UNIVERSIDAD NACIONAL DE COLOMBIA

# Estudio Comparativo de Algoritmos para el Problema de la Subsecuencia Común Más Larga Restringida

por

**Wilson Eduardo Soto Forero**

Tesis como requisito para optar al grado de
Magíster en Ciencias de la Ingeniería de Sistemas y Computación

Dirigido por:
Yoan Pinzón, Ph.D.

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas e Industrial

Bogotá, Junio 2010

*"Si buscas resultados distintos, no hagas siempre lo mismo."*

Albert Einstein

ALGOS
GRUPO DE ALGORITMOS Y COMBINATORIA
UNIVERSIDAD NACIONAL DE COLOMBIA

# *Resumen*

**Estudio Comparativo de Algoritmos para el Problema de la Subsecuencia Común Más Larga Restringida**

por

Wilson Eduardo Soto Forero

Facultad de Ingeniería

Universidad Nacional de Colombia

Esta tesis describe el problema de la subsecuencia común más larga y específicamente una de sus extensiones, el problema de la subsecuencia común más larga restringida. Esta es una de las operaciones de comparación más importantes sobre las secuencias y su aplicación es diversa en muchas áreas, especialmente en la bioinformática.

Principalmente esta tesis tiene dos contribuciones, *i*) el proponer un nuevo algoritmo para solucionar el problema de la subsecuencia común más larga restringida basado en el concepto de dominancia y *ii*) la realización de una evaluación experimental de los algoritmos más representativos para solucionar el problema de la subsecuencia común más larga restringida incluyéndose el algoritmo propuesto.

El trabajo se complementa con una descripción de las técnicas más usuales para solucionar el problema de la subsecuencia común más larga como base para realizar posteriores trabajos en técnicas no aplicadas para solucionar el problema de la subsecuencia común más larga restringida.

# *Abstract*

**Comparative Study of Algorithms for the Constrained Longest Common Subsequence Problem**

by

Wilson Eduardo Soto Forero

Faculty of Engineering

National University of Colombia

This thesis focuses on the longest common subsequence problem. Specifically, it focuses on one of its extensions, the constrained longest common subsequence problem (CLCS). The CLCS problem is one of the most important comparison operations on sequences, where its application is diverse in many areas, especially in bioinformatics.

This thesis has two main contributions, *i*) To propose a new algorithm to solve the constrained longest common subsequence problem based on the concept of dominance, and *ii*) To perform an experimental evaluation of the most representative CLCS algorithms including the proposed algorithm.

The work is complemented with a description of the state of the art techniques to solve the longest common subsequence problem. This description will be the basis of further work, which will pretend the development of novel techniques to solve the constrained longest common subsequence problem.

# *Agradecimientos*

Inmensa gratitud a mi director Yoan Pinzón por sus consejos, serenidad y colaboración durante mi proceso de investigación. Reconocerle a él mi incursión en un tema muy interesante y significativo, no solamente sus aportes fueron valiosos en el área académica sino también lo fueron en mi vida personal.

Quiero también agradecer al Laboratorio de Investigación en Sistemas Inteligentes (LISI), allí trabajan personas admirables las cuales gracias a sus discusiones e ideas contribuyeron en mi proceso académico y más que compañeros son un grupo de amigos.

También quiero mencionar a los docentes que tuve el placer de conocer y hacen parte del Departamento de Ingeniería de Sistemas e Industrial, ellos me aportaron cosas muy útiles para mi formación como futuro investigador.

Durante todo este tiempo de mi maestría agradezco de corazón a mi familia por su apoyo y soporte, e igualmente a una persona muy especial.

