JIT Mahasarakham University, Thailand
http://jit.it.msu.ac.th

Journal of Applied Informatics and Technology, 7(2), 389-405.

Research Article

ขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณโดยอาศัยรายการผกผันแบบเดี่ยว

Approximate String Matching Algorithm using Single Inverted Lists

สุนทรีย์ ธรรมสุวรรณ[1], นวลปราง แสงอุไร[1,*], เชาวลิต ขันคำ[2]
Soontaree Thumsuwan[1], Nuanprang Sangurai[1,*], Chouvalit Khancome[2]

[1] สาขาวิชาเทคโนโลยีสารสนเทศ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏราชนครินทร์ ฉะเชิงเทรา 24000 ประเทศไทย
[2] ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยรามคำแหง กรุงเทพมหานคร 10240 ประเทศไทย
[1] Major in Information Technology Department, Faculty of Science and Technology, Rajabhat Rajanagarindra University, Chachoengsao 24000, Thailand
[2] Computer Science Department, Faculty of Science, Ramkhamhaeng University, Bangkok 10240, Thailand
* Corresponding Author: Nuanprang Sangurai, nuanprang.san@csit.rru.ac.th

Received:
29 November 2023
Revised:
31 January 2024
Accepted:
2 May 2024

คำสำคัญ:

การเปรียบคู่สายอักขระแบบประมาณ, รายการผกผัน, ดัชนีผกผัน, การเปรียบคู่แบบ, การเปรียบคู่สายอักขระแบบอนุญาตให้มีข้อผิดพลาด

Keywords:

String Matching Algorithm, Multiple Character Inverted Lists, Inverted Index, Pattern Matching, Exact String Matching

บทคัดย่อ: การเปรียบคู่สายอักขระแบบประมาณเป็นหลักการสำคัญในการค้นหาข้อมูลที่อนุญาตให้มีข้อผิดพลาดในการสะกดคำหรือการพิมพ์ผิดได้ ซึ่งถูกนำไปประยุกต์ใช้อย่างแพร่หลายในงานด้านฐานข้อมูล ระบบสืบค้นข้อมูล และแอปพลิเคชันหรือบริการออนไลน์ต่างๆ อย่างต่อเนื่อง เพื่อเพิ่มประสิทธิภาพในการค้นหาให้รวดเร็วและตรงตามวัตถุประสงค์มากยิ่งขึ้น การพัฒนาขั้นตอนวิธีใหม่จึงเป็นความท้าทายที่สำคัญในงานวิจัยด้านวิทยาการคอมพิวเตอร์ บทความวิจัยนี้นำเสนอโครงสร้างข้อมูลชนิดใหม่สำหรับการค้นหาแบบประมาณเรียกว่า รายการผกผันแบบเดี่ยว ซึ่งออกแบบมาเพื่อรองรับการกำหนดระดับความผิดพลาดที่ยอมรับได้ในการค้นหา พร้อมทั้งพัฒนาขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณที่อิงกับโครงสร้างข้อมูลดังกล่าว ผลการวิเคราะห์เชิงทฤษฎีแสดงให้เห็นว่า โครงสร้างข้อมูลที่นำเสนอสามารถสร้างได้ด้วยความซับซ้อนเชิงเวลาในระดับเดียวกับความยาวของสายอักขระต้นแบบ และใช้พื้นที่จัดเก็บข้อมูลเท่ากับผลรวมของความยาวสายอักขระและจำนวนอักขระที่ปรากฏจริง ขั้นตอนวิธีที่พัฒนาขึ้นสามารถดำเนินการค้นหาได้ด้วยความซับซ้อนเชิงเวลาเท่ากับผลคูณระหว่างความยาวของข้อความกับความยาวของสายอักขระต้นแบบ ทั้งนี้สามารถกำหนดระดับความผิดพลาดที่ยอมรับได้ในการค้นหา ผลการทดลองจากการพัฒนาโปรแกรมคอมพิวเตอร์เพื่อนำไปเปรียบเทียบกับขั้นตอนวิธีที่มีชื่อเสียงในปัจจุบัน พบว่าโครงสร้างข้อมูลที่พัฒนาขึ้นใช้หน่วยความจำในการประมวลผลน้อยที่สุด และขั้นตอนวิธีที่นำเสนอสามารถดำเนินการค้นหาแบบประมาณได้อย่างมีประสิทธิภาพ รวดเร็วใกล้เคียงกับขั้นตอนวิธีที่เร็วที่สุด และยังทำงานแบบเชิงเส้นอีกด้วย

Abstract: Approximate string matching is a fundamental technique in data retrieval that allows for typo errors or misspellings. It is widely applied in databases, search engines, and various applications or online services. To enhance the speed and accuracy of data retrieval, the development of new algorithms remains a significant challenge in computer science research. This paper introduces a novel data structure for approximate search, called the Single Inverted List, which supports a configurable level of error tolerance. Based on this structure, a new approximate string matching algorithm is developed. Theoretical analysis shows that the proposed structure can be constructed with time complexity proportional to the length of the pattern string and requires storage space equal to the sum of the pattern length and the number of distinct characters. The proposed algorithm achieves search performance with time complexity proportional to the product of the text length and the pattern length, while also supporting error-tolerant matching. Experimental results demonstrate that the proposed structure consumes the least memory compared to well-known existing algorithms, and the developed algorithm performs approximate searches efficiently, nearly as fast as the fastest existing methods, while maintaining linear-time performance.

1. บทนำ

หลักการการเปรียบคู่สายอักขระแบบประมาณ (Approximate String Matching) หรือการเปรียบคู่สายอักขระแบบอนุญาตให้มีข้อผิดพลาด (String Matching with Allowed Errors) เป็นกระบวนการหรือการเทคนิคในการค้นหาสตริง (String) ที่ตรงกับรูปแบบ (Pattern) โดยประมาณ ซึ่งขั้นตอนวิธีพยายามค้นหาข้อความที่คล้ายคลึงกับข้อความที่กำหนดไว้ โดยไม่จำเป็นต้องตรงกันทั้งหมด เพียงตรงกับเงื่อนไขการอนุญาตให้ผิดพลาดได้ (Error Rate หรือ Distance) หลักการนี้มีประโยชน์มากโดยเฉพาะเมื่อมีการพิมพ์ผิด สะกดคำผิด หรือหาความคล้ายคลึงกันในทางตรรกะที่ไม่จำเป็นต้องตรงกันทุกอักขระ ถูกนำไปใช้ในหลายวงการคอมพิวเตอร์ เช่น การค้นหาข้อมูลในฐานข้อมูล การแก้ไขข้อความที่ผิดพลาดในเอกสาร การตรวจสอบการสะกดคำ และการปรับปรุงประสิทธิภาพในการค้นหาข้อมูลของระบบสืบค้นข้อมูล (Search Engine) เช่น Google Search หรือแม้แต่ในเครื่องมือและบริการออนไลน์ที่ให้บริการการจับคู่สตริงโดยประมาณ เช่น Google Docs และ Grammarly เป็นต้น

เมื่อพิจารณาลักษณะเฉพาะของหลักการเปรียบคู่สายอักขระแบบประมาณ คือ การค้นหาอักขระแบบเดี่ยว (Single Pattern String) $p = c_1c_2c_3...c_m$ (โดยกำหนด p แทนอักขระแบบเป้าหมายที่ต้องการค้นหา ที่ $c_1c_2c_3...c_m$ คือ สายอักขระที่ต่อเนื่องกันทีละอักขระมีขนาดความยาว m) ที่ปรากฏในข้อความ (Text) $T = t_1t_2t_3...t_n$ ใดๆ (โดยกำหนดให้ T คือข้อความหรือข้อมูลที่ประกอบด้วยอักขระขนาดยาว n ตัว ที่ต้องการนำอักขระแบบ p ไปค้นหา) โดยวัตถุประสงค์เพื่อพิจารณา p ปรากฏ ณ ตำแหน่งใดบ้างใน T โดยอนุญาตให้มีจำนวนอักขระที่เปรียบคู่ไม่ตรงกันหรืออนุญาตให้ผิดพลาดไม่ตรงกันด้วยค่าตัวเลขจำนวนเต็ม d กลไกการแก้ปัญหาคือนำเอาอักขระ $c_1c_2c_3...c_m$ เก็บไว้ในโครงสร้างข้อมูลที่เหมาะสม เรียกว่า ส่วนเตรียมการประมวลผล (Pre-processing) จากนั้นสร้างขั้นตอนวิธีค้นหาเรียกว่าการค้นหาหรือแมชชิ่ง (Searching or Matching)

ในช่วงเวลาที่ผ่านมา นักวิชาการทางด้านวิทยาการคอมพิวเตอร์พัฒนาโครงสร้างข้อมูลเพื่อรองรับการค้นหาขึ้นมาเป็นจำนวนมาก ดังปรากฏใน

สรุปโครงสร้างข้อมูลในคู่มือการเปรียบคู่ Navarro & Raffinot (2002) เช่น ใช้ออโตเมตัน (Automaton) ตารางการแฮช (Hashing Table) และตารางการแมชหรือไม่แมช (Matched or Mismatched Table) เป็นต้น ขั้นตอนวิธีค้นหาที่ใช้แก้ปัญหานี้ที่ได้รับความนิยม ได้แก่

1) Levenshtein Distance (Levenshtein, 1966)) เป็นขั้นตอนวิธีที่วัดระยะห่างระหว่างสองสตริง โดยพิจารณาจำนวนการลบ เพิ่ม หรือแทนที่ตัวอักษรที่จำเป็นเพื่อให้สองสตริงตรงกัน

2) Damerau-Levenshtein Distance เป็นขั้นตอนวิธีที่ขยาย Levenshtein Distance โดยพิจารณาการสลับที่ตำแหน่งของตัวอักษรด้วย

