การหาลำดับร่วมเหมือนที่ยาวที่สุดด้วยขั้นตอนวิธีไดนามิคโปรแกรมมิงแบบลดพื้นที่

Longest Common Subsequence by New Dynamic Programming with Space Reduction

ศักดิ์พันธุ์ แดงมณี (Sakpan Dangmanee)* และ จีรพร วีระพันธุ์ (Jeeraporn Werapun)*

บทคัดย่อ

บทความวิจัยนี้ได้นำเสนอขั้นตอนวิธีไดนามิคโปรแกรมมิง (Dynamic Programming (DP)) แบบใหม่ (IDPSR) ที่ลดพื้นที่ในการประมวลผลลำดับร่วมเหมือนที่ยาวที่สุด (Longest Common Subsequence (LCS)) ของข้อมูลสายอักษร 2 ชุด ปัญหาการค้นหา LCS ดังกล่าว ถูกนำมาประยุกต์ใช้ในงานหลากหลายประเภท อาทิ DNA matching และการตรวจจับ Web Phishing โดยขั้นตอนวิธี DP เดิม ใช้พื้นที่ในหน่วยความจำแบบเมตริกซ์ขนาด *m*x*n* ซึ่งเป็นอาเรย์ 2 มิติ ที่ต้องใช้พื้นที่ติดกันในหน่วยความจำ ซึ่งเป็นข้อจำกัดในการประมวลผล *m*, *n* ขนาดใหญ่ๆ งานวิจัยที่ผ่านมา จึงเน้นขั้นตอนวิธีที่ใช้พื้นที่จัดเก็บเฉพาะข้อมูลที่สำคัญๆ แต่อาจประมวลผลนานขึ้น ดังนั้นเพื่อแก้ปัญหาดังกล่าว ประกอบกับข้อมูลที่มีอยู่ในยุคปัจจุบันมีการขยายตัวขึ้นอย่างรวดเร็ว ในงานวิจัยนี้ผู้วิจัยจึงได้เสนอขั้นตอนวิธี 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.

Keywords: 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 ใช้พื้นที่จัดเก็บ

* คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

* Faculty of Science, King Mongkut's Institute of Technology Ladkrabang.

ข้อมูลในหน่วยความจำค่อนข้างมาก ซึ่งไม่สอดคล้องกับข้อมูลที่มีอยู่ในยุคปัจจุบันที่มีการขยายตัวขึ้นอย่างรวดเร็ว เพื่อรองรับการใช้งานแอพพลิเคชันที่เป็นเครือข่ายสังคม (Social Network) หรือที่เรียกว่ายุคข้อมูลขนาดใหญ่ (Big Data) ทำให้ขั้นตอนวิธี DP-LCS เดิม ที่มีประสิทธิภาพ อาจไม่สามารถนำมาใช้ในการประมวลผลข้อมูลที่มีขนาดใหญ่ๆ ได้ การปรับปรุงให้ขั้นตอนวิธีใช้พื้นที่ในการประมวลผลที่ลดลง จึงเป็นสิ่งจำเป็นในการนำขั้นตอนวิธี DP ดังกล่าว มาใช้ได้อย่างมีประสิทธิภาพในยุคปัจจุบัน

ในทางทฤษฎี ขั้นตอนวิธี DP-LCS เดิมที่มีประสิทธิภาพจะถูกพัฒนาเป็น 2 ส่วนคือ 1. การจัดเตรียมข้อมูลก่อนประมวลผล (Pre-processing) โดยสร้างตารางเก็บข้อมูลขนาด *mxn* เพื่อระบุตำแหน่งที่มีตัวอักษรที่เหมือนกัน และ 2. นำข้อมูลในขั้นแรกไปใช้ประมวลผลเพื่อหาลำดับร่วมเหมือนที่ยาวที่สุด (LCS) ซึ่งใช้เวลาประมวลผล O(mn) และพื้นที่เก็บข้อมูล O(mn) ในปีค.ศ. 2007 Xiang และคณะ [6] ได้เสนอขั้นตอนวิธี LCS แบบใหม่ที่ลดพื้นที่ด้วยการเก็บข้อมูลจุดที่เป็น Pre-sequence เท่านั้น ในส่วนของการจัดเตรียมข้อมูล ด้วยเวลา O(mn) เท่าเดิม แต่ลดพื้นที่ลงเหลือ O(Ln) ต่อมาในปี ค.ศ. 2014 Yang และคณะ [7] เสนอขั้นตอนวิธี SA-MLCS ที่ลดพื้นที่ ด้วยการจัดเก็บข้อมูลที่เป็น Dominant Point (จุดที่เป็นตัวอักษรร่วมของข้อมูลลำดับ) ก่อนการประมวลผล 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 งานวิจัยที่เกี่ยวข้อง