# Contenido

**Resumen** ii

**Abstract** iii

**Agradecimientos** iv

**Contenido** v

**Índice de Figuras** vii

**Índice de Tablas** x

**Índice de Algoritmos** xi

**1 Introducción** 1

**2 Preliminares** 3
- 2.1 Comparación de secuencias . . . 4
- 2.2 El problema de la subsecuencia común más larga . . . 5
- 2.3 La subsecuencia común más larga como distancia . . . 6
- 2.4 El problema de la subsecuencia común más larga restringida . . . 7
- 2.5 Alineamiento de un par de secuencias con restricción . . . 8

**3 Técnicas de solución de la subsecuencia común más larga** 10
- 3.1 Programación dinámica . . . 10
- 3.2 Autómatas finitos . . . 12
- 3.3 Programación dinámica escasa . . . 13
- 3.4 Divide y vencerás . . . 15
- 3.5 Paralelismo de bits . . . 17

**4 Algoritmos para solucionar el problema de la subsecuencia común más larga restringida** 23
- 4.1 Algoritmo de Yin–Te Tsai . . . 24
- 4.2 Algoritmo de Peng Chao–Li . . . 25
- 4.3 Algoritmo de Chin *et al.* . . . 31
- 4.4 Algoritmo de Arslan y Eğecioğlu . . . 32
- 4.5 Algoritmo de Wang . . . 34

4.6 Algoritmo de Deorowicz . . . . . 38
4.7 Algoritmo de Iliopoulos y Rahman . . . . . 39

**5 Nuevo Algoritmo** **46**
5.1 Algoritmo . . . . . 46
5.2 Ejemplos . . . . . 50
5.3 Complejidad . . . . . 53

**6 Experimentación** **54**
6.1 Procedimiento . . . . . 55
6.2 Resultados . . . . . 56
6.2.1 Resultados con secuencias generadas de forma aleatoria . . . . . 57
6.2.2 Resultados con secuencias de una colección de textos . . . . . 60

**7 Conclusiones y Trabajo Futuro** **65**

**A Anexos** **67**

**Bibliografía** **69**

# Índice de Figuras


2.1 LCS para la secuencia "BIOLOGÍA" contra otras secuencias. Los números representan el LCS entre las secuencias, además, para cada par de secuencias se muestra un posible alineamiento. Se usa el carácter '-' como carácter vacío. . . . . 6

2.2 CLCS para la secuencia "GENÉTICA" contra otras secuencias con la secuencia de restricción "TA". Los números representan el CLCS entre las secuencias, además, para cada par de secuencias se muestra un posible alineamiento (enmarcado el LCS). Se usa el carácter '-' como carácter vacío. . . . . 8

3.1 Construcción de la matriz $L$ con la técnica de programación dinámica para encontrar el LCS para las secuencias $x =$ "ACCAAG" y $y =$ "TCCACA". Las líneas sobresalientes enmarcan los límites y los números encerrados en círculos los puntos dominantes para cada límite. $\mathcal{L}(x, y) = 4$. LCS$(x, y)$ = "CCAA". . . . . 11

3.2 Grafo para las secuencias $x =$ "ACCAAG" y $y =$ "TCCACA". La línea sobresaliente es la ruta más corta desde la parte superior izquierda (posición $(0, 0)$ en la matriz de programación dinámica) al nodo ubicado en la parte inferior derecha (posición $(m, n)$ en la matriz de programación dinámica). Los arcos diagonales representan coincidencias. $\mathcal{L}(x, y) = 4$. . . . . 13

3.3 Matriz de programación dinámica para las secuencias $x =$"ACCAAG" y $y =$ "TCCACA", se muestran las anticadenas (líneas sobresalientes de color gris) para cada $k$. Además, el cálculo de $THRESH$ y $MATCHLIST$ para las secuencias dadas, $x$ y $y$. En $THRESH$ se puede observar el $\mathcal{L}(x, y)$, como el valor de $k$ donde $i = m$. En este caso, $\mathcal{L}(x, y) = 4$, porque el valor de $k$ para $i = m$ es 4. . . . . 15

3.4 Representación de la técnica de divide y vencerás (izquierda) y los llamados recursivos (derecha). . . . . 17

3.5 Las matrices representan el cálculo del punto medio de una "curva" del LCS para las secuencias $x =$"ACCAAG" y $y =$ "TCCACA" usando el algoritmo de Hirschberg. El árbol representa la técnica de divide y vencerás usando el mismo algoritmo y las secuencias dadas $x$ y $y$. Las letras dentro de los círculos representan el LCS de las secuencias $x$ y $y$, es decir, LCS$(x, y) =$ "CCAA". . . . . 18