3) Needleman-Wunsch Algorithm (Needleman & Wunsch, 1970) เป็นขั้นตอนวิธีที่ค้นหาการจับคู่สตริงโดยประมาณโดยใช้การค้นหาแบบไดนามิก

4) Smith-Waterman Algorithm (Smith & Waterman, 1981) เป็นขั้นตอนวิธีที่คล้ายกับ Needleman-Wunsch Algorithm แต่พิจารณาค่าน้ำหนักของตัวอักษรแต่ละตัวในการจับคู่สตริง เป็นต้น การศึกษาวิจัยเพื่อพัฒนาขั้นตอนวิธีการเปรียบคู่แบบประมาณ มีการศึกษามาก่อนเป็นเวลานานมาแล้ว ที่น่าสนใจได้แก่ ในงานวิจัยของ Levenshtein (1965), Gusfield (1997), และ Karp & Rabin (1987) ซึ่งมีคำอธิบายโดยละเอียดปรากฏในหนังสือของ Navarro & Raffinot (2002)

จากนั้นมีการพัฒนาเนื้อหามาโดยตลอดจนกระทั่งถึงปัจจุบันก็ยังมีการศึกษาที่น่าสนใจ เช่น แสดงใน Uhlig *et al*. (2023) Khan, Halim, & Baig (2023), Faro & Scafiti (2022) และ Dondi, Mauri, & Zoppis (2022) ซึ่งจะพบว่าโดยส่วนมากแล้วขั้นตอนวิธีในปัจจุบันจะนำเทคนิคโครงสร้างข้อมูลและขั้นตอนวิธีที่พัฒนาไว้แล้วมาทำงานร่วมกับร่วมกับปัญญาประดิษฐ์ (Artificial Intelligence) (รายละเอียดแสดงในงานวิจัยของ Uhlig *et al*. (2023)) และเทคนิคต่างๆ เช่น กราฟ (รายละเอียดแสดงในงานวิจัยของ Khan, Halim, & Baig (2023) และ Dondi, Mauri, & Zoppis (2022)) เพื่อเพิ่มประสิทธิภาพการค้นหาให้รวดเร็วมากยิ่งขึ้น โดยงานวิจัยของ Boguszewski, Szyma*ń*ski, & Draszawka, (2016) ใช้พจนานุกรมพัฒนาต่อยอดวิธี Levenshtein Distance เพิ่มขึ้น ในขณะที่งานวิจัยของ Abraham & Raj (2014) พัฒนาวิธี Longest Subsequence (LCS) เพื่อสร้างการตรวจจับข้อความที่หลอกลวงต่างๆ เป็นต้น

จากการศึกษาข้างต้นพบว่า นักวิจัยพัฒนาโครงสร้างข้อมูลรวมถึงขั้นตอนวิธีในการแก้ปัญหาการเปรียบคู่สายอักขระแบบประมาณกันอย่างต่อเนื่องและแม้ว่าจะเป็นหลักการที่ได้ศึกษาพัฒนากันมาเป็นเวลานานแล้ว แต่อย่างไรก็ตามปัญหานี้ก็ยังคงอยู่ในความสนใจของนักวิจัยอยู่ตลอดเวลา ฉะนั้น การพัฒนาโครงสร้างข้อมูลและขั้นตอนวิธีการแก้ปัญหาการเปรียบคู่สายอักขระแบบประมาณแบบใหม่จึงยังมีความจำเป็นและเป็นที่ต้องการอย่างยิ่ง งานวิจัยนี้อาศัยแรงบันดาลใจจากงานวิจัยของ Khancome & Boonjing (2010) ซึ่งได้ออกแบบรายการผกผัน (Inverted Lists) สำหรับแก้ปัญหาการเปรียบคู่สายอักขระแบบไม่อนุญาตให้ผิดได้ ซึ่งทำให้เกิดโครงสร้างข้อมูลใหม่เพื่อใช้ในการสร้างขั้นตอนวิธีการเปรียบคู่สายอักขระแบบเดี่ยวได้อย่างมีประสิทธิภาพ

ดังนั้น ผู้วิจัยจึงนำแนวคิดโครงสร้างข้อมูลจากงานวิจัย Khancome & Boonjing (2010) มาพัฒนาโครงสร้างข้อมูลแบบใหม่ เพื่อใช้สำหรับการค้นหาข้อมูลสายอักขระแบบประมาณ โดยมีรายละเอียดดังนี้

1) กระบวนการพัฒนาโครงสร้างข้อมูลแบบใหม่ เรียกว่า รายการผกผันแบบเดี่ยวเพื่อการเปรียบคู่สายอักขระแบบประมาณ และ

2) การพัฒนาขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณโดยใช้โครงสร้างข้อมูลชนิดใหม่ที่พัฒนาขึ้น

ทั้งนี้ เป็นการพัฒนาโครงสร้างข้อมูลแบบใหม่ภายใต้สมมติฐานที่ว่า โครงสร้างข้อมูลใหม่ที่พัฒนาขึ้นจะมีประสิทธิภาพทั้งความซับซ้อนด้านเวลาและเนื้อที่ สามารถนำไปปรับใช้ในสาขาวิทยาการคอมพิวเตอร์ที่เกี่ยวข้องกับการเปรียบคู่โดยประมาณได้ รวมถึงมีความยืดหยุ่น รองกับการค้นหาที่รวดเร็วขึ้น ค้นหาข้อมูลทั้งการเปรียบคู่สายอักขระแบบโดยกำหนดค่าความผิดพลาดได้แบบยืดหยุ่น มีประสิทธิภาพ

2. แนวคิดทฤษฎีที่ใช้ในงานวิจัย

สำหรับการออกแบบงานวิจัยใหม่ครั้งนี้ คณะผู้วิจัยได้นำหลักการการสร้างรายการผกผันพื้นฐานที่เกิดจากการนำโครงสร้างข้อมูลที่ Khancome & Boonjing (2010) ออกแบบไว้ นำมาต่อยอดเพื่อพัฒนาขั้นตอนวิธีใหม่ โดยอาศัยตารางการแฮชเพื่อจัดเก็บและเข้าถึงข้อมูล นำเสนอรายละเอียดดังนี้

2.1 รายการผกผัน (Inverted Lists: IVL)

รายการผกผัน คือโครงสร้างข้อมูลที่ออกแบบโดย Khancome & Boonjing (2010) ซึ่งออกแบบไว้เพื่อใช้สำหรับแก้ปัญหาทางด้านค้นหาสายอักขระแบบที่ปรากฏในข้อความ (Text) ในขั้นตอนวิธีเปรียบคู่สายอักขระแบบตรงกันทั้งหมด (Exact String Matching Algorithm) ด้วยการนำสายอักขระแบบ (Pattern of String) มาเขียนให้อยู่ในรูปแบบตารางขนาด 2 คอลัมน์ โดยคอลัมน์แรกคือ Σ ใช้บรรจุอักขระเดี่ยว (Single Character) ที่ปรากฏในอักขระแบบที่นำมาพิจารณา ส่วนคอลัมน์ที่สองชื่อ IVL ใช้บรรจุตำแหน่งของอักขระเดี่ยวที่สอดคล้องกับคอลัมน์แรก

กลไกการออกแบบรายการผกผัน เริ่มต้นจากการนำแนวคิดของหลักการดัชนีผกผัน (Inverted Index) ซึ่งเป็นหลักการที่มีประสิทธิภาพสูงสำหรับการออกแบบการจัดเก็บดัชนีของการจัดเก็บและค้นคืนเอกสาร หลักการของดัชนีผกผันจะให้หมายเลขแก่เอกสาร และระบุตำแหน่งของแต่ละคำหรือข้อความที่ปรากฏในเอกสาร จากนั้นนำหมายคำต่างๆ เหล่านั้นมาสร้างเป็นดัชนีเพื่อให้สามารถถึงเอกสารได้เร็วขึ้น เขียนโครงสร้างในรูปแบบ “<หมายเลขเอกสาร, คำที่ปรากฏ:ตำแหน่งที่ปรากฏ>” เรียกคำที่ปรากฏว่าคีย์เวิร์ดหรือคำศัพท์ โดยระบุคู่กับตำแหน่งที่ปรากฏในเอกสารที่แสดงแนวคิดการสร้างอักขระแบบให้อยู่ในรูปแบบดังกล่าว โดย Khancome & Boonjing (2010) ได้นำอักขระแบบเป้าหมายมาพิจารณาและเขียนให้อยู่ในรูปแบบรายการผันด้วยการระบุอักขระเดี่ยวที่ปรากฏในอักขระ โดยมีรูปแบบ “<ตำแหน่งที่ปรากฏ:สถานะเป็นตัวสุดท้ายของอักขระแบบหรือไม่ (0=ไม่ใช่/1=ใช่)>” นำเสนอตัวอย่างรายการผกผันของอักขระแบบพอสังเขป ดังตัวอย่างที่ 1

ตัวอย่างที่ 1 กำหนดให้อักขระแบบ p = CCOMZ สามารถแสดงรายการผกผันของ p ได้ดังตาราง 1

ตาราง 1 รายผกผันของอักขระแบบ p = CCOMZ

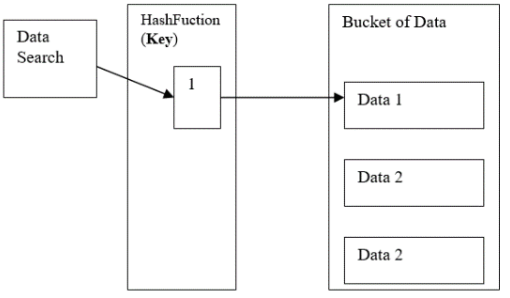
Σ	IVL
C	<1:0>, <2:0>
O	<3:0>
M	<4:0>
Z	<5:1>

