเก็บข้อมูล (IDPSR-LCS) โดยปรับปรุงโครงสร้างในการจัดเก็บข้อมูลในขั้นตอนจัดเตรียมข้อมูล ด้วยการใช้พื้นที่ในการประมวลผลจาก O(mn) เป็น O(n) ด้วยเวลา O(mn) และลดเวลาในส่วนของการค้นหา LCS จาก O(m+n) เป็น O(n)

เนื้อหาในหัวข้อต่อไป เป็นการแสดงผลงานวิจัยที่เกี่ยวข้องในการแก้ปัญหา LCS และ MLCS โดยในหัวข้อที่ 3 เป็นการนำเสนอขั้นตอนวิธีไดนามิคโปรแกรมมิงแบบใหม่ (IDPSR-LCS) ที่ลดพื้นที่จัดเก็บข้อมูล รวมทั้งวิธีการค้นหา LCS และ MLCS ที่มีประสิทธิภาพ ส่วนหัวข้อที่ 4 แสดงการเปรียบเทียบความซับซ้อนด้านเวลาและพื้นที่จัดเก็บข้อมูล พร้อมผลการทดลอง และสรุปผลงานวิจัยในหัวข้อที่ 5

2. ขั้นตอนวิธี LCS และงานวิจัยที่เกี่ยวข้อง

ปัญหาการค้นหาสายอักษรร่วมที่ยาวที่สุด เป็นการเปรียบเทียบสายอักษร ที่อยู่ในลักษณะของข้อมูลสายอักษร จำนวน 2 ชุด กำหนดให้เป็น X และ Y โดยที่ $X = <x_1, x_2, \dots, x_m>$ เป็นสายอักษรชุดหนึ่งที่มีความยาว m ตัวอักษร และ $Y = <y_1, y_2, \dots, y_n>$ เป็นสายอักษรอีกชุดอีกหนึ่งที่มีความยาว n ตัวอักษร

2.1 วิธีการค้นหา LCS แบบไดนามิคโปรแกรมมิง

ในการหาสายอักษรร่วมที่ยาวที่สุด (LCS) $Z = <z_1, z_2, \dots, z_k>$ (ขนาด k ตัวอักษร) ของสายอักษร X_m และ Y_n ขั้นตอนวิธี 2.1 จะดำเนินการค้นหา LCS แบบ Backward โดยพิจารณาจากค่าความยาวร่วมที่ประมวลผลไว้ก่อน (Pre-processing) ในเมตริกซ์ C ขนาด mxn โดยเริ่มค้นหาจาก (i, j) แรกที่มีค่า $c_{i,j}=k$

ในขั้นตอน Pre-processing จะจัดเตรียมเมตริกซ์ C_{mxn} ที่เก็บค่าของความยาวร่วมของสายอักษร (LCS) เมื่อ $c_{i,j}$ แทนค่าความยาวของ LCS จากสายอักษร X_m และ Y_n ที่คำนวณจากทุกค่าของ x_i และ y_j สำหรับ i = 1, 2, …, m และ j = 1, 2, …, n

เมื่อ $c_{i,j} = c_{i-1,j-1} + 1$ ถ้า $x_i = y_i$

$= \max(c_{i-1,j}, c_{i,j-1})$ ถ้า $x_i \neq y_i$

ขั้นตอนวิธี 2.1: การสร้างเมตริกซ์ C_{mxn} และการค้นหา LCS

```
for (i=1; i<=m; i++)
  for(j=1; j<=n; j++)
    if(xi=yj) ci,j = ci-1,j-1 +1;
    else ci,j = max(ci-1,j , ci,j-1);
end for j, i;
```

```
BackwardLCS (Cmn, Xm, Yn, Zk, i, j, k)
  while(k>0) do
    if(xi=yj) zk=xi; k=k-1; i=i-1; j=j-1;
    else if(ci-1,j<ci,j-1) j=j-1; else i=i-1;
  end while;
end LCS;
```

ตัวอย่างเช่น ภาพที่ 1 แสดงการคำนวณค่าเมตริกซ์ C_{10x10} ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA เพื่อหา LCS แบบ Backward จากค่า c_{ij} ที่ยาวที่สุด (ในที่นี้คือ k=5) และในตัวอย่างนี้จะได้ MLCS คือ BECBE และ BECEE

X \ Y	E	B	D	E	D	C	B	E	E	A		LCS_1	LCS_2
B	0	1	1	1	1	1	1	1	1	1	1	B	B
E	1	1	1	2	2	2	2	2	2	2	2	E	E
C	1	1	1	2	2	3	3	3	3	3	3		C
E	1	1	1	2	2	3	3	4	4	4	4		E
C	1	1	1	2	2	3	3	4	4	4	5		
B	1	2	2	2	2	3	4	4	4	4	6		
C	1	2	2	2	2	3	4	4	4	4	7		
C	1	2	2	2	2	3	4	4	4	4	8	C	
B	1	2	2	2	2	3	4	4	4	4	9	B	
E	1	2	2	3	3	3	4	5	5	5	10	E	E
	1	2	3	4	5	6	7	8	9	10			

ภาพที่ 1: ตัวอย่างการค้นหา LCS ด้วยขั้นตอนวิธี DP เดิม

2.2 วิธีการค้นหา MLCS ของ X. Xiang

ในส่วนของขั้นตอนวิธีของ X. Xiang และคณะ [6] การจัดเตรียมข้อมูลก่อนทำการค้นหา LCS จะทำการคำนวณข้อมูล (ค่าความยาวร่วม) ทั้งหมดก่อน เช่นเดียวกับขั้นตอนวิธี DP แต่วิธีนี้จะจัดเก็บเฉพาะบางคู่ลำดับที่เป็น Pre-sequence point ที่จะใช้ในการค้นหา LCS โดยเก็บค่าในอาเรย์ 1 มิติ (P) ซึ่งเป็นคู่ลำดับ (i, j) จากตำแหน่งของตัวอักษรที่เหมือนกัน ที่ปรากฏอยู่ในสายอักษรแต่ละชุด ดังแสดงเป็นตัวอย่างในภาพที่ 2

ขั้นตอนวิธี 2.2: New LCS (I, J) ของ X. Xiang [6]

```
1. construct P = {(i,j): ii = jj in the order of I}
2. construct P1, P2, ... , Pl,
   where l =number of matches between I and J
3. LCS(I,J) = max{Pi ; i = 1, 2, ..., l}
```

กำหนดให้ $P = \{(i, j) : i_i=j_j\}$ แล้ว (i, j) เป็นคู่ลำดับตำแหน่งของตัวอักษรที่เหมือนกัน (Match pair) จากสายอักษร I และ J ซึ่งอาจเป็น p_x ที่สอดคล้องกับตัวอักษรที่เกิดขึ้นทั้งหมด โดยแสดงในรูปแบบของกลุ่มคู่ลำดับ $P = \{p_1, p_2, \dots, p_l\}$ โดยที่แต่ละ p_k เป็นคู่อันดับของ (i_k, j_k) การสร้างเซ็ตของคู่ลำดับ จะกำหนดให้คู่ลำดับของอักษรที่เหมือนกันจำนวนสองคู่ p_x, p_y, ถ้า $i_x < i_y$, $j_x < j_y$ เมื่อกำหนดให้ $p_x < p_y$, p_y เป็นลำดับที่อยู่ตามหลัง (sub-sequence) ของลำดับ p_x หรือ p_x เป็นลำดับที่อยู่ก่อนหน้า (pre-sequence) ของ p_y ตัวอย่างเช่น ภาพที่ 2 แสดงการหา MLCS ของข้อมูล I = ATCTGAT และ J = TGCATA

I \ J	T	G	C	A	T	A	
A	0	0	0	1	1	1	1
T	1	1	1	1	2	2	2
C	1	1	2	2	2	2	3
T	1	1	2	2	3	3	4
G	1	2	2	2	3	3	5
A	1	2	2	3	3	4	6
T	1	2	2	3	4	4	7
	1	2	3	4	5	6	

$P = \{(i,j) : i_i = j_j\}$.

$P = \{ (1,4), (1,6),$
$(2,1), (2,5),$
$(3,3),$
$(4,1), (4,5),$
$(5,2),$
$(6,4), (6,6),$
$(7,1), (7,5) \}$

ภาพที่ 2: ตัวอย่างการเก็บข้อมูลเพื่อค้นหา LCS ของ X. Xiang

1. การเตรียมข้อมูล (Pre-processing) เก็บในอาเรย์ 1 มิติ

   P = {(1,4), (1,6), (2,1), (2,5), (3,3), (4,1), (4,5), (5,2), (6,4), (6,6), (7,1), (7,5)}

2. การหา LCS จาก P

   P_1 = {(1,4), (2,5), (6,6)}
   P_2 = {(1,6)}
   P_3 = {(2,1), (3,3), (4,5), (6,6)}
   P_4 = {(2,1), (3,3), (6,4), (7.5)}
   P_5 = {(4,1), (5,2), (6,4), (7,5)}
   P_6 = {(4,1), (5,2), (6,6)}
   P_7 = {(7,1)}

   ในที่นี้ จะได้ LCS = max{P_i} คือ P_3, P_4, P_5 (หรือ MLCS)

วิธีการค้นหา LCS ในขั้น 2 จะเริ่มโดยสร้างเซตคู่ลำดับ P{(i, j): $i_i=j_j$} ตามลำดับตัวอักษรของ I จากนั้นสร้างชุดลำดับตั้งแต่ P_1 จนไปถึง P_l โดยที่ l แสดงถึงจำนวนคู่ระหว่างสายอักษรทั้งสองลำดับและ $P_k=\{p_z: p_z<*p_{z+1}, z = k, k+1, k+2, \dots, l\}$ จากนั้น LCS คือเซตคู่ลำดับที่มีความยาวที่สุด LCS(I, J) = max{P_i}

2.3 วิธีการค้นหา MLCS ของ J. Yang

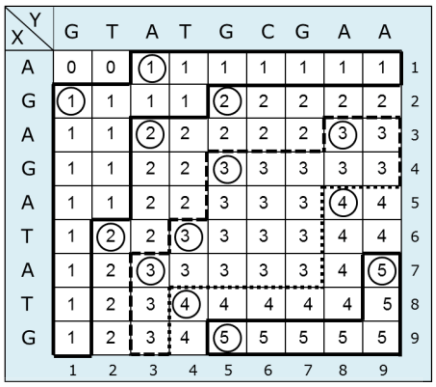
ขั้นตอนวิธี SA-MLCS ของ J.Yang และคณะ [7] จะจัดเตรียมข้อมูลก่อนค้นหา LCS โดยการตรวจสอบข้อมูลทั้งหมด (Pre-processing) เช่นเดียวกับขั้นตอนวิธี DP ตัวอย่างเช่น สำหรับข้อมูลสายอักษร I = AGAGATATG และ J = GTATGCGAA ภาพที่ 3(ก) แสดงการคำนวณค่าความยาวร่วมทั้งหมด และการค้นหาจุดที่เป็น Dominant point คือจุดที่มีอักษรเหมือนกันและเป็นจุดที่เกิดขึ้นก่อนในแนวแกน X และแกน Y จากนั้นจึงนำจุดดังกล่าวที่ได้มาสร้างเป็นกราฟ (ดังแสดงในภาพที่ 3(ข) แล้วจึงทำการค้นหา LCS หลายชุด (MLCS) ในเส้นทางของกราฟที่ถูกนำมาสร้างขึ้น โดยโหนดของกราฟที่ใช้ในการค้นหา จะถูกจัดกลุ่มเป็นระดับชั้น (layers) ด้วยระยะห่างจากโหนดต้นทาง สำหรับแต่ละระดับชั้น ในขั้นตอนวิธี SA-MLCS ดังกล่าว จะมีการใช้โครงสร้างข้อมูลแบบคิว 3 คิว (New, Open, และ Closed queues) ซึ่งมีนิยามการสร้างดังนี้

New queue คือคิวที่จัดเก็บโหนดที่สามารถสร้างโหนดลูกได้
Open queue คือคิวของโหนดที่ยังไม่ได้สร้างลูกทั้งหมด
Closed queue คือคิวของโหนดที่ทำการสร้างลูกแล้วทั้งหมด

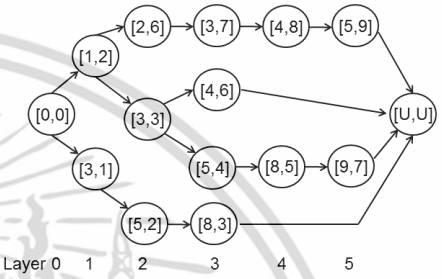
ขั้นตอนวิธี 2.3: SA-MLCS ของ J.Yang [7]

1. add source S into New_0 (in level 0).
2. set init_layer =0 and cur_layer =0.
3. for all nodes in New_{cur_layer} generate their all children and add to $New_{cur\_layer+1}$.
   if cur_layer≠0 then
   - mark Boolean_array of parents with 1.
   - if all children are expanded, then move parent to $Closed_{cur\_layer-1}$;
     else resort parent node in $Open_{cur\_layer-1}$.
4. for first i nodes in $Open_{cur\_layer}$ where i=min(c,|$Open_{cur\_layer}$|); c= widened beam generate unexpanded children in $New_{cur\_layer+1}$.
5. move all nodes in New_{cur_yaler} to $Open_{cur\_layer}$.
6. reserve first i nodes in $New_{cur\_layer+1}$, where i=min(c,|$New_{cur\_layer+1}$|) and delete others.
7. if cur_layer is not the last layer, then
   - if init_layer = cur_layer and |$Open_{cur\_layer}$| =|New_{cur_layer}|=0, then init_layer=init_layer+1.
   - cur_layer=cur_layer+1 and go to step 3 to continue this iteration.
8. if cur_layer is the last layer, then
   - output new solution by tracking back nodes in cur_layer if cur_layer > length of current$_{solution}$.
   - if init_layer=cur_layer and |$Open_{cur\_layer}$| =|New_{cur_layer}|=0, then terminate.
   - otherwise, cur_layer = init_layer and go to step 3 to start new iteration.