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

2.1 วิธีการค้นหา LCS แบบไดนามิคโปรแกรมมิง

ในการหาสายอักษรร่วมที่ยาวที่สุด (LCS) $Z = <z_1, z_2, \ldots, z_k>$ (ขนาด k ตัวอักษร) ของสายอักษร Xm และ Yn 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 Cmxn ที่เก็บค่าของความยาวร่วมของสายอักษร (LCS) เมื่อ $c_{i,j}$ แทนค่าความยาวของ LCS จากสายอักษร Xm และ Yn 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 $x_i \neq y_i$

ขั้นตอนการสร้างเมตริกซ์ Cmxn 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 C10x10 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 Xiang

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

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 Xiang

ขั้นตอนวิธี New LCS (I, J) ของ Xiang [6]

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

กำหนดให้ P = {(i, j) : ii=jj} แล้ว (i, j) เป็นคู่ลำดับตำแหน่งของตัวอักษรที่เหมือนกัน (Match pair) จากสายอักษร I และ J ซึ่งอาจเป็น p_x ที่สอดคล้องกับตัวอักษรที่เกิดขึ้นทั้งหมด โดยแสดงในรูปแบบของกลุ่มคู่ลำดับ $P = \{p_1, p_2, \ldots, p_l\}$ โดยที่แต่ละ p_k เป็นคู่อันดับของ (i_k, j_k) การสร้างเซ็ตของคู่ลำดับ จะกำหนดให้คู่ลำดับของอักษรที่เหมือนกันจำนวนสองคู่ px, p_y, ถ้า $i_x < i_y, jx < j_y$ 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

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): ii=jj} 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, … , l} จากนั้น LCS คือเซตคู่ลำดับที่มีความยาวที่สุด LCS(I, J) = max{P_i}

2.3 วิธีการค้นหา MLCS ของ Yang

ขั้นตอนวิธี SA-MLCS ของ 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) ในเส้นทางของกราฟที่

ถูกนำมาสร้างขึ้น โดยโหนดของกราฟที่ใช้ในการค้นหาจะถูกจัดกลุ่มเป็นระดับชั้น (Layers) ด้วยระยะห่างจากโหนดต้นทาง สำหรับแต่ละระดับชั้น ในขั้นตอนวิธี SA-MLCS ดังกล่าว จะมีการใช้โครงสร้างข้อมูลแบบคิว 3 คิว (New, Open, และ Closed queues) ซึ่งมีนิยามการสร้างดังนี้

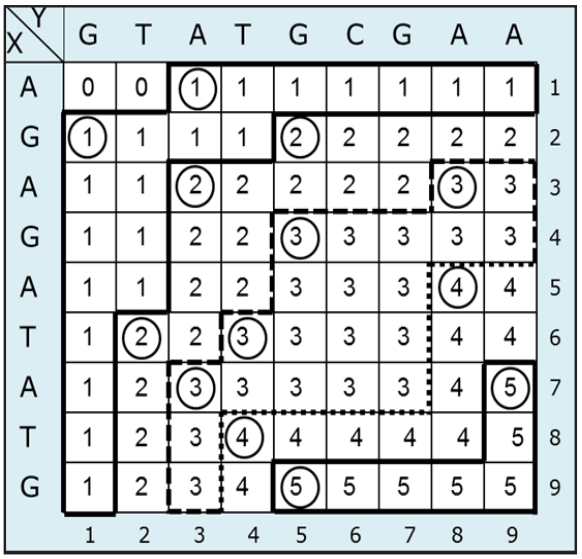
New queue คือคิวที่จัดเก็บโหนดที่สามารถสร้างโหนดลูกได้