จากตารางที่ 1 อธิบายได้ว่า อักขระแบบ CCOMZ มีอักขระเดี่ยวที่ปรากฏในอักขระแบบคือ C, O, M และ Z ซึ่งถูกนำมาเขียนบรรจุในตารางดังแสดงในคอลัมน์แรก Σ จากนั้นพิจารณาตำแหน่งที่ปรากฏของอักขระแต่ละตัวที่ปรากฏในอักขระแบบ

ดังกล่าว จะพบกว่า อักขระ C ปรากฏที่ตำแหน่ง 1 และ 2 ส่วน O ปรากฏในตำแหน่งที่ 3 อักขระM ปรากฏในตำแหน่งที่ 4 และ Z ปรากฏในตำแหน่งที่ 5 พิจารณาสถานะการเป็นอักขระตัวสุดท้ายของอักขระแบบหรือไม่จะพบว่า C O และ M ไม่ใช่อักขระสุดท้ายของอักขระแบบจึงกำหนดสถานะคือ 0 แต่ Z เป็นอักขระตัวสุดท้ายของอักขระแบบนี้จึงกำหนดสถานะเป็น 1 อักขระทั้งหมดจึงถูกเขียนในรูปแบบ อักขระเดี่ยวที่ปรากฏในอักขระแบบ: <ตำแหน่งที่ปรากฏ:สถานะเป็นตัวสุดท้ายของอักขระแบบหรือไม่ (0=ไม่ใช่/1=ใช่)> ดังแสดงในตารางดังกล่าวข้างต้น

2.2 ตารางการแฮช (Hashing Table)

หลักการแฮช (Hashing Principle) หรือตารางการแฮช คือหลักการสร้างและออกแบบโครงสร้างข้อมูลจัดเก็บในรูปแบบตารางเพื่อสนับสนุนการค้นหาและเข้าถึงข้อมูลในโครงสร้างด้วยค่าความซับซ้อน O(1) องค์ประกอบของหลักการนี้ มี 2 ส่วนคือการสร้างฟังก์ชันคีย์ เรียกว่าแฮชฟังก์ชัน (Hash Function) และตารางเก็บข้อมูล (Bucket of Data) โดยที่ผลการของนำค่าคีย์สำหรับค้นหา (Data Search) นำมาคำนวณผ่านแฮชฟังก์ชัน จะได้ผลลัพธ์เป็นตำแหน่งของการจัดเก็บข้อมูล แสดงแนวคิดของโครงสร้างดังกล่าว ดังภาพประกอบ 1

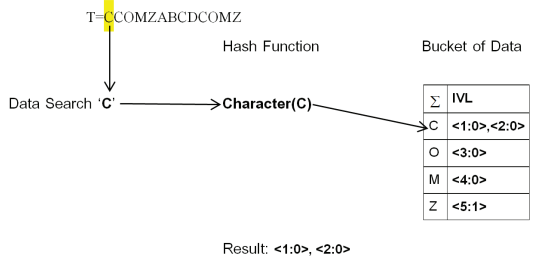


ภาพประกอบ 1 แนวคิดการสร้างตารางการแฮช

จากแนวคิดในภาพประกอบ 1 งานวิจัยนี้ได้นำเสนอรูปแบบของการแสดงรายการผกผัน (แสดงในหัวข้อ 2.1) โดยใช้ Character (W_λ) เป็นฟังก์ชันการแฮช เพื่อเข้าถึงตารางการแฮชในส่วนของ Bucket of Data ในภาพประกอบดังกล่าว โดยที่ W_λ คืออักขระเดี่ยวที่ปรากฏในอักขระแบบ p ใดๆ แนวคิดการเข้าถึงข้อมูลและการเปรียบเทียบตัวอักขระเพื่อจะนำไปวิเคราะห์การค้นพบ แสดงตัวอย่างการสร้างตารางการแฮชของรายการผกผัน p = CCOMZ ประกอบการเข้าถึงด้วยข้อความ T ดังภาพประกอบ 2

จากแนวคิดการสร้างตารางการแฮชแบบนี้กับรายการผกผันที่ได้นำเสนอไปแล้ว ทำให้สามารถเข้าถึงข้อมูลเพื่อเปรียบเทียบในขั้นตอนวิธีการค้นหาสามารถทำได้ด้วยค่าความซับซ้อน O(1) โดยใช้เนื้อที่จัดเก็บเป็น O(n) ซึ่งจะได้แสดงการพิสูจน์และนำเสนอในลำดับต่อไป

จากแนวคิดดังกล่าว Khancome & Boonjing (2010) ได้นำไปออกแบบขั้นตอนวิธีการเปรียบคู่แบบตรงกัน (Exact String Matching Algorithms) จำนวน 2 ขั้นตอนวิธี ทำงานด้วยความซับซ้อนขั้นตอนวิธีแบบ Prefix Approach ค้นหาข้อมูล O(mn) กับขั้นตอนวิธี Suffix Approach ค้นหาข้อมูลด้วยความซับซ้อน O(n/m) โดยขั้นตอนวิธีทั้งสองใช้โครงสร้างรายการผกผันที่สร้างขึ้นทำงานด้วยค่าความซับซ้อน โดยพัฒนาโปรแกรมเปรียบเทียบ



ภาพประกอบ 2 แนวคิดการสร้างตารางการแฮช สำหรับการเปรียบคู่แบบประมาณ

กับขั้นตอนวิธีการเปรียบคู่แบบตรงกันทั้งหมด เช่น Boyer-Moor(BM), KMP(Knuth-Morris-Pratt) และ BruceForce(BF) ซึ่งไม่สามารถรองรับการค้นหาแบบประมาณดังงานวิจัยใหม่นี้ได้

ดังนั้น งานวิจัยใหม่นี้นำหลักการดังกล่าวสร้างตารางการแฮชเก็บรายการผกผันดังกล่าว เพื่อรองรับการค้นหาเปรียบเทียบแบบประมาณหรืออนุญาตให้มีค่าความผิดพลาดในการค้นหาได้ ทั้งนี้เพื่อให้การออกแบบและเข้าถึงข้อมูลผ่านตารางได้สะดวกรวดเร็วและมีประสิทธิภาพจึงออกแบบตารางดังกล่าวด้วยตารางการแฮช คล้ายกับงานวิจัย Khancome & Boonjing (2010) เฉพาะบางส่วนของตารางรายการผกผันเท่านั้น แต่หากขั้นตอนวิธีที่นำเสนอเป็นขั้นตอนวิธีใหม่ที่ใช้สำหรับการเปรียบคู่ค้นหาแบบประมาณที่อนุญาตให้ระบุค่าความผิดพลาดได้ และได้ทดลองเปรียบเทียบกับขั้นตอนวิธีที่มีชื่อเสียงของกลุ่มงานวิจัยที่เกี่ยวข้องด้านนี้เท่านั้น

3. วิธีการดำเนินการวิจัย

3.1 ขั้นตอนการสร้างโครงสร้างข้อมูล

เบื้องต้น กำหนดให้ p คือสายอักขระจาก c_1, c_2, ..., c_m ที่อยู่ภายใต้ $\sum$ เมื่อ $\sum$ คือเซตของอักขระที่ปรากฏใน p

นิยามที่ 1 คำสำคัญของอักขระแบบ คือ p ประกอบด้วยคำสำคัญ $w_{a_{1,0}}$, $w_{b_{2,0}}$, $w_{c_{3,0}}$, ..., $w_{\cdots_{m,1}}$ เมื่อ $w_{n_{k,0}}$ หรือ $w_{n_{k,1}}$ ใดๆ คือ c_k โดยที่ k คือตำแหน่งที่ปรากฏอยู่ใน p ; 0 หรือ 1 คือสถานะที่ระบุการเป็นอักขระสุดท้ายของ c_k ใน p ถ้าไม่ใช่อักขระสุดท้ายระบุ 0 หรือระบุ 1 หากเป็นอักขระสุดท้าย

$$\omega = w_{a_{1,0}}, w_{b_{2,0}}, w_{c_{3,0}}, ..., w_{\cdots_{m,1}} \quad (1)$$

ตัวอย่างที่ 1 แสดงคำสำคัญของอักขระแบบ p=aabcz ซึ่งจะได้ $w_{a_{1,0}}$ = a, $w_{b_{2,0}}$ = a, $w_{c_{3,0}}$ = b, $w_{d_{4,0}}$ = c และ $w_{e_{5,1}}$ = z เขียนเป็นในรูปแบบคำสำคัญได้ดังนี้

$$\omega = a_{1,0}\, a_{2,0}\, b_{3,0}\, c_{4,0}\, z_{5,1}$$

นิยามที่ 2 รายการผกผัน L ของ ω เกิดจากนำแต่ละ $w_{\lambda_{\varepsilon,0}}$ หรือ $w_{\lambda_{\varepsilon,1}}$ ใดๆ ที่อยู่ใน ω มาเขียนในรูปแบบ $w_\lambda :< \varepsilon : 0 >$ หรือ $< \varepsilon : 1 >$ ซึ่ง

$$L_\omega = w_a :< 1 : 0 > \quad w_b :< 2 : 0 > , \; w_c :< 3 : 0 >, ..., w_{\cdots} :< m : 1 > \quad (2)$$

ตัวอย่างที่ 2 แสดงรายการผกผันของอักขระแบบ p=aabcz ได้ดังนี้

L_ω = a:<1:0>, a:<2:0>, b:<3:0>, c:<4:0>, z:<5:1>

นิยามที่ 3 จากนิยาม 2 ให้ I_{λ_0} แทน $< i : 0 >$ และ I_{λ_1} แทน $< i : 1 >$ ของ w_λ ใดๆ ดังนั้น

$$w_\lambda : I_{\lambda_0} \text{ หรือ } w_\lambda : I_{\lambda_1} \quad (3)$$

นิยามที่ 4 ตารางรายการผกผัน τ คือตารางการแฮชที่ประกอบด้วย 2 คอลัมน์คือ w_λ และ I_{λ_0} / I_{λ_1} (4)