ภาพที่ 4 แสดงการค้นหา Multiple LCS (MLCS) จากโครงสร้างข้อมูลกราฟ (ภาพที่ 3) ด้วยการสร้างโครงสร้างข้อมูลแบบคิว 3 คิวคือ New, Open, และ Closed ซึ่งดำเนินการด้วยขั้นตอนวิธี 2.3 เป็นจำนวน 4 iterations และได้ MLCS 2 ชุด คือ GAGAA (จาก Iteration 1: (1,2),(3,3),,(5,4),(8,5),(9,7),) และ GTATG (จาก Iteration 3: (1,2),(2,6),(3,7),(4,8),(5,9))



(ก)



(ข)

ภาพที่ 3: ตัวอย่างการเก็บข้อมูลเพื่อค้นหา MLCS ของ J.Yang

Iteration 1

	Closed	Open	New
Layer 0		[0,0]	~~[0,0]~~
Layer 1		[1,2]	[1,2] ~~[3,1]~~
Layer 2		[3,3]	~~[3,3]~~ ~~[2,6]~~
Layer 3	[5,4]		~~[5,4]~~ ~~[4,6]~~
Layer 4	[8,5]		~~[8,5]~~
Layer 5	[9,7]		~~[9,7]~~

Iteration 2

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1]	[1,2]	~~[3,1]~~
Layer 2	[3,3]	[5,2]	~~[5,2]~~
Layer 3	[5,4] [4,6]		~~[4,6]~~ ~~[8,3]~~
Layer 4	[8,5]		
Layer 5	[9,7]		

Iteration 3

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1] [1,2]		
Layer 2	[3,3] [2,6]	[5,2]	~~[2,6]~~
Layer 3	[5,4] [4,6] [3,7]		~~[3,7]~~
Layer 4	[8,5] [4,8]		~~[4,8]~~
Layer 5	[9,7] [5,9]		~~[5,9]~~

Iteration 4

	Closed	Open	New
Layer 0	[0,0]		
Layer 1	[3,1] [1,2]		
Layer 2	[3,3] [2,6] [5,2]		
Layer 3	[5,4] [4,6] [3,7] [8,3]		~~[8,3]~~
Layer 4	[8,5] [4,8]		
Layer 5	[9,7] [5,9]		

ภาพที่ 4: ตัวอย่างการค้นหา MLCS ในขั้นตอนวิธี 2.3 ของ J.Yang

ตาราง 1: โครงสร้างข้อมูลและการใช้พื้นที่เก็บข้อมูลเพื่อค้นหา LCS

ขั้นตอนวิธี	Pre-processing	LCS searching
Efficient DP-LCS	Matrix C_{mxn} (2D-Array)	-
NewLCS [6] 2007	1D-Array P (ขนาด Ln)	$P_1, P_2, .., P_i$; $\|P_i\| \le n$
SA-MLCS [7] 2014	Graph (cn nodes)	3 Queues; $\|Q\| \le n$
Improved DP with Space Reduction (IDPSR-LCS) นำเสนอในหัวข้อ 3	n-MultipleList L; $\|L_i\| \le k$ (k = length of LCS)	-
	Space Reduction cn < kn (c=constant)	

3. ขั้นตอนวิธี IDPSR-LCS ที่นำเสนอในงานวิจัยนี้

งานวิจัยนี้นำเสนอขั้นตอนวิธี DP ใหม่ ที่ใช้ในการค้นหา LCS (Longest Common Subsequence) ชื่อ IDPSR (Improved DP with Space Reduction) ที่ปรับปรุงมาจากขั้นตอนวิธี DP เดิมที่มีประสิทธิภาพ เนื่องจากในขั้นตอนของการจัดเตรียมข้อมูล (Pre-processing) ของวิธี DP เดิม จะใช้เมตริกซ์ขนาด mxn หรืออาเรย์ 2 มิติ (แสดงในตาราง 1) ซึ่งเป็นข้อจำกัดในทางปฏิบัติ เมื่อ m,n มีค่ามากๆ เพราะตัวแปรอาเรย์ขนาด mxn ต้องใช้พื้นที่จัดเก็บค่าติดกันในหน่วยความจำในขณะประมวลผลโปรแกรม ซึ่งปกติจะจองเนื้อที่อาเรย์ได้ไม่เกิน 800x800 (m, n < 800)

ดังนั้นงานวิจัยนี้ จึงนำเสนอขั้นตอนวิธี IDPSR ที่มีการใช้หน่วยความจำแบบ Dynamic เพื่อแก้ปัญหาดังกล่าว ด้วยโครงสร้างข้อมูลแบบอาเรย์ของลิสต์ชนิดพิเศษคือ Mlist L_i (i = 1, 2,…, n) ที่มีขนาดของลิสต์เป็นแบบ Dynamic (≤ k) เมื่อค่า k เป็นความยาวของ LCS และเสนอขั้นตอนวิธี Improved DP ที่มีประสิทธิภาพยิ่งขึ้น โดยปรับและเพิ่มฟังก์ชันให้สอดคล้องกับการประมวลผลข้อมูลที่จัดเก็บแบบ Mlist ซึ่งแบ่งเป็น 2 ส่วน คือ 1. การจัดเตรียมข้อมูล Mlist (เสนอในหัวข้อ 3.1) และ 2. การค้นหา LCS และ MLCS จาก Mist (เสนอในหัวข้อ 3.2)

3.1 ขั้นตอนวิธีการเตรียม Dynamic Mlist

ขั้นตอนวิธี 3.1 เป็นการสร้าง Mlist ที่เป็นการจัดเตรียมข้อมูล (Pre-processing) ด้วยวิธีการคำนวณค่าความยาวร่วมทั้งหมดเช่นเดียวกับขั้นตอน DP-LCS แต่ในที่นี้ จะจัดเก็บลงใน Mlist เฉพาะค่าใหม่ที่ไม่ซ้ำ (เพื่อลดพื้นที่ และเพื่อค้นหา LCS และ MLCS ได้เร็วขึ้น) พร้อมตำแหน่งของ X และตัวชี้ back

ขั้นตอนวิธี 3.1: การสร้าง Dynamic Mlist ของ IDPSR-LCS

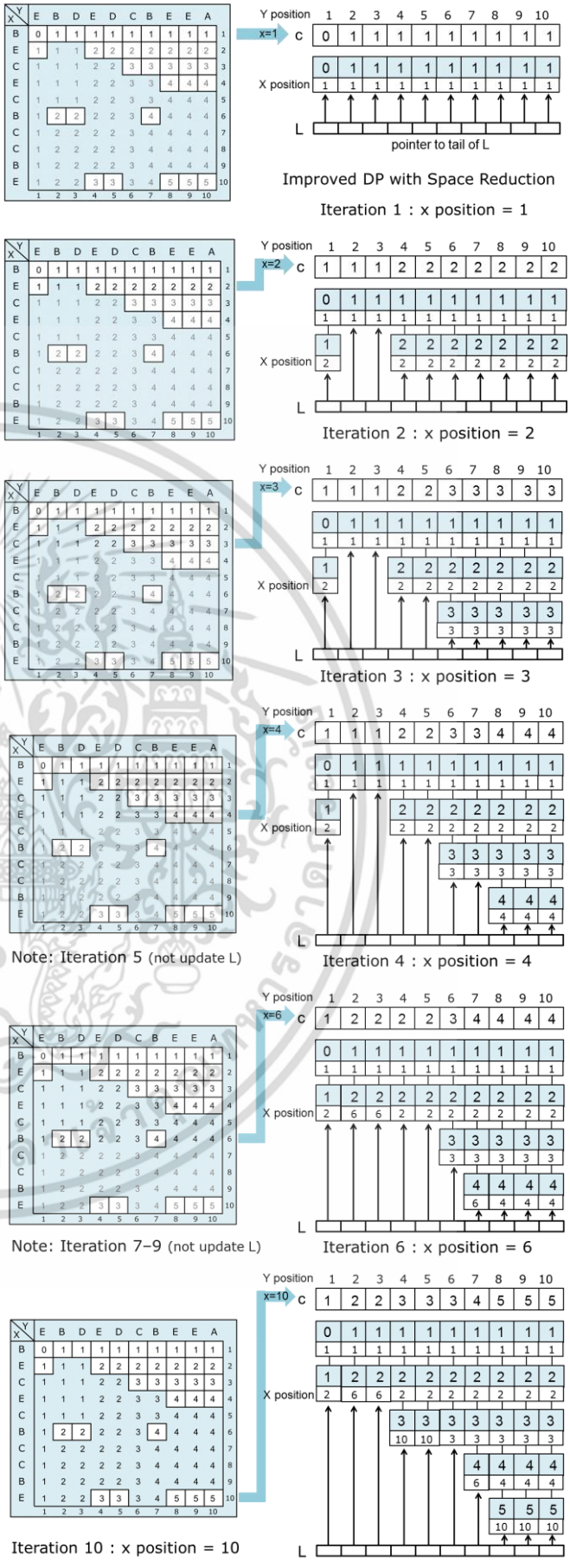
```
for (x=1; x<=m; x++)
  for(y=1; y<=n; y++)
    if(X[x]=Y[y]) c[y] = L[y-1]->c+1;
    else c[y] = max(c[y-1], L[y]->c);
  end for y
  UpdateMlist (Ln, cn, x);
end for x
return ImprovedMlist (Ln, n, k);
[Note: Each node of Mlist L has 3 fields (c, x, back)]
```

UpdateMlist (L_n, c_n, x)

```
for(y=1; y<=n; y++)
  if(c[y] != L[y]->c)          // add new node in L[y]
    allocate new node (N); // memory allocate
    N->c=c[y]; N->x=x; N->back=L[y]; L[y]=N;
  end if
end for y
```

ImprovedMlist (L_n, f, k)

```
y=f; while(L[y]->c = k) y=y-1;  // min y with LCS=k
f=y; for(y=f; y>0; y--)
       c=L[y]; p=L[y+1]; // c=current, p=previous
       while(c->x > p->x) // adjust tail & free node
          L[y]=c->back; free(c); c=L[y];
       end while
     end for y;
return f+1;
```

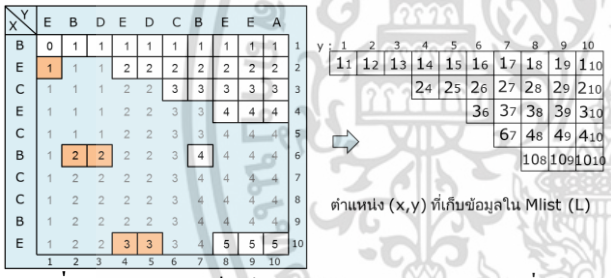


ภาพที่ 5: การเก็บข้อมูลเพื่อการค้นหา LCS ในแบบ IDPSR-LCS

ให้ c[y] แทนค่าความยาวของสายอักษรร่วมจากสายอักษร X และ Y ดังนั้นสำหรับแต่ละค่าของ x (= 0,1,2,…,m) และทุกค่า y (= 0,1,2,…, n) คำนวณ c[y] ก่อนจัดเก็บบางค่าใน Mlist L[y]

c[y] = ค่าใน L[y-1]+1 ถ้า X[x] = Y[y]
= max {c[y-1], L[y]} ถ้า X[x] ≠ Y[y]