3.6 Construcción de las matrices $V$', $M$ y $M$' para encontrar el LCS con la técnica vectores de bits dadas las secuencias $x =$ "ACCAAG" y $y =$ "TCCACA". $\mathcal{L}(x, y) = 4$. . . . . 20

3.7 Construcción de las matrices (a.) $V'$ y (b.) $K$ para recuperar el LCS con la técnica vectores de bits dadas las secuencias $x$ = "ACCAAG" y $y$ = "TCCACA". En (c.) se presentan las coincidencias, las coincidencias dominantes y las coincidencias LCS del ejemplo. Además, se encuentran los valores obtenidos para $k$ y $l$ con su respectiva representación gráfica ($k$ línea punteada y $l$ línea continua). El LCS recuperado son los caracteres correspondientes a las intersecciones donde ocurren los cuadros de color gris. Entonces, el LCS$(x, y)$ = "CCAA". . . . . 22

4.1 Representación del algoritmo de Tsai para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". Los círculos de color negro representan coincidencias en la etapa actual y los de color blanco en etapas anteriores. Dentro de los círculos negros aparecen los LCS de las correspondientes secuencias (límites de las áreas de color gris). El símbolo '-' representa $-\infty$. La primera tabla corresponde a $k = 1$ y la segunda tabla a $k = 2$ . . . . . 26

4.2 Representación del algoritmo de Tsai para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". Los círculos de color negro representan coincidencias en la etapa actual y los de color blanco en etapas anteriores. Dentro de los círculos negros aparecen los LCS de las correspondientes secuencias (límites de las áreas de color gris). El símbolo '-' representa $-\infty$. La primera tabla corresponde a $k = 3$ y la segunda tabla al resultado obtenido $\mathcal{L}_z(x, y) = 5$. . . . 27

4.3 Dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". En la tabla superior se presenta $k = 0$ donde los círculos de color blanco representan el LCS de las secuencias. Los círculos de color gris representan el CLCS de las secuencias dadas. La tabla inferior representa el $k = 1$. . . . . 29

4.4 Dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". En la tabla superior se presenta $k = 2$ y la tabla inferior representa el $k = 3$. Los círculos de color gris representan el CLCS de las secuencias dadas. $\mathcal{L}_z(x, y) = 5$. . . . . 30

4.5 En $(a.)$ se representa el algoritmo propuesto por Chin *et al.* El algoritmo de Chin *et al.* plantea una complejidad de tiempo igual al algoritmo de Peng. En $(b.)$ se representa el uso del algoritmo de Hirschberg (véase Sección 3.4) en la solución del problema del CLCS. . . . . 32

4.6 Dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". En cada celda existe una lista de valores. Se usa ( – ) para mostrar el índice en $z$ y la correspondiente longitud de la subsecuencia común más larga restringida. La celda $L(m, n)$ contiene el $\mathcal{L}_z(x, y)$ en el último valor de la lista de esta celda, para el ejemplo es (3–5), es decir, $|z| = 3$ y $\mathcal{L}_z(x, y) = 5$. . . . . 37

4.7 Representación del algoritmo de Deorowicz para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". Los números representan el cálculo sobre las celdas que son coincidencias y tenidas en cuenta para calcular el CLCS de las secuencias dadas. Las celdas con el símbolo '-' representan $-\infty$ y las celdas sin símbolo alguno no existen. La primera tabla corresponde a $k = 0$ y la segunda tabla a $k = 1$ . . . . . 41

4.8 Representación del algoritmo de Deorowicz para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ = "GTA". Los números representan el cálculo sobre las celdas que son coincidencias y tenidas en cuenta para calcular el CLCS de las secuencias dadas. Las celdas con el símbolo '-' representan $-\infty$ y las celdas sin símbolo alguno no existen. La primera tabla corresponde a $k = 2$ y la segunda tabla a $k = 3$ . . . . . 42
4.9 Representación del funcionamiento del algoritmo de Iliopoulos y Rahman sobre las secuencias $x$ = "ATTGCAGACGGTCAC", $y$ = "GTCCAGCAACGTCCC" y $z$ ="GTA" para $k = 1$. Las áreas de color gris representan las estructuras $BoundedHeap$ utilizadas por el algoritmo. Enmarcado aparece el valor a calcular (posición $L_1(4, 11)$). Las flechas representan la región donde se evalúa la función Maximovalor en las estructuras. . . . . 44