ตัวอย่างที่ 3 แสดงตารางรายการผกผัน τ ของอักขระแบบ p=aabcz

ตาราง 2 ตารางรายผกผันของอักขระแบบ *p = aabcz*

Character (w_λ)	Inverted Lists (I_{λ_0} / I_{λ_1})
a	<1:0>,<2:0>
b	<3:0>
c	<4:0>
z	<5:1>

Algorithm 1: Inverted-List Table ($p = c_1, c_2, c_3, ..., c_m$)

1: Create table for all alphabet over $\sum$
2: $j = 1$
3: While ($j \leq m$) Do
4: Create inverted list of char (C_j) and add to at suitable field I_{λ_0} or I_{λ_1}
5: $j \leftarrow j + 1$
6: End of While

ทฤษฎีบท 1 การเข้าถึงรายการผกผัน I_{λ_0} หรือ I_{λ_1} ใน τ มีความซับซ้อนด้านเวลา O(1)

พิสูจน์ กำหนด f(x) เป็นฟังก์ชันการแฮช; ให้ w_{λ_0} คือ คีย์เข้าถึง I_{λ_0} และ w_{λ_1} คือ คีย์เข้าถึง I_{λ_1} ดังนั้น การเข้าถึงตารางการแฮช τ ตามคุณสมบัติของตารางการแฮช ทำให้ การเข้าถึง I_{λ_0} จาก f(w_{λ_0}) หรือ I_{λ_1} จาก f(w_{λ_1}) มีความซับซ้อน O(1) #

กลไกการทำงานของขั้นตอนวิธี เริ่มจากสร้างตาราง τ ให้สามารถครอบคลุมการบรรจุอักขระทุกตัวที่จะใช้งาน ($\sum$) จากอ่านอักขระทีละอักขระจากอักขระแบบสร้าง รายการผกผัน I_{λ_0} หรือ I_{λ_1} บรรจุลงใน τ จนครบทุกอักขระ ขั้นตอนวิธีแสดงได้นี้

จากขั้นตอนวิธีใน Algorithm 1 พบว่ามีความซับซ้อนด้านเวลาคือ O(m) พิสูจน์ได้ดังทฤษฎีบท 2 ความซับซ้อนด้านเนื้อที่เป็น O(1) เนื่องจากใช้ตารางการแฮช ตามนิยาม 4 ที่มีจำนวนแถวเท่ากับ $\sum$ เท่านั้น

ทฤษฎีบท 2 ส่วนเตรียมการการประมวลผลเพื่อสร้างตารางรายการผกผันสำหรับการเปรียบคู่โดยประมาณใช้เวลา O(m) ด้วยเนื้อที่ $0(|\sum| + m)$

พิสูจน์ จาก Algorithm 1 ที่ p=c_1, c_2, c_3, ..., c_m ซึ่ง มีความยาว m บรรทัดที่ 1 การสร้างตารางเพื่อเก็บรายการผกผันและ บรรทัดที่ 2 กำหนดตัวแปรเริ่มต้น ต่างทำด้วยความซับซ้อนค่าคงที่ O(1) ลูป While จะวนเท่ากับจำนวนอักขระแบบ คือ m รอบ ซึ่งความซับซ้อนเท่ากับ O(m) บรรทัดที่ 4 เข้าถึงตารางเพื่อการเพิ่มรายการผกผัน ซึ่งเข้าถึงรายการผกผันด้วย O(1) ตามทฤษฎีบท 1 ขณะที่บรรทัดที่ทำงานที่ O(1) ซึ่งทำเท่ากับ m ครั้งภายในลูป While

ดังนั้น ความซับซ้อนด้านเวลาสำหรับสร้างรายการผกผันบรรจุลง I_{λ_0} และ I_{λ_1} จึงเท่ากับ O(m) สำหรับความซับซ้อนด้านเนื้อที่จะต้องมี 2 คอลัมน์ $\sum$ ในขณะที่รายการผกผันที่สร้างขึ้นมีค่าเท่ากับ m รายการ ซึ่งทำให้ความซับซ้อนด้านเนื้อที่คือ $0(|\sum| + m)$#

3.2 ขั้นตอนวิธีการค้นหาแบบประมาณ

เนื้อส่วนนี้นำเสนอส่วนสำคัญก่อนพิจารณาขั้นตอนวิธีการเปรียบคู่แบบประมาณที่สร้างขึ้นจำเป็นต้องแสดงการพิสูจน์บทแทรก และกำหนดนิยามที่เกี่ยวข้องเพื่อให้การอธิบายขั้นตอนวิธีและตัวอย่างการค้นหาได้อย่างกระจ่างมากยิ่งขึ้น ดังนี้

บทแทรก 1 กำหนดให้ IVL คือตารางการแฮชย่อยที่มี w_{λ_0} และ w_{λ_1} เป็นคีย์เข้าถึง I_{λ_0} และ I_{λ_1} ซึ่งการเข้าถึง I_{λ_0} และ I_{λ_1} ใน IVL ด้วยฟังก์ชัน f(w_{λ_0}) หรือ f(w_{λ_1}) มีความซับซ้อน O(1)

พิสูจน์ กำหนด IVL เป็นตารางการแฮช ในนิยามที่ 4 และทฤษฎีบท 1 ซึ่งมีคีย์ w_{λ_0} และ w_{λ_1} ดังนั้นจะใช้ f(w_{λ_0}) และ f(w_{λ_1}) เพื่อเข้าถึง I_{λ_0} และ I_{λ_1} ด้วยความซับซ้อน O(1) ตามทฤษฎีบท 1 #

บทแทรก 2 การนำรายการผกผันจาก τ ที่ตรงกับรายการผกผัน text[N] ลง IVL ใดๆ ใช้ความซับซ้อน O(1)

พิสูจน์ กำหนดให้ text[N] คือ อักขระจากข้อความ T ที่แปลงเป็นเป็นคีย์ได้ $w_{\lambda_{pos,0}}$ และ $w_{\lambda_{pos,1}}$

ดังนั้น เมื่อเข้าถึง $I_{\lambda_{pos,0}}$ และ $I_{\lambda_{pos,1}}$ ในตาราง τ จึงใช้ความซับซ้อน O(1) ตามทฤษฎีบท 1 และนำ $I_{\lambda_{pos,0}}$ และ $I_{\lambda_{pos,1}}$ ลง IVL ด้วย O(1) ตามบทแทรก 1 #

นิยามที่ 5 การดำเนินการ (operate) คือการหาความต่อเนื่องของ $I_{\lambda q_{\varepsilon 1,0}}$ และ/หรือ $I_{\lambda q_{\varepsilon 1,1}}$ ใน IVL1 ต่อเนื่องไปยัง $I_{\lambda b_{\varepsilon 2,0}}$ และ/หรือ $I_{\lambda b_{\varepsilon 2,1}}$ ใน IVL2 โดยพิจารณาตำแหน่ง $\varepsilon 2$ ที่ต่อเนื่องมาจาก $\varepsilon 1$ ซึ่งจะได้ผลการดำเนินการเป็น $I_{\lambda b_{\varepsilon 2,0}}$ และ/หรือ $I_{\lambda b_{\varepsilon 2,1}}$

Algorithm 2: Inverted-List-Approximate-Search $p = c_1, c_2, c_3, ..., c_m, T = t_1, t_2, ..., t_n, d$

Preprocessing:

Create Inverted-List-Table (p)

Searching:

```
N=1, SearchWindow=1, pos=1, IVL1=ϕ, IVL2=ϕ, f=0
    While (N<=n-d) Do
        Store all member of row(text[N]) I_λpos,0 / I_λpos,1 in τ to IVL1
        While (f<d and pos<=m)
            If IVL1 =ϕ and f<=d
                IVL1 = <pos:0> if pos<m or IVL1=<pos:1> if pos=m
                N=N+1, pos=pos+1
            Else
                Keep only <pos:0> in IVL1
            N=N+1, pos=pos+1
            Store all member of row(text[N]) I_λpos,0 / I_λpos,1 in τ to IVL2
            Analyze continuity of IVL1 and IVL2 and keep it into IVL1 set f=f+1 if non-continuity
            Report result of matching if IVL1 exist <pos:1> and pos=m
        End of If
        Report result of matching if IVL1 exist <pos:1> and pos=m or f<=d and pos=m
    End of While
        SearchWindow=SearchWindow+1
        N=SearchWindow, pos=1
    End of While
```

ตาราง 3 รายการผกผัน τ ของอักขระแบบ p=aabcz

Character (w_λ)	Inverted Lists (I_{λ_0} / I_{λ_1})
a	**<1:0>,<2:0>**
b	**<3:0>**
c	**<4:0>**
z	**<5:1>**

ตัวอย่างที่ 5 ถ้ากำหนดให้ IVL1={<2:0>} และ IVL2={<1:0>,<3:0>} การหาความต่อเนื่องจากตำแหน่งที่ 2 ไป 3 จะได้ว่า <3:0> ต่อเนื่องมาจาก <2:0> และมีรายการผกผันใหม่คือ <1:0> ซึ่งจะได้ IVL1 ={<1:0>,<3:0>} เป็นต้น

บทแทรก 3 การดำเนินการระหว่าง IVL1 และ IVL2 ใช้ความซับซ้อน O(1)

พิสูจน์ กำหนดให้ IVL1 และ IVL2 คือ IVL ตามบทแทรก 1 โดยที่ IVL1 บรรจุ $I_{\lambda_{\varepsilon 1,0}}$ และ/หรือ $I_{\lambda_{\varepsilon 1,1}}$, IVL2 บรรจุ $I_{\lambda_{\varepsilon 2,0}}$ และ/หรือ $I_{\lambda_{\varepsilon 2,1}}$ การเข้าถึงเพื่อนำ $I_{\lambda_{\varepsilon 1,0}}, I_{\lambda_{\varepsilon 1,1}}, I_{\lambda_{\varepsilon 2,0}}$ และ $I_{\lambda_{\varepsilon 2,1}}$ มาเปรียบเทียบสำหรับการดำเนินการ ตามนิยามที่ 5 จะใช้ O(1) เป็นไปตามบทแทรก 1#