ตัวอย่างเช่น ภาพที่ 5 แสดงการสร้าง Mlist L ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA ดังนี้ ใน Iteration 1 (พิจารณา X ที่ตำแหน่ง x=1) ทุกค่าของ c[y], y=1,2,…,m จะถูกจัดเก็บใน L[y] ทุกค่าของ y พร้อมตำแหน่งแรกของ x=1 ต่อมาใน Iteration 2 (x=2) จะมีเพียงบางค่าใหม่ของ c[y] ที่แตกต่าง และถูกจัดเก็บเพิ่มใน L[y] (คือ y=1,4-10) และในทำนองเดียวกัน จะดำเนินการซ้ำใน Iteration 3-10 (ซึ่งโดยปกติเมื่อ Iteration เพิ่มขึ้น ค่าไม่ซ้ำที่จัดเก็บในลิสต์จะมีน้อยลง เช่น Iteration 3 (เก็บเพิ่ม 5 ค่า), Iteration 4 (เก็บเพิ่ม 3 ค่า) และในบาง Iterations ไม่ต้องเก็บค่า(ใหม่)เพิ่ม เช่น Iterations 5, 7-9) ดังนั้นการเก็บเฉพาะค่าที่ไม่ซ้ำดังกล่าว ทำให้สามารถลดพื้นที่จัดเก็บได้ และฟังก์ชันสุดท้าย (ImprovedMlist) คือการปรับ Mlits L เพื่อตัดบางโหนดที่ไม่ใช่ส่วนของ LCS ออก ดังแสดงในภาพที่ 6 เช่น โหนดในตำแหน่ง (x, y) = (10,5), (10,4), (6,3), (6,2), (2,1) ทำให้สามารถลดพื้นที่ได้ถึง 72% (ในตัวอย่างนี้) ดังแสดงในภาพที่ 6 ซึ่งใช้พื้นที่จัดเก็บข้อมูลเพียง 28% และการเก็บแบบ Mlist ไม่จำเป็นต้องใช้พื้นที่ติดกันในหน่วยความจำ



ภาพที่ 6: ตัวอย่างการเก็บข้อมูลแบบ Dynamic ใน Mlist ที่ลด 72%

3.2 ขั้นตอนวิธีค้นหา LCS และ MLCS จาก Dynamic Mlist

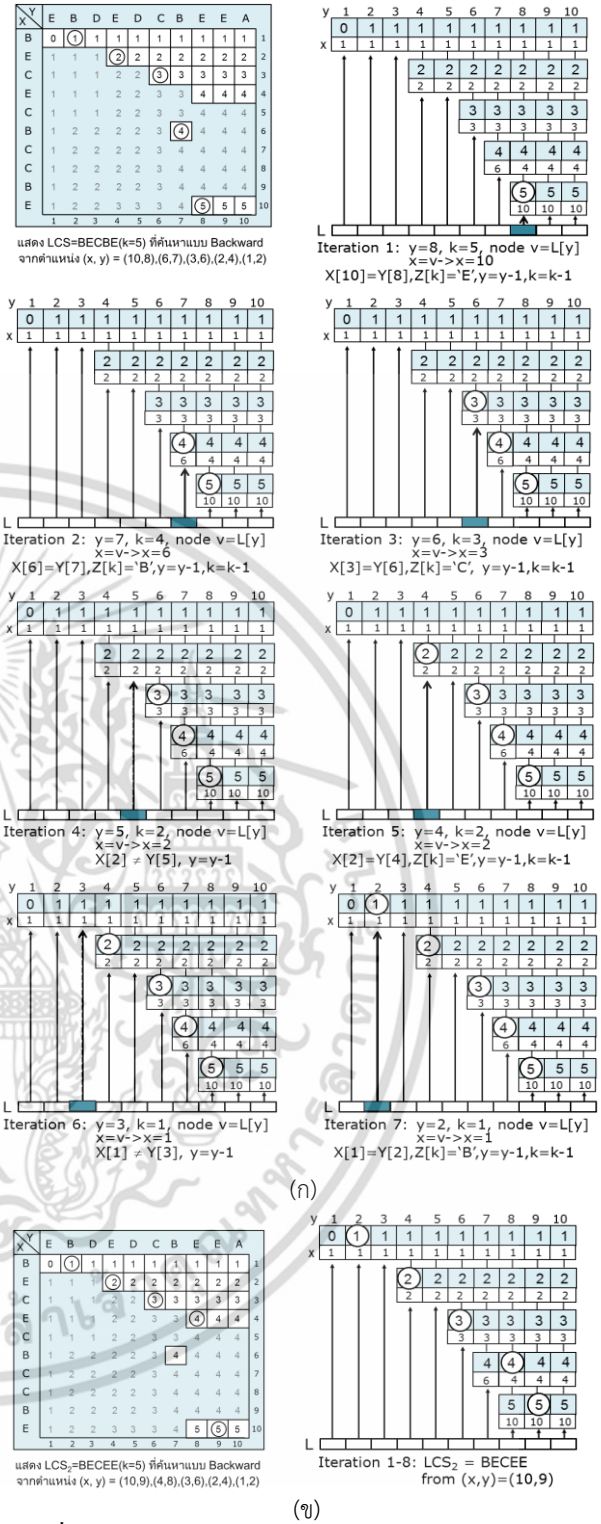
ขั้นตอนวิธี 3.2 เป็นการค้นหา LCS จาก Mlist (ในขั้นตอนวิธี 3.1) โดยมี Input คือ list L_n, strings X_m, Y_n, ความยาว LCS = k พร้อมตำแหน่ง x, y และ Output คือ LCS Z_k

ให้ v แทนโหนดใดๆ ที่เริ่มจาก tail ของ list L[y] (คือ v=L[y])

ตัวอย่างเช่น ภาพที่ 7(ก) แสดงการค้นหา LCS ค่าแรก ที่เริ่มจาก (x, y)=(10, 8) ของ Mlist L_n ใน Iteration 1 (y=8, k=5) โหนดแรก v=L[y]=L[8] จะได้ x = v->x = 10 และ Z[5]='E'

ขั้นตอนวิธี 3.2: การค้นหา LCS จาก Mlist ของวิธี IDPSR-LCS

```
BackLCSfromMlist (Ln,Xm,Yn,Zk,x,y,k)
  while (k>0) do
    v=L[y]; x=v->x;
    if((X[x]=Y[y])&(v->c=k)) Z[k]=X[x];k=k-1;y=y-1;
    else // either Xx≠Y(of LCS1) or find Xx=Yy(of LCSother)
      while(v->c > k) v=v->back; // for other LCS
      x=v->x;
      if((X[x]=Y[y])&(v->c=k))Z[k]=X[x];k=k-1; y=y-1;
      else y=y-1;  // Xx≠Yy goto next L[y] for k
    end if-else;
  end while;
end.
```



ภาพที่ 7: การการค้นหา LCS_1 และ LCS_2 แบบ IDPSR จาก Mlist

ในทำนองเดียวกัน Iteration 2 (y=7, k=4) จะได้ Z[4]='B' และ Iteration 3 (y=6, k=3) จะได้ Z[3]='C' ส่วน Iteration 4 (y=5, k=2) โหนด v=L[y]=L[5] และ x=v->x=2 แต่กรณีนี้ X[2]≠Y[5] จึงข้ามไปยังลิสต์ L[y-1] แต่ยังคงค่า k=2 และจะดำเนินการในทำนองเดียวกันใน Iterations ที่เหลือ ซึ่งจะได้ Z[2]='E' ใน Iteration 5 และ Z[1]='B' ใน Iteration 7 ส่วนภาพที่ 7(ข) แสดงการค้นหา LCS อีกค่า ที่เริ่มจาก k=5 ที่ตำแหน่ง (x, y)=(10, 9)

4. การเปรียบเทียบ LCS วิธีต่างๆ และผลการทดลอง

ในขั้นตอนการเตรียมข้อมูลของวิธี IDPSR-LCS ของข้อมูล X (ขนาด m), Y (ขนาด n) ที่เสนอในหัวข้อ 3 คือ คำนวณทุกค่า mxn ของ c_{ij} (ใน DP เดิม) แต่เก็บชั่วคราวใน c_j และสร้างลิสต์ Mist L_j (ขนาด n) ทุกค่าของ Y คือ j = 1, 2, 3, … , n เพื่อเก็บเฉพาะค่าที่แตกต่าง(หรือเมื่อพบค่าร่วมใหม่) เพื่อเป็น x-pointer jumping ในค่าของ X (ในขั้นตอนค้นหา LCS) นอกจากนั้นในขั้นนี้ จะเพิ่มฟังก์ชันที่ปรับลิสต์ (ImprovedMlist) ในส่วนท้ายๆ ของทุกลิสต์ ให้เหลือเฉพาะค่าที่เป็น LCS ทำให้เก็บค่าน้อยลง (ประมาณ cn ค่า ดังแสดงในภาพที่ 6 เมื่อ c เป็นค่าคงที่) ดังนั้นจึงมีความซับซ้อนด้านเวลา (Time Complexity) = O(mn) และความซับซ้อนด้านพื้นที่ (Space Complexity) = O(n)

ในส่วนการค้นหาสายอักษรร่วมที่ยาวที่สุดหลายชุด (MLCS) ด้วยวิธี IDPSR-LCS จะประมวลผลได้เช่นเดียวกับวิธี DP เดิมที่มีความซับซ้อนด้านเวลา O(n) ในกรณีดีที่สุด และ O(m+n) กรณีช้าที่สุด ส่วนวิธี IDPSR จะค้นหา LCS ด้วยความซับซ้อน O(n) เพราะ LCS ชุดแรก จะพบได้โดยตรงจากปลายลิสต์ ส่วน MLCS อื่นๆ จะใช้ค่าต่อไปของแต่ละลิสต์ ที่จะไปถึงได้ด้วย x-pointer jumping ที่จัดเตรียมไว้ในลิสต์ ซึ่งแตกต่างจาก DP เดิม ที่จะต้องเลื่อนไปทีละค่าของอาเรย์ X และ Y

ตาราง 2: ความซับซ้อน Time และ Space ของขั้นตอนวิธี LCS

ขั้นตอนวิธี	Pre-processing		LCS searching	
	Time	Space	Time	Space
Original DP-LCS	O(mn)	O(mn)	O(m+n)	O(1)
New LCS 2007 [6]	O(mn)	O(n)	O(ckn)	O(ck)
SA-MLCS 2014 [7]	O(mn)	O(n)	O(kn)	O(ck)
IDPSR-LCS ที่เสนอ	O(mn)	O(n)	O(n)	O(1)

ตาราง 3: การเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธีต่างๆ

Original DP-LCS	
ข้อดี	สามารถหา LCS ได้โดยตรงจากค่า c_{ij} ซึ่งเร็วมาก O(m+n)
ข้อจำกัด	ใช้พื้นที่มากเพื่อเก็บค่าเมตริกซ์ c เป็นอาเรย์ 2 มิติ (mxn)
New LCS 2007 [6]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Matching point ในอาเรย์ 1 มิติ
ข้อจำกัด	ใช้เวลาในการหา LCS O(ckn) นานกว่า DP เพราะต้องตรวจสอบเงื่อนไข pre/post ของ LCS ทุกค่าและ CS อื่นๆ
SA-MLCS 2014 [7]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Dominant point ในกราฟ
ข้อจำกัด	ใช้เวลาในการหา LCS O(kn) เร็วกว่า [6] เพราะใช้คิว 3 คิว (New, Open, Closed) ช่วยลดการตรวจความสัมพันธ์ที่ซ้ำซ้อน แต่นานกว่า DP เพราะต้องหาความสัมพันธ์ของ LCS ทุกค่า (MLCS) และ CS อื่นๆ ที่เกี่ยวข้อง
IDPSR-LCS ที่เสนอในงานวิจัยนี้	
ข้อดี	มีข้อดีเหมือน DP คือหา LCS ได้โดยตรง แต่เร็วกว่า O(n) มีข้อดีเหมือน [6,7] ที่สามารถลดพื้นที่เพราะเก็บข้อมูลเป็นอาเรย์ 1 มิติของลิสต์แทน (โดยเก็บเฉพาะค่า c_{ij} ที่แตกต่าง)
ข้อจำกัด	ยังคงใช้เวลาเตรียมข้อมูลในส่วน Pre-processing เป็น O(mn) เหมือน DP และวิธีอื่นๆ [6, 7]