5.1 Ejemplo 5.1, 5.2 y 5.3.($i$) una matriz de representación de las secuencias $x$ y $y$, ($ii$) resultado de las subsecuencias comunes de tamaño 1, ($iii$) el CLCS reportado por el algoritmo propuesto, ($iv$) ruta de cálculo del paso 1. 48
5.2 Ruta de cálculo de Ejemplo 5.1. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia. . . . . 50
5.3 Ruta de cálculo de Ejemplo 5.2. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia. El CLCS encontrado es señalado por el visto. . . . . 51
5.4 Ruta de cálculo de Ejemplo 5.3. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia. El CLCS encontrado es señalado por el visto. . . . . 52

6.1 Resumen de los valores utilizados por las variables en la experimentación. 56
6.2 Comportamiento de los algoritmos variando el tamaño del alfabeto ($\sigma$), con las variables fijas $r = 16$ y (a.) $m = n = 100$, (b.) $m = n = 300$, (c.) $m = n = 500$ y (d.) $m = n = 1000$ 58
6.3 Comportamiento de los algoritmos variando el tamaño de la secuencia de restricción ($r$), con las variables fijas $\sigma = 20$ y (a.) $m = n = 100$, (b.) $m = n = 300$, (c.) $m = n = 500$ y (d.) $m = n = 1000$ . . . . . 59
6.4 Comportamiento de los algoritmos variando el tamaño de las secuencias $x$ y $y$ ($m = n$), con las variables fijas $\sigma = 20$ y (a.) $r = 8$, (b.) $r = 16$, (c.) $r = 24$, (d.) $r = 64$ . . . . . 60
6.5 Comportamiento de los algoritmos variando el tamaño de la secuencia $y$ ($n$), con las variables fijas $m = 1000$, $r = 64$ y (a.) $\sigma = 8$, (b.) $\sigma = 20$, (c.) $\sigma = 32$, (d.) $\sigma = 64$ . . . 61
6.6 Comportamiento de los algoritmos variando el tamaño del alfabeto ($\sigma$), con las variables fijas $r = 16$ y (a.) $m = n = 100$, (b.) $m = n = 300$, (c.) $m = n = 500$ y (d.) $m = n = 1000$ 62
6.7 Comportamiento de los algoritmos variando el tamaño de la secuencia de restricción ($r$), con las variables fijas $\sigma = 215$ y (a.) $m = n = 100$, (b.) $m = n = 300$, (c.) $m = n = 500$ y (d.) $m = n = 1000$ . . . . . 63
6.8 Comportamiento de los algoritmos variando el tamaño de las secuencias $x$ y $y$ ($m = n$), con las variables fijas $\sigma = 215$ y (a.) $r = 8$, (b.) $r = 16$, (c.) $r = 24$, (d.) $r = 64$ . . . . 63
6.9 Comparación de los algoritmos seleccionados variando el tamaño de la secuencia $y$ ($n$), con las variables fijas $m = 1000$, $r = 64$ y (a.) $\sigma = 16$, (b.) $\sigma = 25$, (c.) $\sigma = 133$, (d.) $\sigma = 215$ . . . . . 64

# Índice de Tablas

4.1 Complejidad de los algoritmos para solucionar el CLCS . . . . . 23

6.1 Nombre de los archivos C++ . . . . . 55
6.2 Colección de textos . . . . . 56

A.1 Entropía promedio de las secuencias para la Figura 6.6 . . . . . 67
A.2 Entropía promedio de las secuencias para la Figura 6.7 . . . . . 67
A.3 Entropía promedio de las secuencias para la Figura 6.8 . . . . . 67
A.4 Entropía promedio de las secuencias para la Figura 6.9 . . . . . 68