การทำงานของขั้นตอน เริ่มการเปรียบคู่จากตัวแรกของหน้าต่าง (Search Window) เก็บรายการผกผัน I_{λ_0} ไว้ใน IVL1 โดยให้ตัวแปร N ไปชี้ยังตำแหน่งที่ต้องการเปรียบเทียบ ขั้นตอนต่อไปอ่านและเปรียบคู่อักขระในข้อความจากซ้ายไปขวาทีละอักขระ ซึ่งแต่ละครั้งของการเปรียบคู่ ใช้วิธีเก็บผลลัพธ์ของรายการผกผันไว้ใน IVL1 และ IVL2 แล้วนำรายการผกผันดังกล่าวมาหาความต่อเนื่องและผลการเปรียบคู่สำเร็จไปพร้อมๆ กัน แสดงขั้นตอนวิธีดัง Algorithm 2 สำหรับค่าความผิดพลาดที่อนุญาตให้การเปรียบคู่ไม่ตรงกันได้ กำหนดให้เป็นค่า d นอกจากนั้นมีตัวแปรพิเศษคือ f คือ ค่าจดจำครั้งของการผิดพลาดเมื่อทำการเปรียบคู่แต่ละหน้าต่าง ตัวแปร pos คือตำแหน่งรายการผกผันที่ต้องการนำมาเปรียบคู่ ขั้นตอนวิธีการค้นหาเปรียบคู่แบบประมาณที่พัฒนาขึ้น แสดงดัง Algorithm 2

ตัวอย่างที่ 6 แสดงการค้นหาแบบประมาณด้วย Algorithm 2 เมื่อ T = axbczaabczaabmk และ p=aabcz กำหนดค่า d = 2

หน้าต่างค้นหาที่ 1 : SearchWindow = 1

T = a x b c z a a b c z a a b m k
p = a a b c z

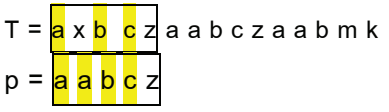
IVL1 = <1:0>,<2:0>, N=1, มีข้อมูล text[N] คือ a ใน τ, N=N+1, f=0, pos=1

T = a x b c z a a b c z a a b m k
p = a a b c z

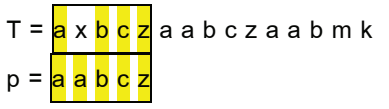
IVL1 = <1:0>,<2:0>, IVL2=<> N=2, pos=2 ไม่พบข้อมูลของ text[N] คือ x กำหนด f=f+1=1 ซึ่งแสดงว่าเปรียบคู่อักขระไม่ตรงกัน แต่ค่า f<= d จึงสามารถเปรียบคู่ต่อไปไปได้ตามหลักของขั้นตอนวิธี เก็บค่า ความต่อเนื่องของอักขระจาก IVL1 = <1:0>, pos=1 เปลี่ยนไปเป็น 2 ได้โดยเก็บค่า IVL1= <pos:0>=<2:0> แล้วค้นหาต่อไป

T = a x b c z a a b c z a a b m k
p = a a b c z

IVL1 = <2:0>, IVL2=<3:0> N=3, pos=3 พบข้อมูลของ text[N] คือ b กำหนด f= 1 วิเคราะห์ความต่อเนื่อง IVL1 = <2:0>, IVL2=<3:0> เก็บผลไว้ใน IVL1 = =<3:0> ซึ่งแสดงว่าเปรียบคู่อักขระตรงกัน แต่ pos<=m แล้วค้นหาต่อไป

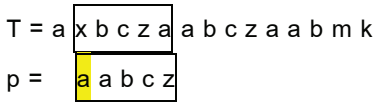


IVL1 = <3:0>, IVL2=<4:0> N=4, pos=4 พบข้อมูลของ text[N] คือ c กำหนด f= 1 วิเคราะห์ความต่อเนื่อง IVL1 = <3:0>, IVL2=<4:0> เก็บผลไว้ใน IVL1 = =<4:0> ซึ่งแสดงว่าเปรียบคู่อักขระตรงกัน แต่ pos<=m แล้วค้นต่อไป

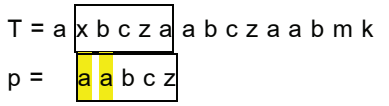


IVL1 = <4:0>, IVL2=<5:1> N=5, pos=5 พบข้อมูลของ text[N] คือ z กำหนด f= 1 วิเคราะห์ความต่อเนื่อง IVL1 = <4:0>, IVL2=<5:1> วิเคราะห์การเปรียบคู่อักขระแบบสำเร็จที่ ตำแหน่งของ text[N] คือ z ด้วยความผิดพลาด คือ 1 ตำแหน่งอักขระเป็นต้น pos=5 ซึ่งเท่ากับ m จบหน้าต่างค้นหาที่ 1 เลื่อนหน้าต่างค้นหา

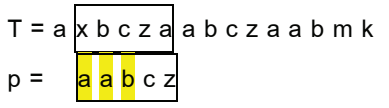
หน้าต่างค้นหาที่ 2 : SearchWindow = 2



VL1 = <>, N=1, ไม่มีข้อมูล text[2] คือ x ใน τ , N=N+1, f=1, pos=1 แต่เก็บ IVL1=<1:0> ตามขั้นตอนวิธี

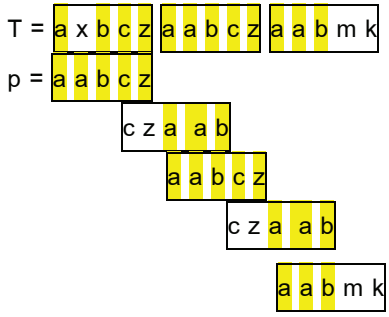


VL1 = <1:0>, IVL2=<> N=2, pos=2 ไม่พบข้อมูลของ text[N] คือ b กำหนด f=f+1=2 แต่ค่า f<= d จึงสามารถเปรียบคู่ต่อไปได้ตามหลักของขั้นตอนวิธี เก็บค่า ความต่อเนื่องของอักขระจาก IVL1 = <2:0>, pos=1 เปลี่ยนไปเป็น 2 ได้ โดยเก็บค่า IVL1= <pos:0>=<2:0> แล้วค้นต่อไป



IVL1 = <2:0>, IVL2=<4:0> N=3, pos=3 พบข้อมูลต่อเนื่องของ text[N] คือ c กำหนด f= 3 วิเคราะห์ความต่อเนื่องและการเปรียบคู่ ค่า f>d แสดงว่าการเปรียบคู่ไม่สำเร็จ เลื่อนเป็นหน้าต่างการค้นหาใหม่ SearchWindow 3 ต่อไป

สำหรับ หน้าต่างค้นหาที่ 3-5 จะเกิดการเปรียบคู่ไม่สำเร็จดังหน้าต่างที่ 2 คือ ค่า f>d จะตรงกันอีกครั้งที่หน้าต่างค้นหาที่ 6 แต่กรณีนี้ค่า f=0 ซึ่งเป็นการเปรียบคู่สำเร็จแบบตรงกันทุกตัวอักขระ นอกจากนั้น ในหน้าต่างค้นหาที่ 11 ในขณะที่หน้าต่างที่ 7-10, 12 จะเปรียบคู่ไม่สำเร็จด้วยค่า f>d แสดงหน้าต่างที่ค้นพบดังนี้



ทฤษฎีบท 3 ขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณ มีความซับซ้อนด้านเวลากรณีเฉลี่ย O((m-k)n) กรณีดีที่สุด O(kn) เมื่อค่า k คือค่าที่กำหนดให้ผิดพลาดในการค้นหาได้ m คือความยาวของอักขระแบบ n คือ ความยาวของข้อความที่นำมาใช้ค้นหา

พิสูจน์ กำหนด n คือความยาวของข้อความ T ซึ่ง $T=t_1t_2t_3...t_n$ อักขระแบบยาว m กำหนดค่า k คือค่าที่อนุญาตให้ผิดพลาดได้ (ในที่นี้คือ d ใน Algorithm 2) พิจารณาความซับซ้อนเริ่มจากอธิบายการวนลูปจากลูปของการเปรียบเทียบแต่ละหน้าต่าง (ลูป While บรรทัดที่ 4-16) จากนั้นอธิบายส่วนลูปภายนอกที่ควบคุมการกวาดเพื่อเปรียบเทียบไปทีละหน้าต่าง

พิจารณาลูป While ภายใน (บรรทัดที่ 4-16) คือลูปของการเปรียบเทียบด้วยความยาวมากที่สุดคือ m ครั้ง เปรียบเทียบน้อยที่สุดเท่ากับ m-d ซึ่งคือ (m-k) นั้นเอง ในขณะที่ในแต่ละบรรทัดของการดำเนินการทำด้วยความซับซ้อน O(1) ตามบทแทรกที่ 1, 2 และ 3

สำหรับลูป While นอกสุด วนทำงานมากที่สุด n ครั้ง หรือน้อยที่สุดคือ n-(d+1) ครั้งเท่านั้น ดังนั้นหากพิจารณาลำดับการวนเพื่อเปรียบเทียบทั้งหมด (เฉลี่ย) จะเท่ากับ (m-d)xn หรือกรณีน้อยที่สุดของการกวาด คือ (m-d)x n-(d+1) ซึ่งไม่ว่าการทำงานจะเป็นกรณีใด ก็จะนำไปสู่ความซับซ้อน O((m-k)n) #