ตาราง 4: ผลการทดลองเปรียบเทียบการลดพื้นที่จัดเก็บข้อมูล

m=n	k	Matrix C_{mxn}	Mlist	%พื้นที่เก็บ	%พื้นที่ลด
100	22	100x100	1077	11%	89%
500	120	500x500	29433	12%	88%
1000	234	1000x1000	119240	12%	88%
10000	2401	10000x10000	11913820	12%	88%

ตาราง 2-3 แสดงการเปรียบเทียบข้อดีของวิธี LCS ต่างๆ โดยวิธี IDPSR-LCS สามารถรักษาข้อดีของวิธี DP เดิม (ในช่วงการค้นหา LCS ที่เร็ว) และมีข้อดีเหมือนวิธีอื่นๆ (New-LCS [6] และ SA-MLCS [7]) ที่ลดพื้นที่จัดเก็บข้อมูลในขั้นตอน Pre-processing จากผลการทดลอง ตาราง 4 แสดงเปอร์เซนต์พื้นที่จัดเก็บข้อมูลที่ลดลงของวิธี IDPSR เทียบกับวิธี DP เดิม

5. สรุปผล

งานวิจัยนี้ นำเสนอขั้นตอนวิธี IDPSR-LCS ซึ่งพัฒนามาจากขั้นตอนวิธีไดนามิคโปรแกรมมิง (DP) ในการค้นหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS) ของข้อมูลสายอักษร 2 ชุด (X_m, Y_n) ขั้นตอนวิธีที่นำเสนอใหม่ เป็นวิธีที่สามารถลดพื้นที่ที่ใช้ในการเก็บข้อมูลเพื่อค้นหา LCS = O(n) ด้วยการใช้อาเรย์ของลิสต์ที่เก็บเฉพาะข้อมูลที่ไม่ซ้ำและเก็บสมาชิกแบบไดนามิค แทนอาเรย์ 2 มิติแบบเดิมที่ใช้พื้นที่ในการเก็บข้อมูล = O(mn) ซึ่งในทางปฏิบัติจะประมวลผลได้ไม่เกิน m, n = 800 เพราะอาเรย์ต้องใช้หน่วยความจำเก็บค่าที่ต่อเนื่องกัน (< 800 x 800) ส่วนงานวิจัยที่จะทำต่อไป จะเป็นการประยุกต์ใช้ขั้นตอนวิธี IDPSR-LCS ที่มีประสิทธิภาพนี้ ในการตรวจจับ Web Phishing

เอกสารอ้างอิง

[1] P. P. Lin and K. Jules, "An intelligent system for monitoring the microgravity environment quality on-board the International Space Station," *IEEE Trans. on Instrumentation and Measurement*, vol. 51, no. 5, pp. 1002-1009, 2002.

[2] R.A.C. Campos and F.J.Z Martinez, "Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem," *9th International Conference on Electrical Engineering, Computing Science and Automatic Control*, pp. 1 – 4, 2012.

[3] D. Abraham and N.S. Raj, "Approximate string matching algorithm for phishing detection," *2014 International Conference on Advances in Computing, Communications and Informatics* (ICACCI), pp. 2285 – 2290, 2014.

[4] D. Frolova, H. Stern and Berman, S, "Most Probable Longest Common Subsequence for Recognition of Gesture Character Input," *IEEE Trans. on Cybernetics*, vol. 43, no.3, pp. 871 - 880, 2013.

[5] J. Liaw and et al., "Recognition of the Ambulance Siren Sound in Taiwan by the Longest Common Subsequence," *2013 IEEE international conference on Systems, Man, and Cybernetics*, pp.3824-3828, 2013.

[6] X. Xiang, D. Zhang, and J. Qin. "An New Algorithm for the Longest Common Subsequence Problem," *2007 International Conference on Computational Intelligence and Security Workshops*, pp. 112 – 115, 2007.

[7] J. Yang, Y. Xu, Y. Shang, and G. Chen, "A Space-Bounded Anytime Algorithm for the Multiple Longest Common Subsequence Problem," *IEEE Trans. on Knowledge and Data Engineering*, vol. 26, no. 11, pp. 2599 - 2609, 2014.

14 - 17 December 2016, Chiang Mai, Thailand

The 20th International Computer Science & Engineering Conference 2016

"Smart Ubiquitous Computing & Knowledge"

ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุดแบบขนานที่มีประสิทธิภาพด้วยเทคนิคไพน์ไลน์นิง
Efficient Parallel Longest Common Subsequence with Pipelining Technique

ศักดิ์พันธุ์ แดงมณี[1] และ จีรพร วีระพันธุ์[2]
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, กรุงเทพมหานคร
E-mail: sakpan.da@up.ac.th[1], jeeraporn.we@kmitl.ac.th[2]

บทคัดย่อ

การหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS: Longest Common Subsequence) ของข้อมูล 2 ชุด (X, Y) ขนาด m และ n ที่เร็วและใช้พื้นที่จัดเก็บข้อมูลน้อยๆ มีบทบาทสำคัญอย่างมากในยุคปัจจุบันที่ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) อันเป็นผลเนื่องมาจากการใช้งานอุปกรณ์อิเล็กทรอนิกส์ของยุค IoT (Internet of Things) และการใช้งานแอพลิเคชันสังคมออนไลน์ งานวิจัยเพื่อพัฒนาขั้นตอนวิธี LCS ที่ผ่านมามุ่งเน้นที่การลดพื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำที่น้อยกว่าแบบเดิมที่เป็นอาร์เรย์ 2 มิติขนาด nxn ซึ่งต้องใช้พื้นที่ติดกันและเป็นข้อจำกัดในการประมวลผล เมื่อ n มีค่ามาก แต่ยังคงมีความซับซ้อนด้านเวลาของขั้นตอนวิธี LCS เป็น $O(n^2)$ ดังนั้นบทความวิจัยนี้ จึงนำเสนอขั้นตอนวิธี LCS แบบขนาน (Parallel LCS: PLCS) ที่จะประมวลผลด้วยเทคนิคไพน์ไลน์นิงที่มีประสิทธิภาพเร็วขึ้นเป็น O(n) และใช้พื้นที่ในการจัดเก็บข้อมูลได้อย่างเหมาะสมในขั้นตอนการจัดเตรียมข้อมูลที่จัดเก็บลงในอาร์เรย์ 1 มิติของลิสต์ พร้อมด้วยการค้นหา LCS หลายค่าแบบขนาน ซึ่งทำให้สามารถประมวลผลได้เร็วขึ้นและใช้พื้นที่ในการเก็บข้อมูลได้อย่างเต็มประสิทธิภาพ

คำสำคัญ: ลำดับร่วมเหมือนที่ยาวที่สุด (LCS), ขั้นตอนวิธี LCS แบบขนาน (PLCS), เทคนิคไพน์ไลน์นิงสำหรับการประมวลผล PLCS

Abstract

Fast LCS (Longest Common Subsequence) of X(m) and Y(n) with space reduction is really effect in the Big Data Era and IoT (Internet of Things), where recently Big Data are increasing very fast and unlimited by consuming of various electronic devices and social applications. A crucial limitation of the existing LCS approaches is the contiguous-space requirement for the nxn 2D-array in pre-processing. Existing LCS algorithms focused on the improved memory-space by storing only important data in the pre-processing with the 2D-array processing in $O(n^2)$. Therefore, this paper proposes the new parallel LCS (PLCS) algorithm for the efficient parallel pre-processing data in O(n) using a pipelining technique. Our efficient PLCS with space reduction can store large n with an array of multiple lists in the pre-processing step. In addition, our parallel searching time for finding the LCSs is also efficient.

Keywords: Longest Common Subsequence (LCS), Parallel LCS (PLCS), Pipelining technique for PLCS.

1. คำนำ

ในยุค IoT (Internet of Things หรือ Internet of Everything) มนุษย์นำอุปกรณ์อิเล็กทรอนิกส์มาใช้ในชีวิตประจำวันมากขึ้นเป็นทวีคูณ อุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ ถูกเพิ่มความสามารถให้สามารถประมวลผลอย่างง่ายตามความสามารถที่พึงมีของอุปกรณ์และยังสามารถเชื่อมต่อกับเครือข่ายต่าง ๆ เพื่อสามารถแลกเปลี่ยนข้อมูลกับอุปกรณ์อื่น อีกทั้งบนเครือข่ายอินเตอร์เน็ตมีการใช้งานแอพลิเคชันสังคมออนไลน์ ข้อมูลที่เกิดจากการใช้งานอุปกรณ์อิเล็กทรอนิกส์และแอพลิเคชันเหล่านี้ทำให้ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) ขั้นตอนวิธีต่างๆ ที่มีอยู่เดิมอาจไม่มีประสิทธิภาพที่เร็วเพียงพอกับการประมวลผลข้อมูลที่มีขนาดใหญ่ดังกล่าว ตัวอย่างเช่น ขั้นตอนวิธีหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS: Longest Common Subsequence) ซึ่งเป็นขั้นตอนวิธีที่ถูกนำมาประยุกต์ใช้ในงานมากมายหลายประเภท เช่น การเปรียบเทียบการเรียงตัวของ DNA หรือโปรตีนของผู้ป่วยกับคนปกติเพื่อตรวจหาความผิดปกติอย่างละเอียด[1] การตรวจสอบการคัดลอกโปรแกรมของนักศึกษา[2] การตรวจสอบเว็บไซต์ว่า เป็นเว็บไซต์ Phishing หรือไม่ [3] การตรวจสอบรูปแบบการป้อนอักขระด้วยท่าทาง (Gesture Character Input) [4] การตรวจสอบเสียงไซเรนของรถพยาบาล [5] เป็นต้น จากตัวอย่างการนำขั้นตอนวิธี LCS มาใช้ในงานตรวจสอบความเหมือนของข้อมูลชนิดต่างๆ ดังกล่าวอย่างแพร่หลาย เป็นผลอันเนื่องมาจากข้อดีของขั้นตอนวิธี LCS คือสามารถประมวลผลได้รวดเร็วกว่าและแม่นยำกว่าขั้นตอนวิธีแบบฮิวริสติกอื่น นอกจากนั้นยังมีกระบวนการและฟังก์ชันการคำนวณค่าที่ไม่ซับซ้อนจนเกินไป แต่หากยังต้องการพื้นที่จัดเก็บข้อมูลเพื่อการประมวลผลค่อนข้างมาก และในยุคที่ข้อมูลมีขนาดใหญ่มากๆ นี้ การใช้ขั้นตอนวิธี LCS แบบเดิมอาจส่งผลให้การประมวลผลไม่เสร็จทันในเวลาที่ต้องการ

การอิมพลีเมนท์ขั้นตอนวิธี LCS ของงานวิจัยในอดีตที่ผ่านมาเน้นเรื่องการลดพื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำเพราะจะต้องมีการสร้างเมตริกซ์ขนาด nxn โดยใช้โครงสร้างข้อมูลแบบอาร์เรย์ 2 มิติสำหรับใช้เก็บข้อมูลความยาวร่วมของข้อมูล 2 ชุด X และ Y (ขนาด m และ n) ซึ่งทางปฏิบัติพบว่า จะไม่สามารถประมวลผลเมื่อ n มีค่ามากได้ เนื่องจากอาร์เรย์ต้องใช้หน่วยความจำเก็บค่าที่ต่อเนื่องกัน (nxn ไม่เกิน 800x800) และจากการค้นคว้าของผู้วิจัยพบว่า ขั้นตอนวิธี LCS แบบใหม่ของ X. Xiang [6] เสนอการลดพื้นที่โดยมีการเก็บข้อมูลจุดเหมือน (Matching points) ลงในอาร์เรย์มิติเดียว ต่อมาขั้นตอนวิธี MLCS ของ J. Yang [7] จัดเก็บข้อมูลจุดเหมือนในรูปแบบข้อมูลโครงสร้างต้นไม้ที่เหมาะ