# Índice de Algoritmos


3.1 Algoritmo de Hunt–Szymanski para recuperar el LCS . . . . . 16
3.2 Algoritmo FindRow (Algoritmo de Hirschberg) . . . . . 19
3.3 Algoritmo de Hirschberg para recuperar el LCS . . . . . 19
3.4 Algoritmo de vectores de bits para recuperar el LCS . . . . . 21
4.1 Algoritmo de Tsai para encontrar el CLCS . . . . . 25
4.2 Algoritmo de Peng para encontrar el CLCS . . . . . 31
4.3 Algoritmo de Chin *et al.* para encontrar el CLCS . . . . . 33
4.4 Algoritmo $f_1$ (Algoritmo de Arslan) . . . . . 34
4.5 Algoritmo $f_2$ (Algoritmo de Arslan) . . . . . 34
4.6 Algoritmo de Arslan para encontrar el CLCS . . . . . 34
4.7 Algoritmo $Caso_1$ (Algoritmo de Wang) . . . . . 35
4.8 Algoritmo $Caso_2$ (Algoritmo de Wang) . . . . . 35
4.9 Algoritmo $RCLCS$ (Algoritmo de Wang) . . . . . 36
4.10 Algoritmo de Wang para encontrar el CLCS . . . . . 36
4.11 Algoritmo de Deorowicz para encontrar el CLCS . . . . . 40
4.12 Algoritmo de Iliopoulos y Rahman para encontrar el CLCS . . . . . 43
5.1 Nuevo Algoritmo CLCS . . . . . 49

# *Declaración de Autoría*

Parte del material presentado en este trabajo ha sido publicado como artículos de conferencia y artículos de revista.

- ✓ An Algorithm for Constrained LCS. En *proceedings of ACS/IEEE International Conference on Computer Systems and Applications AICCSA 2010 – Algorithms and Bioinformatics*. IEEE. Hammamet, Túnez. Mayo 2010. ISBN: 978–1–4244–7715–9.
- ✓ Resultados experimentales de algoritmos para solucionar el problema de la Subsecuencia Común Más Larga Restringida. En *memorias del V Congreso Colombiano de Computación 5CCC – Computación teórica y matemática de la computación*. Cartagena, Colombia. Abril 2010. ISBN: 978–958–8387–40–6.
- ✓ Una revisión de las técnicas de solución al problema de la subsecuencia común más larga. En *memorias del Encuentro Nacional de Investigación en Posgrados (ENIP 2009)*, Universidad Nacional de Colombia, Bogotá. Diciembre 2009. ISBN: 978–958–719–374–9.
- ✓ Análisis algorítmico de la subsecuencia común más larga como medida de similitud entre secuencias genómicas para construcción de árboles filogenéticos. En *III Congreso Colombiano de Biotecnología y II Seminario Internacional de Bionegocios*, CCB Bogotá, Colombia. 2008.
- ✓ Sobre el Longest Common Subsequence: Algoritmos y Extensiones. *Revista Colombiana de Computación*, 8(2): 79 – 100, (2007). ISSN 1657–2831.
- ✓ El Problema del Longest Common Subsequence: Algoritmos y Extensiones. En *Memorias del Seminario Internacional de Informática y Computación (SIIC 2007)*, UPB Bucaramanga, Colombia. Septiembre 2007. Boletín informático No. 2 Facultad de Ingeniería Informática, Octubre – Noviembre, págs. 20 – 40.

En Adición, los siguientes artículos no fueron incluidos en este trabajo pero se realizaron en el transcurso de su elaboración:

- ✓ Optimización basada en un algoritmo de colonia de hormigas adaptativo para el problema de la mochila múltiple. En *memorias del V Congreso Colombiano*

*de Computación 5CCC – Computación teórica y matemática de la computación.* Cartagena, Colombia. Abril 2010. ISBN: 978–958–8387–40–6.

- ✓ Classification of DNA Sequences Using Gap–Weighted Subsequences Kernel. En *memorias del XV Congreso Argentino de Ciencias de la Computación CACIC 2009 – WASI Workshop en Agentes y Sistemas Inteligentes.* Jujuy, Argentina. Octubre 2009. ISBN: 978–897–24068–4–1, págs. 100 – 108.