กรณีที่ดีที่สุดของการค้นหาจะเป็นกรณีที่เปรียบเทียบที่ไม่ตรงกันด้วยจำนวนครั้งของการทำงานที่ตรวจไม่พบที่ เปรียบเทียบเท่ากับ d ครั้งในแต่ละหน้าต่าง ซึ่งคือค่า k ทำให้ความซับซ้อนดีที่สุดคือ O(kn)#

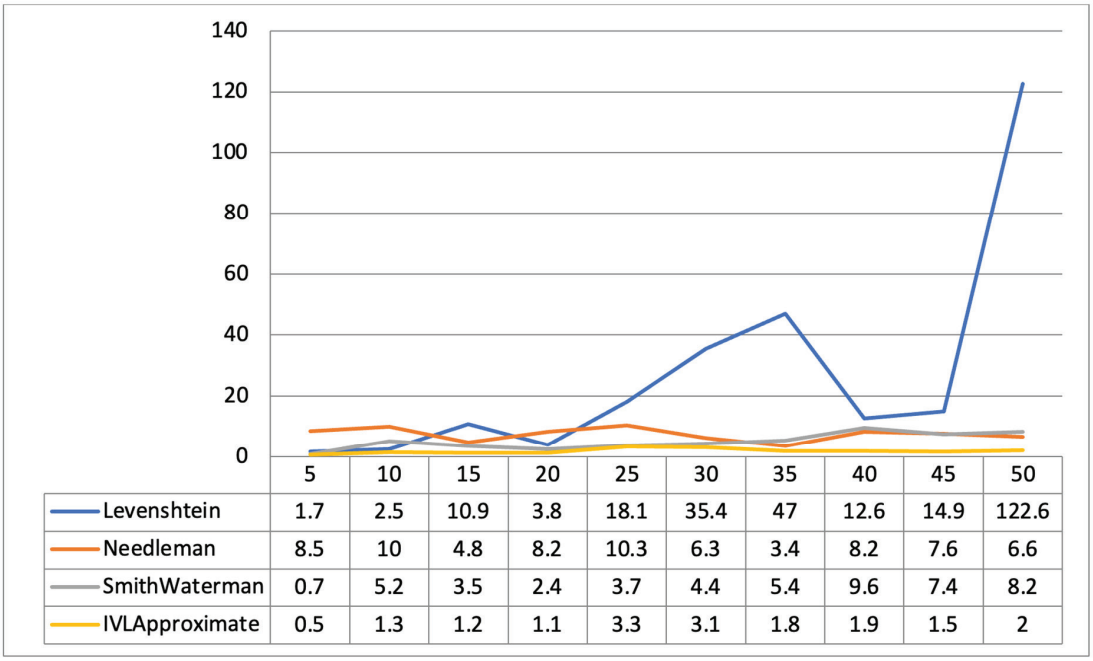
4. ผลการทดลอง

การทดลองพัฒนาโปรแกรมการเปรียบคู่แบบประมาณ ด้วยโปรแกรมภาษาจาวา (jdk 21), Netbeans 20 ทดลองด้วยเครื่องคอมพิวเตอร์ Dell Vostro 5400, Windows 10 Pro, Intel(R) Core(TM) i7-7700HQ RAM 16.0 GB โดยพัฒนาโปรแกรมโครงสร้างข้อมูลและขั้นตอนวิธีที่ออกแบบใหม่ กำหนดให้ชื่อ IVLApproximate หลังจากนั้นนำโปรแกรมที่พัฒนาขึ้นทดลองโดยอาศัยข้อมูลทั้งข้อมูลที่โปรแกรมคอมพิวเตอร์สุ่มขึ้นและข้อมูลจริงมาตรฐาน เปรียบกับขั้นตอนวิธีที่มีประสิทธิภาพที่มีมาก่อน ได้แก่ Levenshtein (จาก https://www.baeldung.com/java-levenshtein-distance), Neesleman (จาก https://github.com/Aqcurate/Needleman-Wunsch/blob/master/NeedlemanWunsch

.java), SmithWaterman (จาก https://github.com/JayakrishnaThota/Sequence-Alignment/blob/master/SmithWaterman.java) ทดลองใน 4 ประเด็น ดังนี้

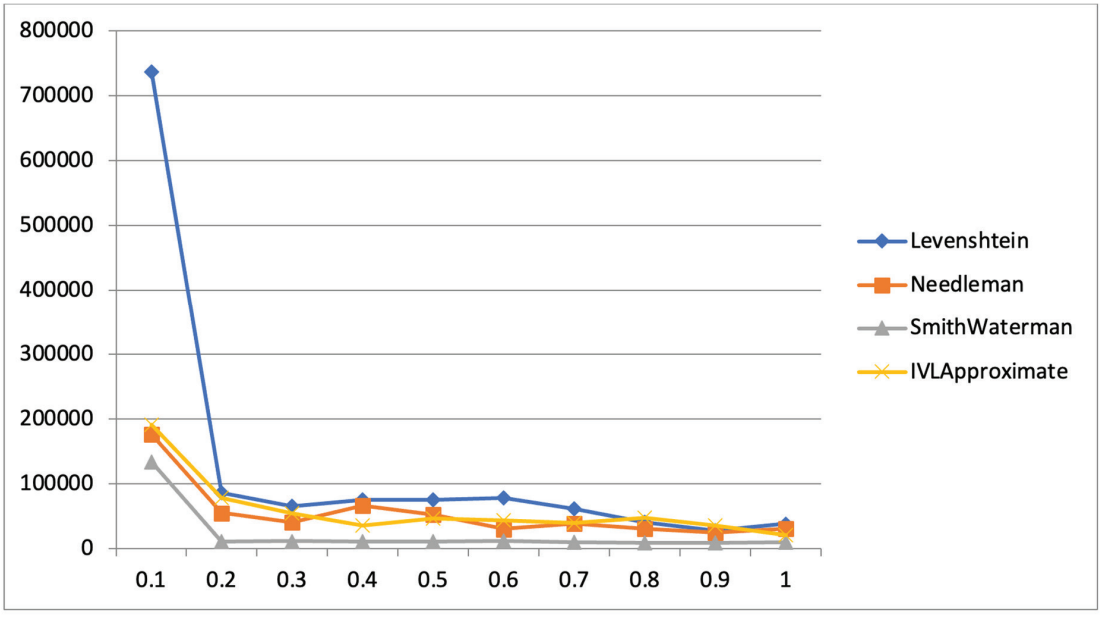
เขียนโปรแกรมสุ่มตัวอักขระแบบด้วยความยาวระหว่าง 5-50 อักขระ และวัดหน่วยความจำที่ใช้ระหว่างประมวลผลเพื่อจัดเตรียมสำหรับการค้นหาแสดงดังภาพประกอบ 3 (วัดในหน่วยเมกะไบต์)

จากภาพประกอบ 3 แสดงให้เห็นว่าการสร้างโครงสร้างสำหรับจัดการอักขระแบบของ IVLApproximate ซึ่งเป็นโครงสร้างในงานวิจัยนี้ใช้เนื้อที่ในการสร้างต่ำสุด โดยเป็นเชิงเส้นอยู่ไม่เกิน 2 MB ในหน่วยความจำขณะที่ขั้นตอนวิธีที่นำมาเปรียบเทียบจำเป็นต้องใช้เนื้อที่หน่วยความจำมากกว่าดังแสดงในภาพประกอบ 3 ดังกล่าว



	5	10	15	20	25	30	35	40	45	50
Levenshtein	1.7	2.5	10.9	3.8	18.1	35.4	47	12.6	14.9	122.6
Needleman	8.5	10	4.8	8.2	10.3	6.3	3.4	8.2	7.6	6.6
SmithWaterman	0.7	5.2	3.5	2.4	3.7	4.4	5.4	9.6	7.4	8.2
IVLApproximate	0.5	1.3	1.2	1.1	3.3	3.1	1.8	1.9	1.5	2

ภาพประกอบ 3 เนื้อที่สำหรับสร้างโครงสร้างข้อมูลสำหรับค้นหา



ภาพประกอบ 4 แสดงอัตราการอนุญาตให้ผิดพลาดได้ (Distance-ratio)

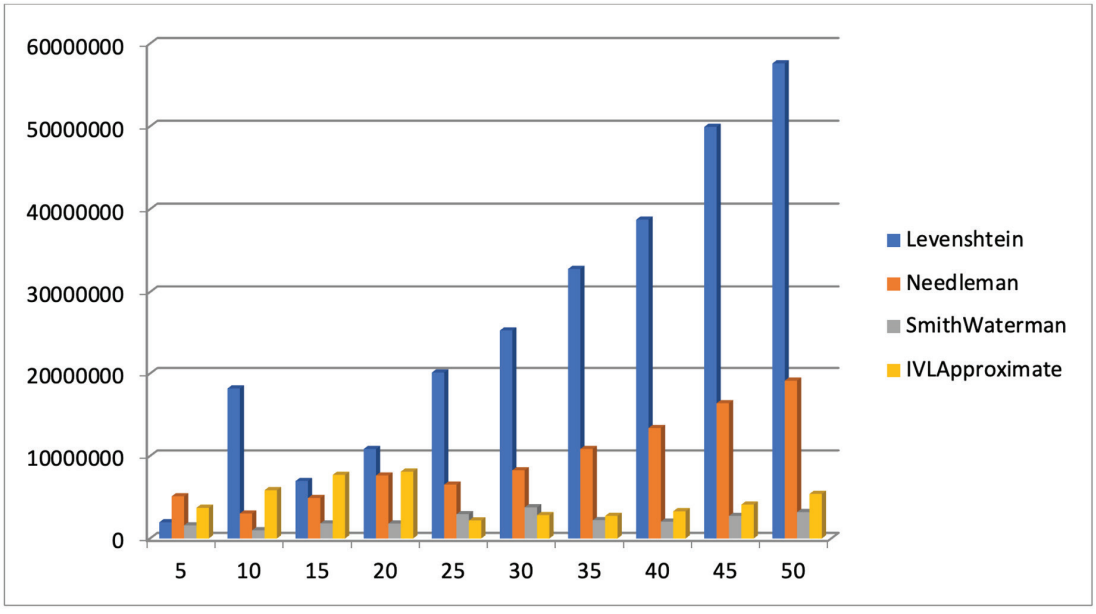
การสุ่มคำอักขระภาษาอังกฤษทั้งตัวพิมพ์เล็กและพิมพ์ใหญ่ และเขียนโปรแกรมเพื่อเปรียบกับขั้นตอนวิธีที่มีประสิทธิภาพได้แก่ Levenshtein, Neesleman และ SmithWaterman โดยสุ่มอักขระที่ความยาว 10 ตัวอักษร และเขียนทดสอบวัดเวลาโดยให้อัตราการเปรียบคู่ไม่ตรงกัน (Distance Ratio) ระหว่าง 0.1-1.0 ของความยาวอักขระ ทดลองแต่ละรายการจำนวน 10 ครั้งหลังจากนั้นหาค่าเฉลี่ย ได้ผลการทดลองดังภาพประกอบ 4 ความเร็วในอัตราการ d (Mismatch) ระหว่าง 0.1 ถึง 1.0 แสดงดังกราฟดังรูปภาพที่ mismatch