Open queue คือคิวของโหนดที่ยังไม่ได้สร้างลูกทั้งหมด

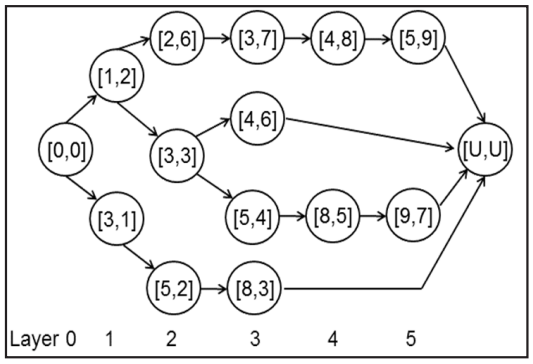
Closed queue คือคิวของโหนดที่ทำการสร้างลูกแล้วทั้งหมด

ขั้นตอนวิธี SA-MLCS ของ 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 Newcur_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.



(ก)



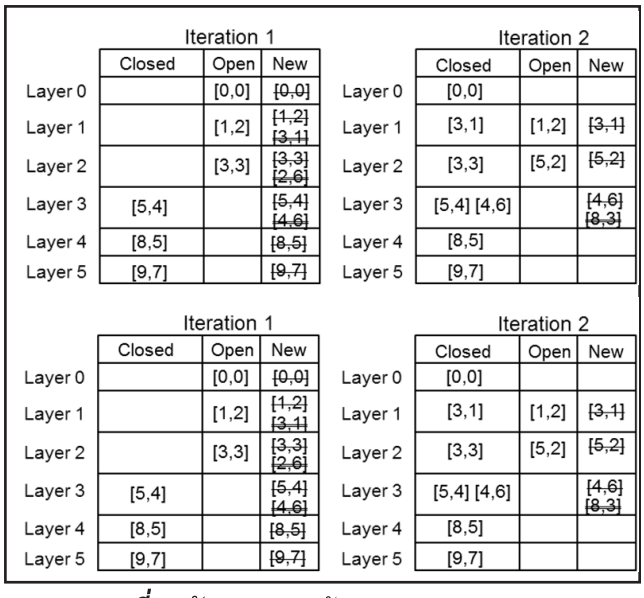
(ข)

ภาพที่ 3 *ตัวอย่างการเก็บข้อมูลเพื่อค้นหา MLCS ของ Yang*

ตารางที่ 1 *โครงสร้างข้อมูลและการใช้พื้นที่เก็บข้อมูลเพื่อค้นหา LCS*

ขั้นตอนวิธี	Pre-processing	LCS searching
Efficient DP-LCS	Matrix Cmxn (2D-Array)	-
NewLCS [6] 2007	1D-Array P (ขนาด Ln)	P1, P2, .., Pl; |Pi| ≤ n
SA-MLCS [7] 2014	Graph (cn nodes)	3 Queues; |Q| ≤ n
Improved DP with Space Reduction (IDPSR-LCS) นำเสนอในหัวข้อ 3	n-MultipleList L; |Li| ≤ k (k = length of LCS)	-
	Space Reduction cn < kn (c=constant)	

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))



ภาพที่ 4 *ตัวอย่างการค้นหา MLCS ของ Yang*

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) ซึ่งเป็นข้อจำกัดในทางปฏิบัติเมื่อ m, n มีค่ามากๆ เพราะตัวแปรอาเรย์ขนาด mxn ต้องใช้พื้นที่จัดเก็บค่าติดกันในหน่วยความจำในขณะประมวลผลโปรแกรม ซึ่งปกติจะจองเนื้อที่อาเรย์ได้ไม่เกิน 800x800 (m, n < 800)

ดังนั้นงานวิจัยนี้ จึงนำเสนอขั้นตอนวิธี IDPSR ที่มีการใช้หน่วยความจำแบบ Dynamic เพื่อแก้ปัญหาดังกล่าว ด้วยโครงสร้างข้อมูลแบบอาเรย์ของลิสต์ชนิดพิเศษคือ Mlist Li (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 แต่ในที่นี้จะจัดเก็บลงใน Mlist เฉพาะค่าใหม่ที่ไม่ซ้ำ (เพื่อลดพื้นที่และเพื่อค้นหา LCS และ MLCS ได้เร็วขึ้น) พร้อมตำแหน่งของ X และตัวชี้ back