สำหรับข้อมูลหลายชุด และสุดท้ายขั้นตอนวิธี IDPSR-LCS [8] ของผู้วิจัยเองที่ลดการเก็บข้อมูลแบบอาร์เรย์ 2 มิติ ลงในอาร์เรย์ของลิสต์โดยไม่เก็บจุดที่เป็นข้อมูลซ้ำในแต่ละคอลัมน์ ตารางที่ 1-2 แสดงการเปรียบเทียบขั้นตอนวิธี LCS ซึ่งพบว่าทุกวิธียังคงมีความซับซ้อนของขั้นตอนวิธี LCS ในส่วนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) เป็น $O(n^2)$ นอกจากนั้นยังพบว่า มีงานวิจัยแบบขนานที่เน้นการประมวลผลที่เร็วขึ้น เพื่อรองรับความต้องการประมวลผลข้อมูลที่มีจำนวนมหาศาลให้ได้ผลลัพธ์ที่รวดเร็วยิ่งขึ้น ดังนั้นขั้นตอนวิธี LCS แบบขนาน (Parallel LCS) จึงถูกนำเสนอขึ้น เช่น ขั้นตอนวิธี Quick-DPPAR ของ Q. Wang [9] และขั้นตอนวิธี ProMLCS ของ J.Yang [10] แต่อย่างไรก็ตามประสิทธิภาพของ MLCS แบบขนานในงานวิจัยที่ผ่านมามุ่งเน้นไปที่การประมวลผลข้อมูลหลายชุด (X_1 X_2, ..., X_s) ส่วนการออกแบบขั้นตอนวิธี LCS แบบขนานสำหรับข้อมูล 2 ชุด (X, Y) ยังมีข้อจำกัดในส่วนของขั้นตอนการจัดเตรียมข้อมูล ซึ่งทำได้ค่อนข้างยาก เพราะค่าของแต่ละสมาชิกจะต้องคำนวณจากผลลัพธ์ที่ต่อเนื่องกัน

จากปัญหาข้างต้น ผู้วิจัยจึงได้เสนอขั้นตอนวิธี PLCS-SR ที่เป็นขั้นตอนวิธี LCS แบบขนานที่มีประสิทธิภาพเร็วขึ้นเป็น O(n) ด้วยการลดเวลาส่วนใหญ่ที่ใช้ในการประมวลผล คือขั้นตอนการจัดเตรียมข้อมูล (Pre-processing) ด้วยเทคนิคไพน์ไลน์นิง พร้อมด้วยการลดพื้นที่ในการจัดเก็บค่าความยาวของ LCS โดยปรับโครงสร้างการจัดเก็บที่เสนอในงานวิจัยเดิม [8] ให้เหมาะสมเพื่อการประมวลผลแบบขนานที่มีประสิทธิภาพ และในส่วนของขั้นตอนการค้นหา LCS หลายค่า ก็จะสามารถค้นหาได้เร็วขึ้นด้วยการประมวลผลแบบขนาน (หนึ่งหน่วยประมวลผลต่อหนึ่ง LCS) เพื่อลดเวลาในการค้นหา LCS ทั้งหมดด้วยความซับซ้อนด้านเวลาเท่ากับ O(n)

เนื้อหาในหัวข้อต่อไป เป็นการแสดงผลงานวิจัยที่เกี่ยวข้องในการแก้ปัญหา LCS แบบหนึ่งหน่วยประมวลผล (Sequential Processing) และแบบหลายหน่วยประมวลผล (Parallel Processing) หัวข้อที่ 3 เป็นการเสนอขั้นตอนวิธี LCS แบบขนาน (PLCS: Parallel LCS) ด้วยเทคนิคไพน์ไลน์นิง และแบบ PLCS-SR ที่มีการลดพื้นที่จัดเก็บข้อมูล (Space Reduction) ที่ใช้หน่วยประมวลผล $p \le n$ หน่วย ส่วนหัวข้อที่ 4 แสดงการเปรียบเทียบประสิทธิภาพของความซับซ้อนด้านเวลาและพื้นที่ (Time and space complexity) และความเร็วที่เพิ่มขึ้น (Speedup) ของขั้นตอนวิธีแบบขนาน และสรุปผลงานวิจัยในหัวข้อที่ 5

2. งานวิจัยที่เกี่ยวข้อง

ปัญหาการค้นหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS: Longest Common Subsequence) เป็นการหาสายอักษรร่วมที่อยู่ในสายอักษร 2 ชุด คือ X และ Y ซึ่ง LCS เป็นปัญหาพื้นฐานของ MLCS สำหรับข้อมูลหลายชุด (X_1, X_2, ..., X_s) โดยที่ $X = \langle x_1, x_2, \dots, x_m \rangle$ เป็นสายอักษรชุดหนึ่งที่มีความยาว m ตัวอักษร และ $Y = \langle y_1, y_2, \dots, y_n \rangle$ เป็นสายอักษรอีกชุดอีกหนึ่งที่มีความยาว n ตัวอักษร โดยในที่นี้จะสมมุติให้ m=n เพื่อความสะดวกในการแสดงความซับซ้อนด้านเวลาขั้นตอนวิธี

ในปัจจุบันมีหลายงานวิจัยที่พัฒนาขั้นตอนวิธี LCS [6-8] ด้วยข้อดีและข้อจำกัดที่แตกต่างกัน ดังแสดงการเปรียบเทียบไว้ในตารางที่ 1-2 คือ ขั้นตอนวิธี DP-LCS ดั้งเดิม, ขั้นตอนวิธี New LCS ของ X. Xiang [6], ขั้นตอนวิธี SA-MLCS ของ J. Yang [7] ที่มุ่งเน้นปัญหา MLCS ของข้อมูลหลายชุด และขั้นตอนวิธี IDPSR-LCS [8] จากตารางความซับซ้อนด้านเวลาและพื้นที่ของขั้นตอนวิธี LCS ต่างๆ จะเห็นได้ว่าทุกๆ ขั้นตอนวิธี LCS แบบใหม่ๆ จะลดพื้นที่ในขั้นตอนจัดเตรียมข้อมูลก่อนการประมวลผล แต่ในขั้นตอนของการค้นหา LCS หลายชุดนั้น ขั้นตอนวิธี LCS ใหม่จะต้องใช้พื้นที่หน่วยความจำเพิ่มขึ้นในส่วนของการค้นหาดังกล่าว ยกเว้นขั้นตอนวิธี IDPSR-LCS ในขณะที่ความซับซ้อนด้านเวลาในขั้นตอนการจัดเตรียมข้อมูลก่อนการประมวลผลยังคงเป็นข้อจำกัดคือทุกๆ ขั้นตอนวิธี LCS จะใช้เวลาเท่ากับ $O(n^2)$ ส่วนเวลาที่ใช้ในการค้นหา LCS จะขึ้นกับจำนวน LCSs เช่น O(n) ถ้ามี LCSs เป็นจำนวนจำกัด และ O(cn) ถ้ามี LCSs อยู่ c จำนวน

ตาราง 1: ความซับซ้อน Time และ Space ของขั้นตอนวิธี LCS [6-8]

ขั้นตอนวิธี	Pre-processing		LCS searching	
	Time	Space	Time	Space
DP-LCS ดั้งเดิม	$O(n^2)$	$O(n^2)$	O(cn)	O(1)
New LCS 2007 [6]	$O(n^2)$	O(n)	O(ckn)	O(ck)
SA-MLCS 2014 [7]	$O(n^2)$	O(n)	O(kn)	O(ck)
IDPSR-LCS 2016 [8]	$O(n^2)$	O(n)	O(cn)	O(1)

ตาราง 2: การเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธี LCS [6-8]

New LCS 2007 [6]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Matching points ในอาเรย์ 1 มิติ
ข้อจำกัด	ใช้เวลาในการหา LCS O(ckn) นานกว่า DP เพราะต้องตรวจสอบเงื่อนไข pre/post ของ LCS ทุกค่าและ CS อื่นๆ
SA-MLCS 2014 [7]	
ข้อดี	ลดพื้นที่โดยเก็บเฉพาะค่า Dominant points ในกราฟ ซึ่งจะเหมาะสำหรับหา MLCS ของข้อมูลหลายชุด (X_1, X_2, ..., X_k)
ข้อจำกัด	ใช้เวลาในการหา LCS O(kn) เร็วกว่า [6] เพราะใช้คิว 3 คิว (New, Open, Closed) ช่วยลดการตรวจความสัมพันธ์ที่ซ้ำซ้อน แต่นานกว่า DP เพราะต้องหาความสัมพันธ์ของ LCS ทุกค่า และ CS อื่นๆ ที่เกี่ยวข้อง
IDPSR-LCS 2016 [8]	
ข้อดี	มีข้อดีเหมือน DP คือหา LCS ได้โดยตรง ด้วยเวลา O(cn) มีข้อดีเหมือน [6,7] ที่สามารถลดพื้นที่เพราะเก็บข้อมูลเป็นอาเรย์ 1 มิติของลิสต์แทน (โดยเก็บเฉพาะค่า c_{ij} ที่แตกต่าง)
ข้อจำกัด	ยังคงใช้เวลาเตรียมข้อมูลในส่วน Pre-processing เป็น $O(n^2)$ เหมือน DP ดั้งเดิม และวิธีอื่นๆ [6-7]

2.1 ขั้นตอนวิธี IDPSR-LCS [8]

ขั้นตอนวิธี IDPSR (Improved Dynamic Programming with Space Reduction) เป็นขั้นตอนวิธีที่ปรับปรุงจาก DP-LCS ดั้งเดิม โดยทำการลดพื้นที่ในขั้นตอนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) ที่เดิมใช้อาร์เรย์ 2 มิติ C (mxn) ซึ่งเป็นข้อจำกัดเมื่อ m, n มีค่ามาก โดยค่าของแต่ละสมาชิก (c_{ij}) ของอาร์เรย์ C (สำหรับ i=1, 2, 3, ..., m และ j=1, 2, 3, ..., n) คือ

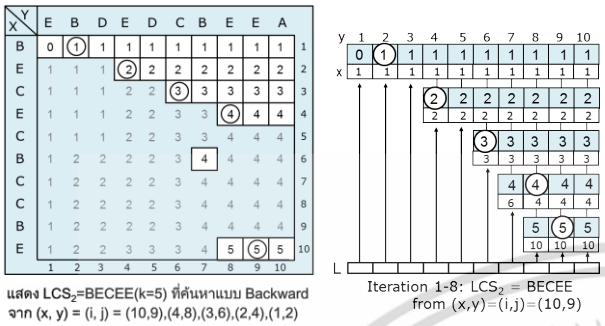
$$c_{i,j} = c_{i-1,j-1} + 1 \quad \text{ถ้า } x_i = y_j$$
$$= \max(c_{i-1,j}, c_{i,j-1}) \quad \text{ถ้า } x_i \neq y_j$$

ดังนั้นขั้นตอนวิธี IDPSR-LCS [8] ได้เสนอการแก้ปัญหาโดยการใช้หน่วยความจำแบบ Dynamic ด้วยโครงสร้างข้อมูลแบบอาร์เรย์ของลิสต์ Mlist L_i (i = 1, 2,..., n) ที่มีขนาดของลิสต์เป็นแบบ Dynamic ($\le$ k) เมื่อค่า k เป็นความยาวของ LCS โดยโครงสร้างข้อมูล Mlist จะจัดเก็บเฉพาะค่าใหม่ที่ไม่ซ้ำเพื่อลดพื้นที่จัดเก็บข้อมูลและเพื่อค้นหา LCS ได้เร็วขึ้น

ในขั้นตอนจัดเตรียมข้อมูล กำหนดให้ c_j แทนค่าความยาวของสายอักษรร่วมจากสายอักษร X และ Y ดังนั้นสำหรับแต่ละค่าของ x_i (i = 0,1,2,...,m) และทุกค่า y_j (j = 0,1,2,..., n) คำนวณ c_j ก่อนจัดเก็บเพียงบางค่า (ที่ตรวจสอบแล้วว่าเป็นค่าที่ไม่ซ้ำ) ลงใน Mlist L_j (ที่แต่ละสมาชิกของลิสต์ประกอบด้วยค่า c, x, และตัวชี้ back) เมื่อค่า c_j คำนวณได้ดังนี้

$$c_j = \text{ค่า c ในลิสต์ } L_{j-1} + 1 \quad \text{ถ้า } x_i = y_j$$
$$= \max(\text{ค่า c ในลิสต์ } L_j, c_{j-1}) \quad \text{ถ้า } x_i \neq y_j$$