เมื่อเปรียบเทียบโดยพิจารณาค่าการอนุญาตให้ผิดพลาดได้ระหว่าง 0.1-1.0 ดังภาพประกอบ 4 แสดงให้เห็นว่าขั้นตอนวิธีใหม่ที่นำเสนอในงานวิจัยนี้ ค้นหาเปรียบเทียบโดยใช้เวลาต่ำกว่าขั้นตอนวิธี Levenshtein และ Neesleman แต่ใช้เวลาค้นหาใกล้เคียงและมากกว่าขั้นตอนวิธี SmithWaterman

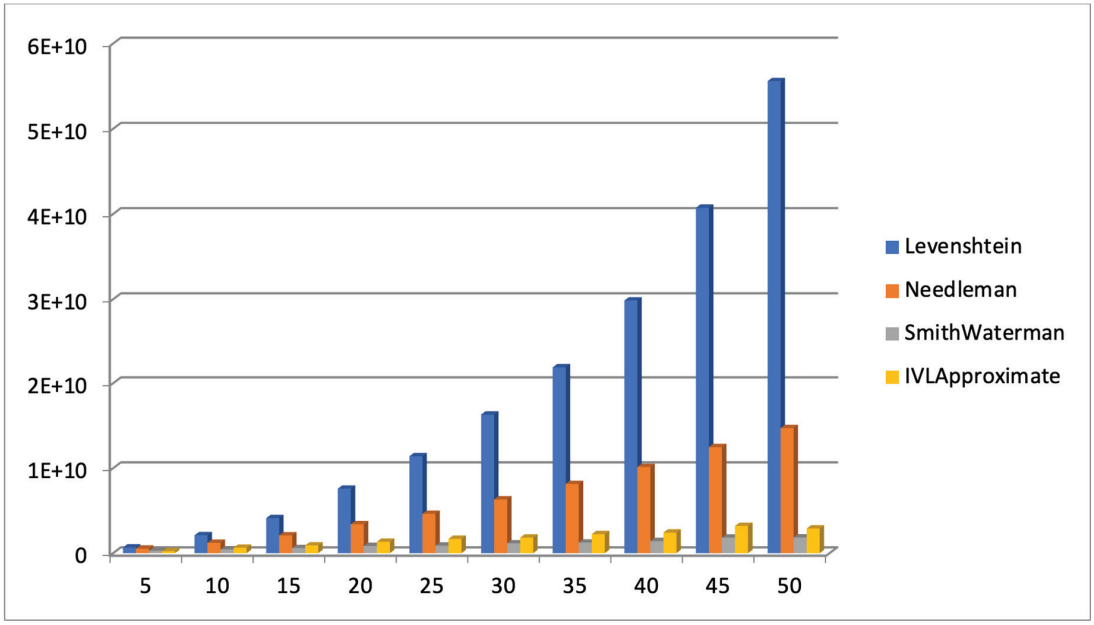
การสุ่มคำอักขระภาษาอังกฤษทั้งตัวพิมพ์เล็กและพิมพ์ใหญ่ จำนวน 1 กิโลไบต์ และเขียนโปรแกรมทดสอบเมื่อกำหนดอักขระแบบที่ได้จากการสุ่มของโปรแกรมคอมพิวเตอร์ และสุ่มอัตราการประมาณ (Distance Ration) โดยวัดจากความยาวอักขระแบบ 5-50 ตัวอักษร เขียนโปรแกรมเพื่อเปรียบกับขั้นตอนวิธีที่มีประสิทธิภาพได้แก่ Levenshtein, Neesleman และ SmithWaterman การทดลองแต่ละรายการจำนวน 10 ครั้งหลังจากนั้นหาค่าเฉลี่ย นำมาเขียนผลการทดลองดังภาพประกอบ 5

จากภาพประกอบ 5 เมื่อค้นหาด้วยการสุ่มข้อมูลทั้งอักขระแบบและข้อความเพื่อค้นหา ยังได้ผลการทดลองที่ ขั้นตอนวิธีใหม่ที่นำเสนอในงานวิจัยนี้ ค้นหาเปรียบเทียบโดยใช้เวลาต่ำกว่าขั้นตอนวิธี Levenshtein และ Neesleman แต่ใช้เวลาค้นหาใกล้เคียงและมากกว่าขั้นตอนวิธี SmithWaterman

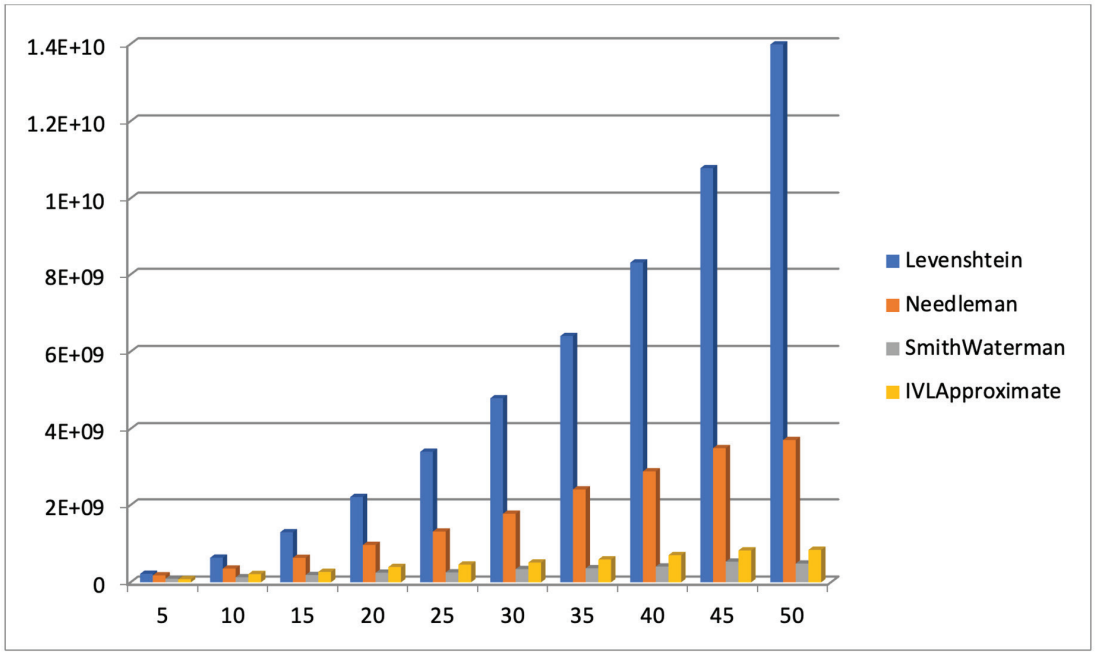
นำเข้าข้อมูลจริงจาก แฟ้มข้อมูลจริง (anage_data.txt---ขนาด 768 KB) ของ The Human Ageing Genomic Resources (HAGR), จาก https://genomics.senescence.info/download.html เขียนโปรแกรมสุ่มอักขระแบบในแฟ้มดังกล่าว



ภาพประกอบ 5 เปรียบความเร็วการค้นหาเมื่อข้อความและอักขระแบบสุ่มจากโปรแกรมคอมพิวเตอร์



ภาพประกอบ 6 เปรียบความเร็วการค้นหาจากข้อมูลจริง
ของ The Human Ageing Genomic Resources (HAGR)



ภาพประกอบ 7 เปรียบความเร็วการค้นหาจากข้อมูลจริงของดีเอ็นเอ

ด้วยความยาวแตกต่างกัน และเปรียบกับขั้นตอนวิธีที่มีประสิทธิภาพได้แก่ Levenshtein, Neesleman และ SmithWaterman การทดลองแต่ละรายการจำนวน 10 ครั้งหลังจากนั้นหาค่าเฉลี่ย นำมาเขียนผลการทดลองดังภาพประกอบ 6

นำเข้าข้อมูล DNA ด้วยแฟ้มข้อมูลจริง (gbbct107.seq---ขนาดต้นฉบับ 240,865 KB) ของ GenBank, National Center for Biotechnology Information, National Library of Medicine, จาก http://biomirror.aarnet.edu.au/biomirror/genbank/GBBCT107.SEQ เขียนโปรแกรมสุ่มอักขระแบบจากแฟ้มดังกล่าวด้วยความยาวแตกต่างกัน และเปรียบกับขั้นตอนวิธีที่มีประสิทธิภาพ ได้แก่ Levenshtein, Neesleman และ SmithWaterman การทดลองแต่ละรายการจำนวน 10 ครั้งหลังจากนั้นหาค่าเฉลี่ย นำมาเขียนผลการทดลองดังภาพประกอบ 7