ขั้นตอนการสร้าง 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 (L_n, c_n, x);
end for x
return ImprovedMlist (L_n, 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)

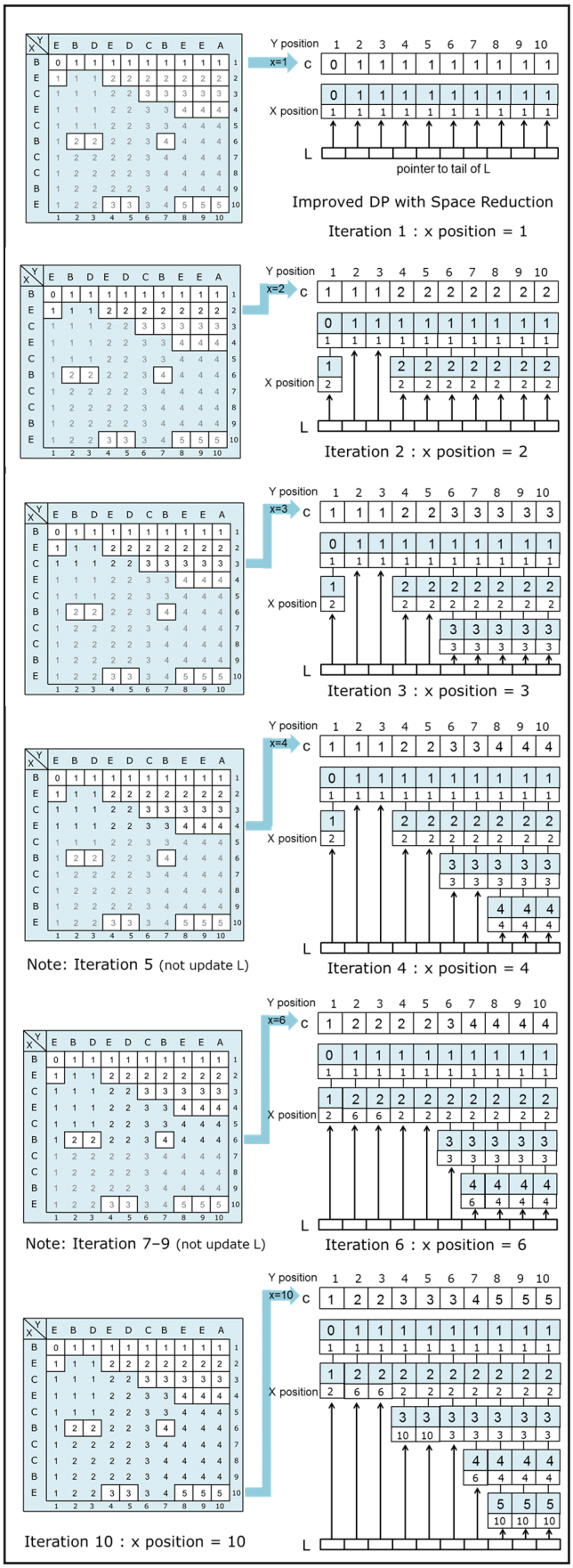
```
y=f; while(L[y]->c = k) y=y-1;  // min y with LCS=k
f=y; for(y=f; y>0; y--)
       c=L[y]; p=L[y+1]; // c=current, p=previous
       while(c->x > p->x) // adjust tail & free node
         L[y]=c->back; free(c); c=L[y];
       end while
     end for y;
return f+1;
```

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]