จากภาพที่ 1 เป็นตัวอย่างของ Mlist (ภาพขวา) ที่จัดเก็บเพียงบางค่าของ $c_{i,j}$ ของเมตริกซ์ C (สีขาวในภาพซ้าย) คือค่า c ใน Mlist L_j ซึ่งแสดงเปรียบเทียบให้เห็นว่า ขั้นตอนวิธี IDPSR สามารถลดพื้นที่สำหรับจัดเก็บข้อมูลได้ถึง 72% โดยไม่จำเป็นต้องใช้พื้นที่ติดกันในหน่วยความจำและการค้นหา LCS ชุดที่2 จาก LCS ที่มีอยู่ 2 ค่า ก็ทำได้อย่างรวดเร็ว



ภาพที่ 1: การเก็บข้อมูลด้วย Mlist และค้นหา LCS แบบ IDPSR [8]

2.2 ขั้นตอนวิธี Quick-DP และ Quick-DPPAR [9]

ขั้นตอนวิธี Quick-DP เป็นขั้นตอนวิธีที่ทำการปรับปรุงจากขั้นตอนวิธี Dominant point เดิม ซึ่งประกอบด้วยการประมวลผล 2 ส่วน ในส่วนแรกเซตของ dominant points ทั้งหมดจะถูกคำนวณในแต่ละ iteration โดยเริ่มจาก dominant point ที่ 0 (มีอยู่ 1 สมาชิก) เซตของ dominants ที่ k+1 หรือ $D^{(k+1)}$ จะสามารถหาได้จากเซตของ dominants ที่ k หรือ $D^{(k)}$ และในส่วนที่ 2 จะค้นหาเส้นทางที่เป็น LCS ด้วยการคำนวณแบบย้อนกลับ (โดยจะเริ่มจากสมาชิกสุดท้าย) ผ่านเซตของ dominant points ที่ได้มาจากส่วนแรกของขั้นตอนวิธี

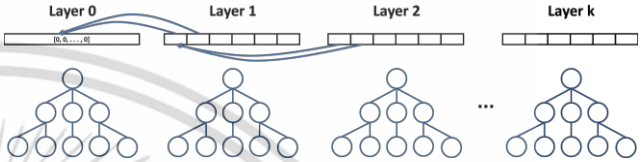
ขั้นตอนวิธี 2.1: Quick-DPAR แบบขนาน [9]

```
Algorithm Quick-DPPAR({a1, a2, . . . , aO}, Σ, Np)
1  Proc0 : Preprocessing; D0={|-1,-1,...,-1|};k=0;
2  while Dk not empty do {
3     Proc0 : distribute element of Dk
      Each processor, Proci, 1 ≤ i ≤ Np, performs :
4        get D_i^k from Proc0;
5        for q ∈ D_i^k do {
6           B = Minima(Par(q, Σ));
7           for s ∈ Σ do{
8              Pars = Pars ∪ {q(s) | q(s) ∈ B};}}
9        Send Pars, s ∈ Σ, to Proc0;
10 Proc0 : calculate Pars = ∪_{1≤i≤Np} Pars, s ∈ Σ;
11 Proc0 : distribute Pars, s ∈ Σ;
   Each processor, Proci, 1 ≤ i ≤ Np, perform :
12    get Pars, s ∈ Σ;
13    D_i^{k+1}= Minima(Pars);
14    send D_i^{k+1} to Proc0;
15 Proc0 : defines Dk+1 = ∪_{1≤i≤Np} D_i^{k+1};
16 k = k-1; }
```

ขั้นตอนวิธีแบบขนาน Quick-DPPAR [9] ได้ถูกพัฒนาขึ้นโดยการใช้เทคนิควิธี Divide-and-Conquer โดยให้โพรเซสเซอร์ $Proc_0$ แบ่งเซตของ N dominant points ออกเป็นสองซับเซตย่อย Q และ R จากนั้นทำการส่งมอบงานไปที่ $Proc_q$ และ $Proc_r$ เพื่อทำการประมวลผลหาค่า Minima โดย $Proc_q$ และ $Proc_r$ สามารถมีหน่วยประมวลผลที่แบ่งต่อไปได้เช่นเดียวกัน ดังนั้นในขณะที่ทำการประมวลผลซ้ำ โครงสร้างแบบ Binary tree จะถูกสร้างขึ้นจากความสัมพันธ์แบบแม่-ลูก และในกรณีที่กำหนดให้มีหน่วยประมวลผล p หน่วย ความลึกของโครงสร้างต้นไม้เป็น $\log_2 p$ โดยโหนดที่เป็นใบ (Leaf nodes) จะเป็นจุดเริ่มต้นในการค้นหา LCS แบบย้อนกลับจากฟังก์ชัน Divide-and-Conquer ที่จะทำได้พร้อมๆ กัน

2.3 ขั้นตอนวิธี PRO-MLCS และ DPRO-MLCS [10]

Pro-MLCS เป็นขั้นตอนวิธีที่ถูกปรับปรุงมาจากขั้นตอนวิธีแบบ Dominant point เดิมเช่นเดียวกับ Quick-DP ร่วมกับฮิวริสติกที่เป็นแบบ Best-first search เพื่อนำมาใช้ในการค้นหา LCS จากข้อมูลโครงสร้างต้นไม้ที่ถูกสร้างขึ้นจาก dominant points ในแต่ละเลเยอร์ (layer) จะทำการเก็บข้อมูลในคิวแบบ priority queue 1 คิวที่บรรจุโหนดในเลเยอร์เดียวกัน ในแต่ละ iteration ขั้นตอนวิธีจะทำการขยายโหนดในแต่ละคิวแบบ priority queue ในขั้นตอนสุดท้ายของแต่ละรอบ ขั้นตอนวิธีจะสร้างคำตอบในเบื้องต้นที่เป็น Approximate solution และเมื่อขั้นตอนวิธีดำเนินการจนเสร็จสิ้นจะเป็นคำตอบที่เป็น Optimal solution



ภาพที่ 2: โครงสร้างข้อมูลที่ใช้ในขั้นตอนวิธี Pro-MLCS [10]

ภาพที่ 2 แสดงโครงสร้างข้อมูลที่ใช้ในขั้นตอนวิธี Pro-MLCS ที่แต่ละเลเยอร์มีการเก็บรักษา priority queue และต้นไม้ d-index และด้วยโครงสร้างข้อมูลทั้งสองชนิดนี้ถูกใช้เก็บข้อมูล point ที่จะถูกขยายและ point ที่ถูกขยายแล้ว priority queue ของเลเยอร์ 0 จะถูกสร้างไว้ในขณะเริ่มต้นและเก็บจุด R = [0,0, ..., 0]

ขั้นตอนวิธีแบบขนานโดยทั่วไปมักมีข้อจำกัดในการประมวลผลข้อมูลเฉพาะในเลเยอร์เดียวกัน และมีการแบ่งภาระงานไปยังแต่ละหน่วยประมวลผลที่ไม่เท่ากัน ตลอดจนไม่สามารถใช้ประโยชน์จากการมีหน่วยประมวลผลเป็นจำนวนมากได้ ดังนั้นขั้นตอนวิธี DPro-MLCS [10] จะทำการกำหนดการประมวลผลเลเยอร์ให้กับแต่ละหน่วยประมวลผล เพื่อให้ได้รับภาระงาน (Workload) ที่เท่าๆ กัน และจากการพิจารณาโครงสร้างของ Dominant points ที่มีการกระจายตัวที่หลากหลาย งานวิจัยนี้จึงได้นำวิธีการกำหนดภาระงานแบบหมุนเวียน ที่เรียกว่า Cyclic assignment method ที่จะกำหนดให้เลเยอร์ที่ pk+i กับหน่วยประมวลผล i (P_i) เมื่อ p เป็นจำนวนหน่วยประมวลผลทั้งหมด

อย่างไรก็ตามขั้นตอนวิธีแบบขนานทั้งสองวิธี [9-10] มีการดำเนินการที่เร็วขึ้นสำหรับข้อมูลหลายชุด (X_1, X_2, ... , X_s) แต่จะเริ่มการประมวลผลแบบขนาน หลังจากขั้นตอนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) ในแต่ละคู่ของข้อมูล (X, Y) ซึ่งเป็นเวลาส่วนหนึ่งของการประมวลผล LCS ที่ใช้เวลานานเท่ากับ $O(n^2)$ เนื่องจากขั้นตอนเตรียมข้อมูล จะทำการประมวลผลแบบขนาน ได้ค่อนข้างยากเพราะค่าของแต่ละสมาชิก จะต้องคำนวณจากผลลัพธ์ที่ต่อเนื่องกัน

3. ขั้นตอนวิธี PLCS และ PLCS-SR แบบใหม่

งานวิจัยนี้นำเสนอการออกแบบขั้นตอนวิธี LCS แบบขนาน (Parallel LCS) ที่มีประสิทธิภาพทั้งในขั้นตอนการเตรียมข้อมูล และขั้นตอนการค้นหา LCS ซึ่งนำเสนอไว้ 2 ขั้นตอนวิธี คือ

1. ขั้นตอนวิธี PLCS (Parallel LCS (Longest Common Subsequence)) ด้วยเทคนิคไพน์ไลน์นิง (Pipelining Technique) ซึ่งจะนำเสนอในหัวข้อ 3.1 และ

2. ขั้นตอนวิธี PLCS-SR (Parallel LCS with Space Reduction) ที่ปรับปรุงมาจากขั้นตอนวิธี IDPSR-LCS [8] ที่ลดพื้นที่ในการจัดเก็บข้อมูลและเพิ่มเทคนิคไพน์ไลน์นิง ซึ่งจะนำเสนอในหัวข้อ 3.2

โดยทั้ง 2 วิธี เป็นการออกแบบขั้นตอนวิธีแบบขนานบนโมเดลของคอมพิวเตอร์แบบขนาน CREW-PRAM (Concurrent Read,Exclusive Write - Parallel Random Access Machine) และในการศึกษาเบื้องต้นจะสมมุติให้ p=n เพื่อความสะดวกในการอธิบายขั้นตอนวิธีแบบขนาน

3.1 ขั้นตอนวิธี PLCS

ขั้นตอนวิธี 3.1 เป็นการออกแบบขั้นตอนวิธี PLCS เพื่อการประมวลผลบนโมเดล CREW-PRAM ด้วยเทคนิคไพพ์ไลน์นิงระหว่างหน่วยประมวลผลในแต่ละแถวในขั้นตอนการจัดเตรียมข้อมูล (Pre-processing) โดยมีหลักการทำงาน (ขั้นตอนวิธี 3.1-1) คือ P_i จะเริ่มประมวลผลค่า c_{ij} ในแถวที่ i ใน clock j (โดยเริ่มที่แถวที่ i (i=1) หลักที่ j (j=1)) ต่อไปใน clock j+1 P_{i+1} จะเริ่มคำนวณค่า $c_{i+1,J}$ โดยที่ J=j-i+1 และใน clock ต่อไป (clock j+2) ก็จะเริ่มการทำงานของ P_{i+2} ไปเรื่อยๆ จนกระทั่งทุกหน่วยประมวลผล ทำการคำนวณค่า c_{IJ} เสร็จสิ้นภายใน 2n clocks

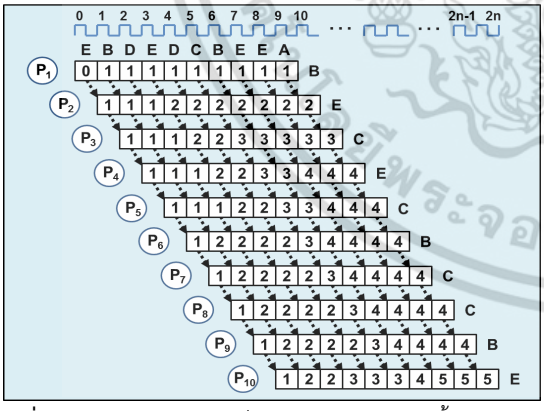
หมายเหตุ ในกรณีที่ p<n, n≠m จะใช้เทคนิคมอบภาระงาน (Workload) แบบ Cyclic เช่น ถ้า p=2 ดังนั้น P_1 จะประมวลผลแถว 1, 3, 5, ..., m-1 และ P_2 จะประมวลผลแถว 2, 4, 6, ... , m สลับกันไปจนครบทุกแถว