- ✓ Una Metaheurística Híbrida Aplicada a CVRP. En *memorias del IV Congreso Colombiano de Computación 4CCC – Computación teórica y matemática de la computación.* Bucaramanga, Colombia. Abril 2009. ISBN: 978–958–8166–43–8.

- ✓ Una Metaheurística Híbrida Aplicada a un Problema de Planificación de Rutas. *Revista Avances en Sistemas e Informática* 5(3): 135 – 145, 2008. ISSN: 1657–7663.

# 1

# Introducción

Las secuencias de símbolos o caracteres (*strings* en inglés) son una forma natural para representar información. Estas secuencias son muy importantes en áreas como la biología molecular, la representación de imágenes, el procesamiento de texto y la transmisión de información. Una secuencia es una cadena finita donde todos sus elementos pertenecen a un alfabeto, *e.g.*, en biología molecular una secuencia de ADN está formada por los caracteres pertenecientes al alfabeto $\{\texttt{A,C,G,T}\}$; cada carácter representa una base nitrogenada: `A` = Adenina, `C` = Citosina, `G` = Guanina y `T` = Tiamina.

Uno de los problemas más frecuentes sobre cadenas de caracteres es la comparación de secuencias. La comparación de secuencias sirve para determinar las similitudes y las diferencias de dos o más cadenas de caracteres. En bioinformática, el alineamiento de secuencias y encontrar la subsecuencia común más larga son algunos de los métodos más usados en la comparación de secuencias.

Uno de los principales métodos usados para comparar y determinar la similitud entre secuencias, es hallar la subsecuencia común más larga – *Longest Common Subsequence* (LCS) – entre dos secuencias dadas, es decir, la subsecuencia común (sucesión de caracteres de forma ordenada y no necesariamente contiguos) de mayor longitud. Se han desarrollado para este problema diversas técnicas de solución. Adicional, existen un gran número de extensiones del problema que de igual manera tienen una aplicación [7, 9, 19, 22, 23]. Una de estas extensiones es el encontrar la subsecuencia común más larga restringida – *Constrained Longest Common Subsequence* (CLCS) –, es decir, buscar la subsecuencia común más larga que contiene una secuencia específica o secuencia

de restricción. Los trabajos desarrollados hasta el momento para solucionar el CLCS [2, 8, 13, 14, 20, 30] están basados en la técnica de la programación dinámica.

Debido a la gran utilidad y relevancia en determinar la similitud o semejanza de las secuencias, sumado a la búsqueda en la reducción de recursos computacionales para estos tipos de métodos, este trabajo plantea un estudio comparativo de los algoritmos para el problema de la subsecuencia común más larga restringida (CLCS).

La motivación del estudio se debe a la falta de implementaciones sobre los algoritmos sobre el problema de la subsecuencia común más larga restringida (CLCS), o bien, estas implementaciones han sido realizadas con diversos estilos de programación afectando los resultados, por ello, se espera desarrollar una implementación unificada de estos algoritmos y observar su comportamiento en algunos casos específicos con el fin de indicar sus ventajas y desventajas.

Además del objetivo principal, se incluye en este trabajo un nuevo algoritmo para el problema de la subsecuencia común más larga (CLCS) como parte del estudio realizado sobre el tema. El algoritmo propuesto también hace parte del estudio experimental del comportamiento de los algoritmos.

Para poder leer esta tesis se explican los conceptos básicos en el capítulo 2. Esta tesis principalmente se divide en dos partes; la primera parte es una revisión de las técnicas utilizadas para el problema de la subsecuencia común más larga (LCS) y la segunda parte abarca la explicación, implementación y experimentación de diversos algoritmos desarrollados para el problema de la subsecuencia común más larga restringida (CLCS).

La primera parte está contenida en el capítulo 3. El problema de la subsecuencia común más larga restringida (CLCS) es una variante del problema clásico de la subsecuencia común más larga (LCS), por eso se hace necesario presentar una visión global de las diversas técnicas computacionales aplicadas para solucionar este problema clásico.