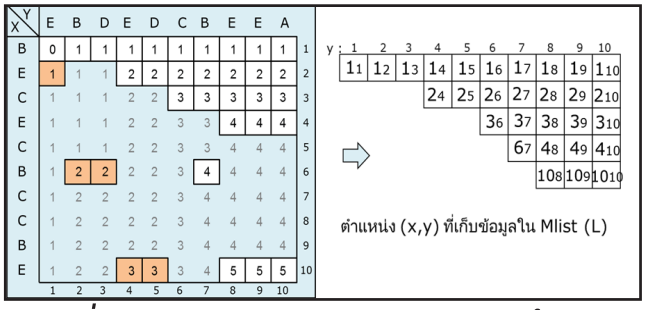
= max {c[y-1], L[y]} ถ้า X[x] ≠ Y[y]

ตัวอย่างเช่น ภาพที่ 5 แสดงการสร้าง Mlist L ของข้อมูล



ภาพที่ 5 *การเก็บข้อมูลเพื่อการค้นหา LCS ในแบบ IDPSR-LCS*

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 (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) 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

เป็นการค้นหา LCS จาก Mlist (ในขั้นตอนวิธี 3.1) โดยมี Input คือ list Ln, strings Xm, Yn, ความยาว LCS = k พร้อมตำแหน่ง x, y และ Output คือ LCS Zk

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 Ln ใน Iteration 1 (y = 8, k = 5) โหนดแรก v = L[y] = L[8] จะได้ x = v->x = 10 และ Z[5]='E'

ขั้นตอนการค้นหา LCS จาก Mlist ของวิธี IDPSR-LCS

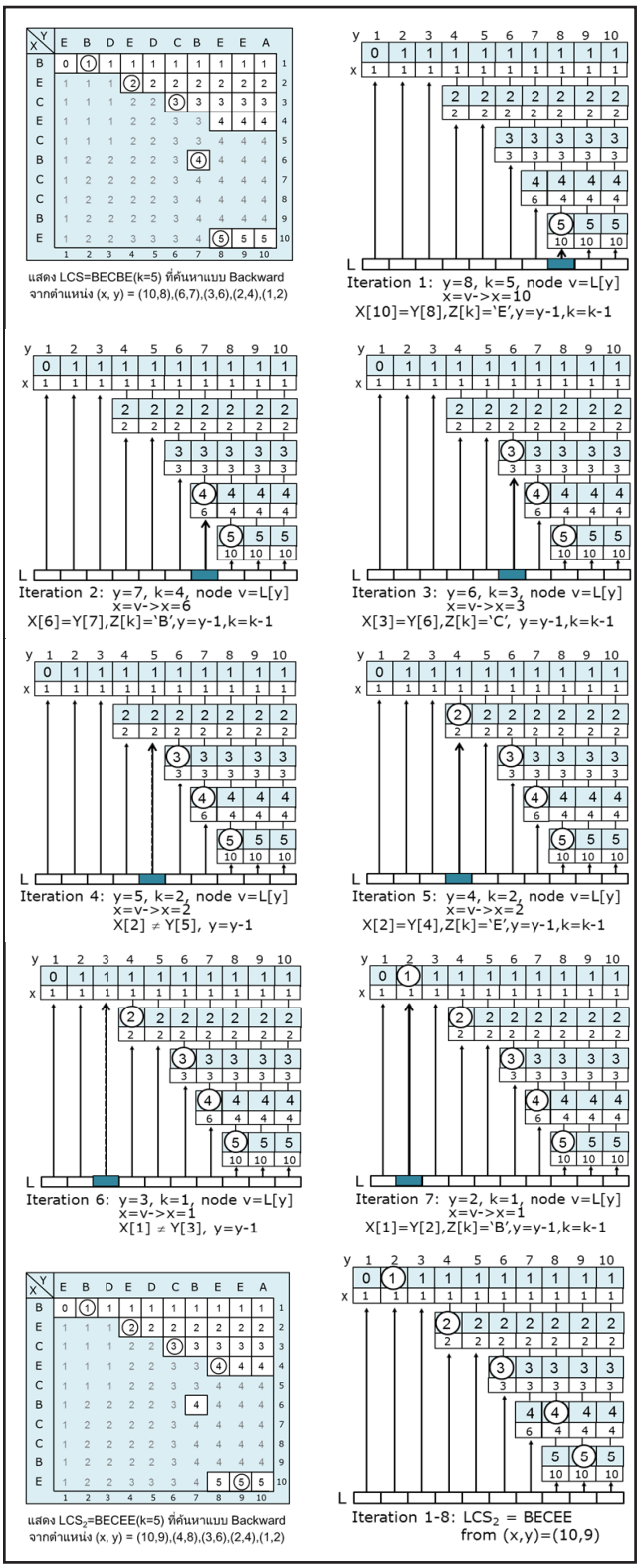
```
BackLCSfromMlist (L_n, X_m, Y_n, Z_k, x, y, k)
 while (k>0) do
   v=L[y]; x=v->x;
   if((X[x]=Y[y])&(v->c=k)) Z[k]=X[x];k=k-1;y=y-1;
   else // either X_x≠Y(of LCS1) or find X_x=Y_y(of LCS_other)
     while(v->c > k) v=v->back; // for other LCS
     x=v->x;
     if((X[x]=Y[y])&(v->c=k))Z[k]=X[x];k=k-1;y=y-1;
     else y=y-1;   // X_x≠Y_y goto next L[y] for k
   end if-else;
 end while;
end.
```