จากภาพประกอบ 6-7 ยืนยันผลการทดลองค้นหากับข้อมูลจริงที่ได้นำมาเปรียบเทียบ พบว่าแสดงขั้นตอนวิธีใหม่ที่นำเสนอในงานวิจัยนี้ ค้นหาเปรียบเทียบโดยใช้เวลาต่ำกว่าขั้นตอนวิธี Levenshtein และ Neesleman แต่ใช้เวลาค้นหาใกล้เคียงและมากกว่าขั้นตอนวิธี SmithWaterman ซึ่งการทำงานของขั้นตอนวิธีใหม่ ทำงานอยู่ในระดับความเร็วสูงเช่นเดียวกับ ขั้นตอนวิธี SmithWaterman แต่อย่างไรก็ตามหากมีการปรับปรุงให้ดีขึ้น น่าจะทำให้สามารถทำงานได้รวดเร็วกว่าที่แสดงไว้นี้ก็อาจเป็นได้

5. อภิปรายผลและข้อเสนอแนะ

พิจารณาการทำงานในเชิงทฤษฎีของขั้นตอนวิธีที่มีการสร้างโครงสร้างใช้เวลา O(m) ใช้เนื้อที่ เมื่อ m คือ ความยาวของอักขระแบบ คือ จำนวนอักขระแบบ เมื่อนำมาเขียนโปรแกรมทดลองสร้างโครงสร้างพบว่าใช้เนื้อที่บรรจุโครงสร้างในหน่วยความจำคอมพิวเตอร์น้อยกว่าขั้นตอนวิธีที่เคยมีมาก่อนอย่างมีนัยสำคัญ

สำหรับขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณที่พัฒนาขึ้น ค้นหาข้อมูลด้วยกรณีความซับซ้อนมากสุด O((m-k)n) กรณีค้นหาดีที่สุด O(kn) เมื่อ n คือ ความยาวของสายสตริงเท็กซ์ที่ต้องการค้นหา k เมื่อนำมาพัฒนาโปรแกรมคอมพิวเตอร์ค้นหาทั้งข้อมูลการสุ่มและข้อมูลจริงทำให้ได้เห็นผลการทดลองที่ดีกว่าขั้นตอนวิธี Levenshtein, Neesleman ในทุกกรณี ในขณะที่ทำงานได้ใกล้เคียงกับขั้นตอนวิธี SmithWaterman ในบางกรณี ทั้งนี้เนื่องจากระหว่างการค้นหาขั้นตอนวิธีใหม่จะมีดำเนินการหาความแตกต่างและการเปรียบเทียบข้อมูลเพื่อตรวจสอบผลการค้นหาพบในทุกๆ ครั้งของการวนค้นหาในแต่ละหน้าต่างค้นหา แต่ในขั้นตอนวิธี SmithWaterman ดำเนินการเพียงหาค่าความแตกต่างของอักขระแบบเท่านั้น ดังนั้นในกรณีนี้จึงทำให้ขั้นตอนวิธีดังกล่าวมีความรวดเร็วกว่าในการค้นหา อย่างไรก็ตามขั้นตอนวิธีใหม่สามารถพัฒนาต่อยอดเพื่อทำให้มีประสิทธิภาพได้มากยิ่งขึ้นได้ ดังนี้

สามารถพัฒนาขั้นตอนวิธีให้สามารถมีความซับซ้อน O(n) แบบสมบูรณ์ได้โดยการวิเคราะห์รายการผกผันที่เก็บวิเคราะห์ในแต่ครั้งที่การเข้าถึงตารางผกผัน โดยไม่ต้องมีการกวาดซ้ำเหมือนใน Algorithm 2 ได้

สามารถพัฒนาขั้นตอนการค้นหาแบบขนานได้ด้วยการเขียนโปรแกรมแบบใช้เทรดได้อีกด้วย

สามารถพัฒนาการเลื่อนหน้าต่างค้นหาให้มีตารางการเลื่อน (Shift Table) เพื่อใช้เลื่อนหน้าต่างสำหรับการกวาดตรวจการค้นพบ เช่น ขั้นตอนของ Boyer-Moor หรือ KMP จะทำให้มีการค้นหาที่รวดเร็วและมีความซับซ้อนน้อยลงกว่าที่นำเสนอในงานวิจัยนี้ได้

สามารถพัฒนาเป็นขั้นตอนการเปรียบคู่สายอักขระพหุแบบแบบประมาณ (Multiple String Pattern Matching with Allow Error) ได้ด้วยการสร้างตารางรายการผกผันแบบพหุแบบ (Multiple String Pattern)

สามารถนำเอาเทคนิคของปัญญาประดิษฐ์มาร่วมสำหรับการวิเคราะห์หรือทำนายคำเหมือนและแตกต่างได้ในแต่ละพยางค์ของการค้นหาเพื่อเพิ่มประสิทธิภาพให้รวดเร็วขึ้น ในทางตรงกันข้ามหากนำขั้นตอนวิธีที่พัฒนาขึ้นนี้สู่กลไกภายในของปัญญาประดิษฐ์ก็สามารถดำเนินการได้เช่นกัน

6. สรุปผลการวิจัย

บทความงานวิจัยนี้นำเสนอโครงสร้างข้อมูลใหม่เพื่อใช้สำหรับการออกแบบขั้นตอนวิธีเปรียบคู่สายอักขระแบบประมาณ โครงสร้างข้อมูลใหม่พัฒนาต่อยอดจากรายการผกผันแบบเดี่ยวของ Khancome & Boonjing (2010) นำมาออกแบบให้จัดเก็บอักขระแบบอาศัยตารางการแฮชจัดเก็บรายการผกผันจากอักขระแบบเดี่ยว เพื่อให้รองรับการวิเคราะห์ผลการเปรียบคู่สำเร็จได้แบบประมาณที่อนุญาตให้มีค่าความผิดพลาดที่ไม่ตรงกันของตัวอักขระในระหว่างการค้นหาได้ จากนั้นพัฒนาขั้นตอนวิธีการเปรียบคู่สายอักขระแบบประมาณใหม่โดยโครงสร้างข้อมูลชนิดใหม่ที่พัฒนาขึ้น ผลการวิจัยทางทฤษฎีพบว่า ขั้นตอนวิธีที่พัฒนาขึ้นใหม่ มีความซับซ้อนต่ำ สามารถทำงานแบบเชิงเส้นอันเป็นลักษณะของการค้นหาที่มีประสิทธิภาพ โดยการสร้างโครงสร้างข้อมูลใช้เวลาเท่ากับจำนวนอักขระแบบที่นำมาใช้เพื่อค้นหา และความซับซ้อนด้านเนื้อที่จัดเก็บเท่ากับจำนวนอักขระแบบที่ใช้ในอักขระแบบรวมกับขนาดอักขระแบบที่ใช้ ขณะที่ความซับซ้อนด้านเวลาในการค้นหามีค่าดีที่สุดเท่ากับความยาวของข้อความที่ค้นคูณกับความยาวของอักขระแบบ ผลการทดลองด้วยการเขียนโปรแกรมคอมพิวเตอร์วัดผลกับขั้นตอนวิธีที่มีชื่อเสียงพบว่า ขั้นตอนวิธีใหม่ใช้เนื้อที่สำหรับสร้างโครงสร้างด้วยหน่วยความจำที่ต่ำกว่าขั้นตอนวิธีที่นำมาเปรียบเทียบ และสามารถค้นหาได้ดีเป็นลักษณะเชิงเส้นที่ใช้เวลาในการค้นหาได้รวดเร็วเท่าๆ กับขั้นตอนวิธีที่ค้นหาได้รวดเร็วที่สุดที่นำมาเปรียบเทียบ

เอกสารอ้างอิง

Abraham, D., & Raj, N. S. (2014). Approximate string matching algorithm for phishing detection. *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2285–2290. https://doi.org/10.1109/icacci.2014.6968578

Boguszewski, A., Szymański, J., & Draszawka, K. (2016). Towards increasing F-measure of approximate string matching in O(1) complexity. *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), 8*, 527–532. https://doi.org/10.15439/2016f311

Dondi, R., Mauri, G., & Zoppis, I. (2022). On the complexity of approximately matching a string to a directed graph. *Information and Computation, 288*, 104748. https://doi.org/10.1016/j.ic.2021.104748

Faro, S., & Scafiti, S. (2022). A weak approach to suffix automata simulation for exact and approximate string matching. *Theoretical Computer Science, 933*, 88–103. https://doi.org/10.1016/j.tcs.2022.08.028

Gusfield, D. (1997). *Algorithms on strings, trees, and sequences: Computer science and computational biology*. Cambridge University Press.

Karp, R. M., & Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development, 31*(2), 249–260. https://doi.org/10.1147/rd.312.0249

Khan, M. G., Halim, Z., & Baig, A. R. (2023). An efficient approach for faster matching of approximate patterns in graphs. *Knowledge-Based Systems, 276*, 110770. https://doi.org/10.1016/j.knosys.2023.110770

Khancome, C., & Boonjing, V. (2010). Inverted Lists String Matching Algorithms. *International Journal of Computer Theory and Engineering*, 352–357. https://doi.org/10.7763/ijcte.2010.v2.166

Levenshtein, V. (1965). Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission, 1*(1), 8-17.

Levenshtein, V. I. (1966) Binary codes of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady, 10*(8), 707-710.

Navarro, G., & Raffinot, M. (2002). *Flexible pattern matching in strings: Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology, 48*(3), 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

Uhlig, F., Struppek, L., Hintersdorf, D., Göbel, T., Baier, H., & Kersting, K. (2023). Combining AI and AM – Improving approximate matching through transformer networks. *Forensic Science International: Digital Investigation, 45*, 301570