La segunda parte del documento aborda el capítulo 4 con la explicación detallada de los diversos algoritmos para el problema de la subsecuencia común más larga restringida (CLCS). El capítulo 5 presenta un nuevo algoritmo para el problema de la subsecuencia común más larga restringida (CLCS). El capítulo 6 muestra los resultados experimentales de los algoritmos implementados y las recomendaciones para cada caso particular. El capítulo 7 presenta las conclusiones del trabajo y el final de la tesis.

# 2

# Preliminares

Una secuencia (*string* en inglés) es una sucesión finita de símbolos los cuales pertenecen a un alfabeto. Un alfabeto es un conjunto finito de símbolos denotado por $\Sigma$ y su tamaño o número de símbolos por $\sigma$. Una secuencia $x$ de longitud $m$ se representa $x_1 \dots x_m$, donde $x_i \in \Sigma$ para $1 \leq i \leq m$. La secuencia sin símbolos es denotada por $\varepsilon$. La longitud de una secuencia se representa por $|S|$ (*e.g.* $|\varepsilon|= 0$) o cuando las secuencias de entrada son conocidas, $m$ para la secuencia $x$, $n$ para la secuencia $y$ y $r$ para el patrón $z$ o secuencia de restricción. Una secuencia invertida de una secuencia $x = x_1 x_2 \dots x_m$, se representa por $x^*$ y se define como $x^* = x_m x_{m-1} \dots x_1$. Una subsecuencia es cualquier subconjunto de elementos de una secuencia conservando el mismo orden relativo, así una subsecuencia $s_1 \dots s_k$ de $x$ es obtenida eliminando $m- |s|$ símbolos de $x$, también se dice $s$ está contenida en $x$. Una subsecuencia común – *Common Subsequence* (CS) – de $x$ y $y$ es una subsecuencia que se encuentra en $x$ y en $y$. La subsecuencia común más larga – *Longest Common Subsequence* (LCS) – de $x$ y $y$, denotado por $\text{LCS}(x, y)$ es una subsecuencia común de máxima longitud. La longitud del $\text{LCS}(x, y)$ se denota por $\mathcal{L}(x, y)$. La subsecuencia común más larga restringida – *Constrained Longest Common Subsequence* (CLCS) – de las secuencias $x$, $y$ y una secuencia de restricción $z$, se da, si y sólo si, $z$ es una subsecuencia de $\text{LCS}(x, y)$. Se denota el CLCS de las secuencias $x, y$ y $z$ como $\text{LCS}_z(x, y)$ y su longitud por $\mathcal{L}_z(x, y)$.

La distancia entre dos secuencias $x$ y $y$ es el mínimo costo de operaciones en transformar $x$ a $y$. El costo de operaciones de una secuencia es la suma de los costo de las operaciones individuales (inserción, eliminación o sustitución). La operación de inserción sobre la secuencia $x =$ "`TCA`" adicionando la letra "`G`" produce $x =$ "`TCGA`", la operación de eliminación sobre la secuencia $x =$ "`TCGA`" eliminando la letra "`T`" produce $x =$ "`CGA`"

y la operación de sustitución sobre la secuencia $x =$ "`GATC`" reemplazando la letra "`A`" por "`T`" es $x =$ "`GTTC`". Un prefijo de $s$ es una secuencia $v$ tal que $s = vt$ para algunos $t \in \varepsilon$. Un sufijo de $s$ es una secuencia $v$ tal que $s = tv$ para algunos $t \in \varepsilon$.

Alineamiento significa insertar espacios de forma que letras iguales se alineen. En esta operación se debe evitar la alineación de los espacios, pero estos son aceptados al comienzo o al final de las secuencias *e.g.*, el alineamiento para las secuencias "`ACCAAGC`" y "`TCCACAC`" donde los espacios son representados por "`-`" es:

```
- T C C A C A - C
A - C C A - A G C
```

Algunas de las técnicas usadas para la comparación de secuencias se realizan por medio de la comparación de sus caracteres o símbolos. Por lo tanto, si una pareja ordenada de posiciones $i$ y $j$ de la forma $(i, j)$ donde $x_i$ y $y_j$ son iguales se conoce como coincidencia (*match* en inglés). El número total de coincidencias entre $x$ y $y$ se representa por $\mathcal{M}$. Si $(i, j)$ es una coincidencia y si $\mathcal{L}(x_{1..i}, y_{1..j})$ tiene una longitud $k$, entonces $k$ es el rango de $(i, j)$. Una coincidencia $(i, j)$ es $k$–dominante si tiene rango $k$ y para alguna otra pareja $(i', j')$ de rango $k$, cualquier $i' > i$ y $j' \leq j$ o $i' \leq i$ y $j' > j$. El número total de puntos dominantes (todos los rangos) se simboliza como $\mathcal{Q}$.