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 วิธีต่าง ๆ 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 Lj (ขนาด n) ทุกค่าของ Y คือ j = 1, 2, 3, … , n เพื่อเก็บเฉพาะค่าที่แตกต่าง (หรือเมื่อพบค่าร่วมใหม่) เพื่อเป็น x-pointer jumping ในค่าของ X (ในขั้นตอนค้นหา LCS) นอกจากนั้นในขั้นนี้ จะเพิ่มฟังก์ชันที่ปรับลิสต์ (ImprovedMlist) ในส่วน ท้าย ๆ 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



ภาพที่ 7 *การค้นหา LCS1 และ LCS2 แบบ IDPSR จาก Mlist*

และ O(m+n) กรณีช้าที่สุด ส่วนวิธี IDPSR จะค้นหา LCS ด้วยความซับซ้อน O(n) เพราะ LCS ชุดแรก จะพบได้โดยตรงจากปลายลิสต์ ส่วน MLCS อื่นๆ จะใช้ค่าต่อไปของแต่ละลิสต์ ที่จะไปถึงได้ด้วย X-pointer jumping ที่จัดเตรียมไว้ใน

ลิสต์ ซึ่งแตกต่างจาก DP เดิม ที่จะต้องเลื่อนไปทีละค่าของอาเรย์ X และ Y

ตารางที่ 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)

ตารางที่ 3 *การเปรียบเทียบข้อดีและข้อจำกัดของขั้นตอนวิธีต่าง ๆ*

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 ทุกคำและ 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 ทุกคำ (MLCS) และ CS อื่นๆ ที่เกี่ยวของ
IDPSR-LCS ที่เสนอในงานวิจัยนี้	
ข้อดี	มีข้อดีเหมือน DP คือหา LCS ได้โดยตรง แต่เร็วกว่า O(n) มีข้อดีเหมือน [6, 7] ที่สามารถลดพื้นที่เพราะเก็บข้อมูลเป็นอาเรย์ 1 มิติของลิสต์แทน (โดยเก็บเฉพาะคา c_{ij} ที่แตกต่าง)
ข้อจำกัด	ยังคงใช้เวลาเตรียมข้อมูลในส่วน Pre-processing เป็น O(mn) เหมือน DP และวิธีอื่นๆ [6, 7]

ตารางที่ 4 *ผลการทดลองเปรียบเทียบการลดพื้นที่จัดเก็บข้อมูล*

m = n	k	Matrix C_{mxn}	Mlist	%พื้นทีเก็บ	%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 ต่าง ๆ โดยวิธี IDPSR-LCS สามารถรักษาข้อดีของวิธี DP เดิม (ในช่วงการค้นหา LCS ที่เร็ว) และมีข้อดีเหมือนวิธีอื่น ๆ (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 ที่มีประสิทธิภาพนี้ในการตรวจจับ Web Phishing

6. 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.” *International Conference on Advances in Computing, Communications and Informatics (ICACCI)* 2014, pp. 2285 – 2290, 2014.

[4] D. Frolova, H. Stern and S. Berman. “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.” *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.” *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 Transactions on Knowledge and Data Engineering*, Vol. 26, No. 11, pp. 2599 - 2609, 2014.