ขั้นตอนวิธี 3.1-1: การสร้าง Matrix C(nxn) ของ PLCS

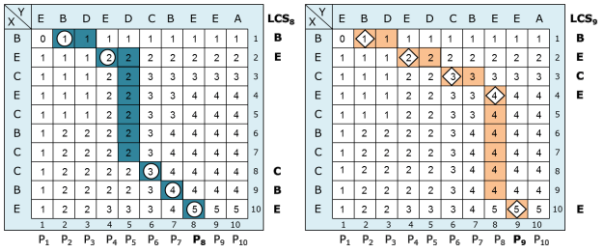
```
forall processors i=1 to n pardo
  for j = 1 to 2n do
    J = j–i+1;
    Pi: if (j≥i) and (J≤n) do
          if (xi=yJ) ci,J = ci-1,J-1+1;
          else ci,J = max(ci-1,J , ci,J-1);
        end if
  end for j
end forall
```

ขั้นตอนวิธี 3.1-2: การค้นหา LCS แบบขนานของ PLCS

```
forall processors j=1 to n pardo
  Ij=n; Jj=j; Kj=k;
  Pj: if (cIj,Jj<k) LCSj=Null;
      else do
        while (Kj>0) do
          if (xIj=yJj) zKj=xIj; Kj--; Ij--; Jj--;
          else if(cIj-1,Jj < cIj,Jj-1) Jj--; else Ij--;
        end while
end forall
```



ภาพที่ 3: การประมวลผลและเก็บค่าในเมตริกซ์ C ของขั้นตอนวิธี PLCS



ภาพที่ 4: การค้นหา LCS ของขั้นตอนวิธี PLCS

ภาพที่ 3 แสดงตัวอย่างการจัดเตรียมข้อมูล (Pre-processing) แบบขนานบนโมเดล CREW-PRAM ของข้อมูล X = BECECBCCBE และ Y = EBDEDCBEEA โดยใน clock 1 เริ่มการทำงานของ P_1 เพื่อคำนวณค่า $c_{1,1}$ และใน clock 2 P_1 คำนวณค่า $c_{1,2}$ และเริ่มการทำงานของ P_2 เพื่อคำนวณค่า $c_{2,1}$ และจะดำเนินการซ้ำในทำนองเดียวกันนี้ไปเรื่อย ๆ ใน clock ต่อไป เช่น เมื่อประมวลผลถึง clock n P_1 จะประมวลผลเสร็จ n สมาชิก ส่วน P_2 จะประมวลผลได้ n-1 สมาชิก P_3 จะประมวลผลได้ n-2 สมาชิก และ P_n จะเริ่มประมวลผลได้เพียง 1 สมาชิก ดังนั้นขั้นตอนนี้จะยังคงทำซ้ำจนกระทั่งสุดท้าย P_n คำนวณค่า $c_{n,n}$ เสร็จสิ้นภายใน 2n clocks ซึ่งด้วยเทคนิคไพพ์ไลน์นิงดังกล่าว (ทำให้สามารถคำนวณค่า c_{IJ} ในแถวถัดไปใน clock ที่ต่อเนื่องได้อย่างเหมาะสม) ภายในเวลา 2n clocks หรือ O(n) ซึ่งเร็วกว่า $O(n^2)$ ที่ประมวลผลด้วย 1 หน่วยประมวลผล

ขั้นตอนวิธี 3.1-2 เป็นการค้นหา LCS แบบขนาน ด้วยเวลา O(n) โดยที่ทุกหน่วยประมวลผล P_j (j = 1, 2, 3, ... , n) จะสามารถเริ่มพร้อมกันด้วยการตรวจสอบเงื่อนไขเพื่อหา LCS ดังแสดงผลลัพธ์ในภาพที่ 4 ซึ่งมีเพียง 3 หน่วยประมวลผล (P_8, P_9, P_{10}) ที่จะค้นหา LCS ขนาด k=5 (ซึ่งในกรณีนี้ P_{10} จะได้ LCS_{10} เหมือน LCS_9) ส่วนหน่วยประมวลผลอื่นๆ (P_j ; j=1-7) ที่ $c_{10,j} < k$ (หรือไม่พบ LCS) ดังนั้นจะให้ค่าเป็น Null

3.2 ขั้นตอนวิธี PLCS-SR

ขั้นตอนวิธี PLCS-SR ประกอบไปด้วยการประมวลผล 2 ขั้นตอนคือ 1. การจัดเตรียมข้อมูล Mlist L(n) ด้วยเทคนิคไพพ์ไลน์นิง (ขั้นตอนวิธี 3.2-1) และ 2. การค้นหา LCS จาก Mlist (ขั้นตอนวิธี 3.2-2) เมื่อ Mlist L(n) จัดเก็บข้อมูลค่าความยาวร่วมเฉพาะค่าใหม่ที่ไม่ซ้ำเพื่อลดเวลาในการคำนวณค่าความยาวร่วมด้วยขั้นตอนวิธีแบบขนานด้วยเทคนิคไพพ์ไลน์นิงและลดพื้นที่การจัดเก็บข้อมูลในอาร์เรย์ของลิสต์

ขั้นตอนวิธี 3.2-1: การสร้าง Mlist L(n) [c,x,back] ของ PLCS-SR

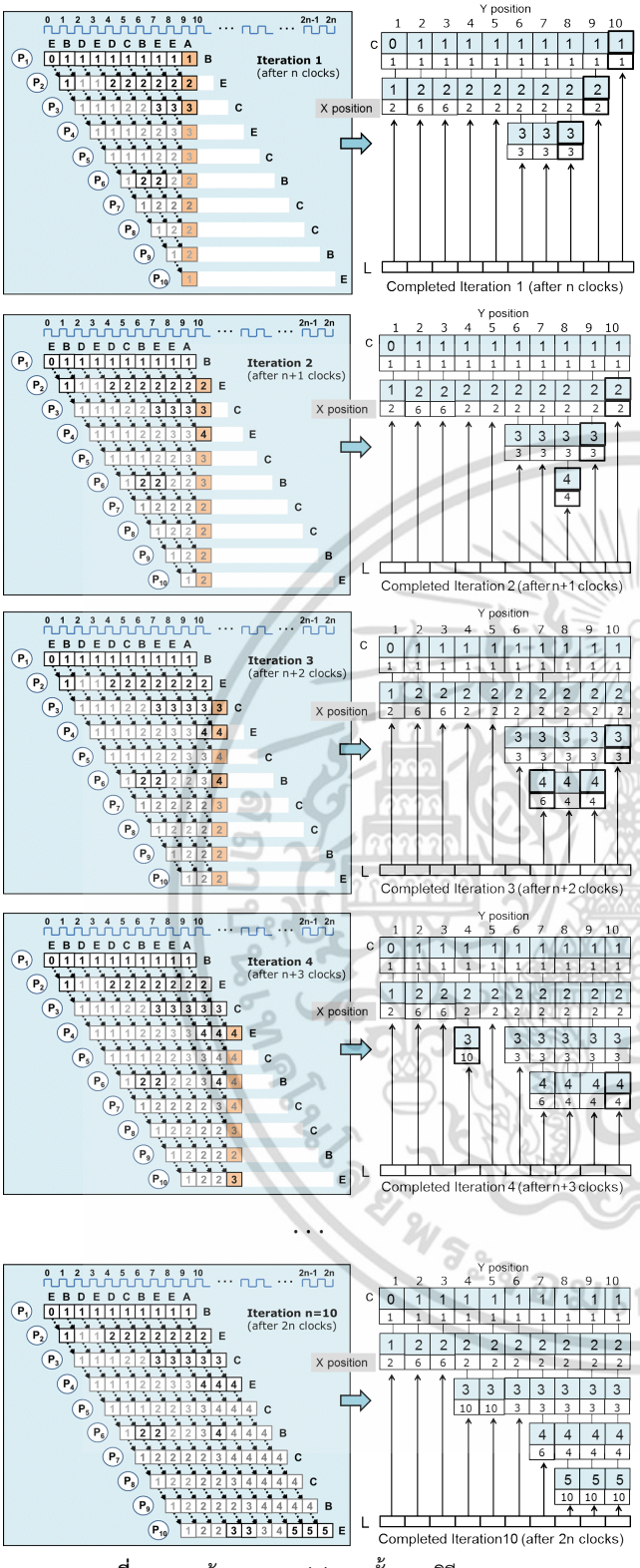
```
forall processors i=1 to n pardo
  for j=1 to 2n do
    J=j–i+1;
    Pi: if (j≥i) and (J≤n) do
          if (xi=yJ) cJ=LJ-1.c+1;
          else cJ=max(LJ.c, cJ-1);
          UpdateMlist(L(n),c(n),i,J)
        end if
  end for j
end forall
ImprovedMlist (L(n),n,Ln.c);
```

```
UpdateMlist (L(n),c(n),i,J) : O(1) by all processors
Pi: if(cJ ≠ LJ.c) do                  // add new node in LJ
      allocate new node (N); // memory allocate
      N.c=cJ; N.x=i; N.back=LJ; LJ=N;
    end if
```

```
ImprovedMlist (L(n), f, k) : O(n) by P1
 j=f; while(LJ.c=k) j=j-1;  // min j with LCS=k
 f=j; for(j=f; j>0; j--)
        Cr=LJ; Pr=LJ+1;     // Cr=Current, Pr=Previous
        while(Cr.x > Pr.x) // adjust tail & free node
          LJ=c.back; free(Cr); Cr=LJ;
        end while
      end for j
 return f+1
```

ขั้นตอนวิธี 3.2-2: การค้นหา LCS แบบขนานของ PLCS-SR

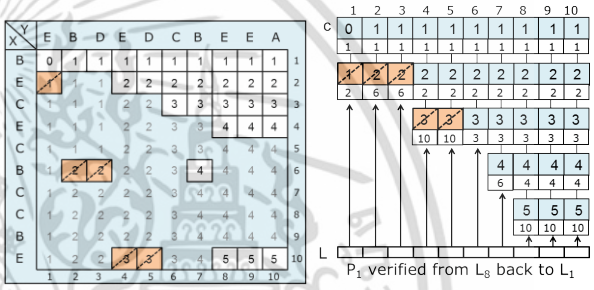
```
forall processors j=1 to n pardo
  Ij=n; Jj=j; Kj=k;
  Pj: if (cIj,Jj<k) LCSj=Null;
      else do
        while (Kj>0) do
          vj=Lj; Ij=vj.x;
          if(xIj=yJj)and(vj.c=k) zKj=xIj; Kj--; Jj--;
          else do          // xIj≠yJj
            while(vj.c > k) vj=vj.back;
            Ij=vj.x;
            if (xIj=yJj)and(vj.c=k) zKj=xIj; Kj--; Jj--;
            else Jj--;   // xIj ≠YJj goto next LJj for k
          end if-else
        end while
      end if-else
end forall
```



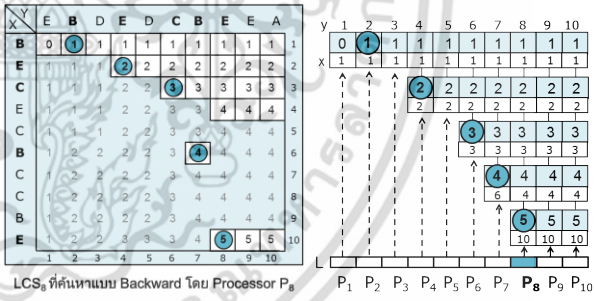
ภาพที่ 5: การสร้าง Mlist L(n) จากขั้นตอนวิธี PLCS-SR

ภาพที่ 5 แสดงการสร้าง Mlist L(n) ที่แต่ละโหนดเก็บค่า 3 ค่า [c, x, back] ในขั้นตอนการเตรียมข้อมูลของวิธี PLCS-SR จากข้อมูล X = BECECBCCBE และ Y=EBDEDCBEEA เมื่อทุกหน่วยประมวลผล P_i รับผิดชอบแถว i (=1, 2, ..., n) ของ X ซึ่งจะเริ่มเมื่อ j ≥ i (ด้วยเทคนิคไพพ์ไลน์นิงตามค่า J=j-i+1) เช่นเดียวกับวิธี PLCS (ในภาพที่ 3) แต่ในกรณีนี้ P_i จะเก็บค่า c_J ลงในลิสต์ L_J เฉพาะค่าที่ไม่ซ้ำ และเมื่อดำเนินการถึง n clocks แรก จะได้ผลลัพธ์ดังนี้ ในภาพที่ 5 (ภาพแรก) P_1 ประมวลผล Iteration 1 เสร็จ (n สมาชิก คือ c_J; J=1, 2,...,n) พร้อมด้วยการเก็บค่าที่ไม่ซ้ำลงในลิสต์ ส่วน P_2 คำนวณได้ n-1 สมาชิกและเก็บเพียงบางค่า (i=2 และ J=1,4-9) ลงในลิสต์ ในทำนองเดียวกับ P_3 – P_9 ซึ่งจะคำนวณได้ n-i+1 สมาชิก และสุดท้าย P_{10} คำนวณได้ 1 สมาชิก กล่าวโดยสรุปใน clock ที่ n=10 P_1–P_3 เก็บข้อมูลใหม่ลงในลิสต์ ส่วน P_4–P_{10} จะไม่เก็บข้อมูล เพราะเป็นค่าซ้ำที่มีอยู่ก่อนหน้า ต่อมาในภาพที่ 2 แสดงผลลัพธ์หลังจาก n+1 clocks (ที่ประมวลผลโดย P_2 – P_{10}) โดย P_2 ประมวลผล Iteration 2 เสร็จ และ P_2 – P_4 เก็บข้อมูลใหม่ลงในลิสต์ ส่วน P_5–P_{10} จะไม่เก็บข้อมูลที่เป็นค่าซ้ำ และในทำนองเดียวกันจะดำเนินการประมวลผลซ้ำๆ ไปจนเสร็จ Iterations 3–10 โดยจะแสดงผลลัพธ์ Mlist ในภาพสุดท้าย

ภาพที่ 6 แสดงผลลัพธ์ของฟังก์ชัน ImprovedMlist คือการปรับ Mlist L(n) เพื่อตัดบางโหนดที่ไม่ใช่ส่วนของ LCS ออก เช่น โหนดในตำแหน่ง (i, j) = (10,5), (10,4), (6,3), (6,2), (2,1) ทำให้สามารถลดพื้นที่ได้ถึง 72% และการเก็บค่าใน Mlist ไม่ต้องใช้พื้นที่ติดกันในหน่วยความจำ



ภาพที่ 6: ตัวอย่างผลลัพธ์การปรับ Mlist ด้วยฟังก์ชัน ImprovedMlist



ภาพที่ 7: การค้นหา LCS หลายค่า แบบ PLCS-SR จาก Mlist

ในขั้นตอนการค้นหา LCS (ขั้นตอนวิธี 3.2-2) จาก Mlist ของวิธี PLCS-SR โดยมี Input คือ Mlist L(n), X, Y, ความยาว LCS = k โดยทุกหน่วยประมวลผล (P_j ; j=1, 2, ..., 10) สามารถประมวลผลได้พร้อมๆ

กันด้วยเวลา O(n) โดยที่แต่ละหน่วยประมวลผลจะรับผิดชอบหา LCS 1 ชุด โดยในแต่ละ P_j จะกำหนดให้ v_j แทนโหนดใดๆ ที่เริ่มจาก tail ของ list L_j และ P_j จะหา LCS ได้ก็ต่อเมื่อค่า $v_j.c$ ของโหนดเริ่มต้นมีค่า=k ส่วนโหนดเริ่มต้นที่มีค่า $v_j.c < k$ จะกำหนดให้ LCS=Null (ไม่เป็นค่า LCS)

ภาพที่ 7 แสดงการค้นหา LCS แบบขนาน โดยที่ทุกหน่วยประมวลผล P_j (j = 1, 2, 3, ... , 10) จะต้องตรวจสอบเงื่อนไขก่อนการหา LCS พร้อมๆ กัน ซึ่งในตัวอย่างนี้จะมีเพียง 3 หน่วยประมวลผล (P_8, P_9, P_{10}) ที่จะทำการค้นหา LCS ขนาด k=5 ส่วนหน่วยประมวลผลอื่นๆ (P_j ; j = 1, 2, ... ,7) ที่มีค่าเริ่มต้นในโหนด v_j ของลิสต์ L_j คือ $v_j.c < k$ (แสดงว่าไม่เป็น LCS ซึ่งจะกำหนดให้ค่า LCS เป็น Null) สำหรับหน่วยประมวลผล P_8 – P_{10} ที่มีค่าเริ่มต้นของโหนด $v_j.c=k$ ในลิสต์ L_8 – L_{10} จะทำการค้นหา LCS พร้อมๆ กัน ภาพที่ 7(ก) แสดงการค้นหา LCS_8 โดย P_8 ที่เริ่มจากโหนด v_j ของ L_8 ที่มีค่า $v_j.c=k$ ในตำแหน่ง (i, j)=(10, 8) ซึ่งจะได้ LCS_8 = BECBE และภาพที่ 7(ข) แสดงการค้นหา LCS_9 โดย P_9 ที่เริ่มจากโหนด v_j ของ L_9 ที่มีค่า $v_j.c=k$ ในตำแหน่ง (i, j)=(10, 9) ซึ่งจะได้ LCS = BECEE ส่วน P_{10} จะได้ LCS_{10} เหมือนกับ LCS_9

4. ประสิทธิภาพของขั้นตอนวิธี PLCS และ PLCS-SR

การเปรียบเทียบประสิทธิภาพของขั้นตอนวิธี PLCS และ PLCS-SR (ที่นำเสนอในหัวข้อ 3) ในที่นี้จะเป็นการแสดงประสิทธิภาพด้านเวลาเป็นหลัก คือความสามารถในการประมวลผลที่เร็วขึ้นด้วย n หน่วยประมวลผล ซึ่งจะเปรียบเทียบกับขั้นตอนวิธีแบบเดิม ที่ใช้หนึ่งหน่วยประมวลผล ดังแสดงเป็นค่าความซับซ้อน Time และ Space ในตาราง 3 (PLCS ที่มีข้อจำกัดเรื่องพื้นที่จัดเก็บข้อมูลแบบเมตริกซ์ nxn) และตาราง 4 (PLCS-SR ที่ลดพื้นที่จัดเก็บข้อมูลในส่วน Pre-processing) โดยความซับซ้อนด้านเวลาของการประมวลผลแบบขนานของทั้ง 2 วิธี เท่ากับ O(n) เทียบกับเดิม $O(n^2)$ ที่ประมวลผลด้วย 1 หน่วยประมวล ดังนั้น Speedup ($S_P=T_1/T_P$) ของทั้ง 2 วิธีมีค่า ≈n แต่วิธี PLCS เก็บข้อมูลในอาร์เรย์ 2 มิติที่เป็นแบบ shared matrix nxn ที่ไม่สามารถประมวลผลเมื่อ n มีค่ามาก ส่วนวิธี PLCS-SR เก็บข้อมูลใน shared Mlist ที่สามารถรองรับ n ทุกค่า

ตาราง 3: ความซับซ้อน Time และ Space ของขั้นตอนวิธี PLCS

ขั้นตอนวิธี	Sequential 1 PE		Parallel n PEs	
	Time (T_1)	Space	Time (T_P)	Shared Space
Pre-processing	$O(n^2)$	$O(n^2)$	O(n)	$O(n^2)$
MLCS searching	O(cn)	O(n)	O(n)	O(n)
Total	**$O(n^2)$**	**$O(n^2)$**	**O(n)**	**$O(n^2)$**

ตาราง 4: ความซับซ้อน Time และ Space ของขั้นตอนวิธี PLCS-SR

ขั้นตอนวิธี	Sequential 1 PE		Parallel n PEs	
	Time (T_1)	Space	Time (T_P)	Shared Space
Pre-processing	$O(n^2)$	O(kn)	O(n)	O(kn)
MLCS searching	O(cn)	O(n)	O(n)	O(n)
Total	**$O(n^2)$**	**O(kn)**	**O(n)**	**O(kn)**

5. สรุปผล

งานวิจัยนี้ นำเสนอขั้นตอนวิธีการหาลำดับร่วมเหมือนที่ยาวที่สุด LCS แบบขนานบนโมเดล Shared memory CREW-PRAM ของข้อมูลสายอักษร 2 ชุด (X, Y) ที่มีประสิทธิภาพด้านความเร็วเท่ากับ O(n) ทั้งในส่วนการจัดเตรียมข้อมูลก่อนการประมวลผล (Pre-processing) และการค้นหา LCS ซึ่งเสนอไว้ 2 ขั้นตอนวิธี โดยวิธีแรก PLCS พัฒนามาจากขั้นตอนวิธี DP-LCS ดั้งเดิม ด้วยเทคนิคไพน์ไลน์นิงแบบขนาน และวิธีที่ 2 PLCS-SR ที่พัฒนามาจากขั้นตอนวิธี IDPSR-LCS ที่ลดพื้นที่จัดเก็บข้อมูล [8] ร่วมกับการประมวลผลแบบขนานด้วยเทคนิคไพน์ไลน์นิง ซึ่งในการเปรียบเทียบประสิทธิภาพของทั้งสองวิธีพบว่า ค่า Speedup ของทั้งสองวิธีมีค่า ≈n แต่วิธี PLCS มีข้อจำกัดที่ค่า Shared matrix ขนาด nxn ส่วนวิธี PLCS-SR เก็บข้อมูลใน Shared Mlist ที่รองรับกรณีที่ n มีค่ามากได้สอดคล้องกับยุคปัจจุบันที่ข้อมูลมีการเพิ่มขึ้นอย่างมหาศาล (Big Data) และต้องการความรวดเร็วในการประมวลผล

ส่วนงานวิจัยที่จะศึกษาต่อไป จะเป็นการออกแบบขั้นตอนวิธี MLCS แบบขนาน สำหรับข้อมูลหลายชุด (X_1, X_2, ... , X_s) ซึ่งเป็นปัญหาที่น่าสนใจ เพราะเป็นปัญหา NP-hard ชนิดหนึ่ง

เอกสารอ้างอิง

[1] K. Ning, H. Kee and H. W. Leong, “Finding Patterns in Biological Sequences by Longest Common Subsequences and Shortest Common Supersequence,” Sixth IEEE Symposium on BioInformatics and BioEngineering, pp. 53-60, 2006.

[2] R.A.C. Campos and F.J.Z Martinez, “Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem,” 9th International Conference on Electrical Engineering, Computing Science and Automatic Control, pp. 1 – 4, 2012.

[3] D. Abraham and N.S. Raj, “Approximate string matching algorithm for phishing detection,” 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2285 – 2290, 2014.

[4] D. Frolova, H. Stern and Berman, S, "Most Probable Longest Common Subsequence for Recognition of Gesture Character Input," IEEE Trans. on Cybernetics, vol. 43, no.3, pp. 871 - 880, 2013.

[5] J. Liaw and et al., “Recognition of the Ambulance Siren Sound in Taiwan by the Longest Common Subsequence,” 2013 IEEE international conference on Systems, Man, and Cybernetics, pp.3824-3828, 2013.

[6] X. Xiang, D. Zhang, and J. Qin. “An New Algorithm for the Longest Common Subsequence Problem,” 2007 International Conference on Computational Intelligence and Security Workshops, pp. 112 – 115, 2007.

[7] J. Yang, Y. Xu, Y. Shang, and G. Chen, “A Space-Bounded Anytime Algorithm for the Multiple Longest Common Subsequence Problem,” IEEE Trans. on Knowledge and Data Engineering, vol. 26, no. 11, pp. 2599 - 2609, 2014.

[8] S. Dangmanee and J. Werapun, “Longest Common Subsequence by New Dynamic Programming with Space Reduction,” The 12th National Conference on Computing and Information Technology, pp. 286 – 291, 2016.

[9] Q. Wang, D. Korkin, and Y. Shang, “A Fast Multiple Longest Common Subsequence (MLCS) Algorithm,” IEEE Transactions on Knowledge and Data Engineering, vol.23, no.3, pp. 321 – 334, 2011.

[10] J. Yang and et al., “A New Progressive Algorithm for a Multiple Longest Common Subsequences Problem and Its Efficient Parallelization,” IEEE Transactions on Parallel and Distributed Systems, vol.24, no.5, pp. 862 – 870, 2013.

ประวัติผู้เขียน

ชื่อ	นายศักดิ์พันธุ์ แดงมณี
วัน เดือน ปีเกิด	21 ธันวาคม 2527
ที่อยู่ปัจจุบัน	66 หมู่ 6 ต.แม่กา อ.เมือง จ.พะเยา 56000
ประวัติการศึกษา	
2550	วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ เกรดเฉลี่ย 3.23 มหาวิทยาลัยนเรศวร