Se denota $\mathcal{R}$ como un orden de relación parcial sobre el conjunto de coincidencias $\mathcal{M}$: una coincidencia $(i, j)$ precede a una coincidencia $(i', j')$ si $i < i'$ y $j < j'$. Un conjunto de coincidencias tales que en alguna de las parejas de las coincidencias siempre precede otra en $\mathcal{R}$ constituye una *cadena* relativa a el orden parcial de la relación $\mathcal{R}$. Un conjunto de coincidencias tal que en alguna pareja ningún elemento de la pareja precede a la otra en $\mathcal{R}$ constituye una *anticadena*.

## 2.1 Comparación de secuencias

La comparación de secuencias es una importante operación que aparece en diversas áreas, como la biología molecular, ciencias de la computación y teoría de la codificación. Uno de los principales factores en la comparación de secuencias es establecer como dos secuencias se parecen o son similares. Existen varios métodos para calcular esta cantidad

entre dos secuencias, entre los cuales están los alineamientos y la subsecuencia común más larga.

En bioinformática existen dos tipos de alineamientos importantes: el alineamiento global y el alineamiento local. El alineamiento global intenta alinear las secuencias ocupando la longitud total de las secuencias dadas. Esto es útil cuando las secuencias a alinear son altamente similares, por ejemplo, secuencias de proteínas provenientes de una misma familia o son muy conservadas. Para obtener este alineamiento, es común usar el algoritmo Needleman–Wunsch [24]. Sin embargo, es más común en las aplicaciones biológicas, utilizar el algoritmo Smith–Waterman [29], alineamiento en el cual las secuencias a alinear contienen grandes diferencias pero se sospecha la existencia de regiones de similitud, es decir, un alineamiento local [9, 21].

En el análisis de secuencias biológicas la similitud de secuencias es uno de los conceptos de mayor importancia, principalmente por el hecho de observar las posibles relaciones de evolución compartidas por los organismos; esta cantidad puede ser expresada como un porcentaje de identidad útil para determinar las homologías entre secuencias, la cual puede, por ejemplo, precisar si comparten un mismo ancestro. Este porcentaje de identidad es una tasa entre el número de columnas con caracteres idénticos encontrados en un alineamiento y el número de símbolos de la secuencia más larga. Otros conceptos biológicos en el cual la similitud de secuencias aplica, es la detección de patrones en secuencias y en la construcción de árboles filogenéticos, los cuales muestran las posibles relaciones y diferencias entre los organismos [9, 26].

## 2.2 El problema de la subsecuencia común más larga

Una de las medidas de similitud que se puede extraer a dos secuencias, consiste en recuperar el mayor número de caracteres coincidentes apareciendo de forma ordenada y no necesariamente consecutiva de ambas secuencias, concepto conocido como la subsecuencia común más larga.

**Definición 2.1 (*Subsecuencia común más larga*).** – en inglés *Longest Common Subsequence* (LCS) – Sean las secuencias $x$ y $y$ que pertenecen a $\Sigma$, y con longitud $m$ y $n$ ($m \geq n$) respectivamente, el LCS de las secuencias $x$ y $y$ consiste en encontrar una subsecuencia de $x$ y de $y$ de máxima longitud posible.

Se deduce entonces como el LCS$(x, y)$ es una subsecuencia de $x$ y $y$ e igualmente, como $x$ y $y$ son supersecuencias de LCS$(x, y)$. El LCS no permite la operación de sustitución, únicamente las operaciones de inserción y eliminación [23].

**Ejemplo 2.1.** El LCS entre la secuencia "`BIOLOGÍA`" y algunas otras secuencias o palabras del idioma español se puede observar en la Figura 2.1.

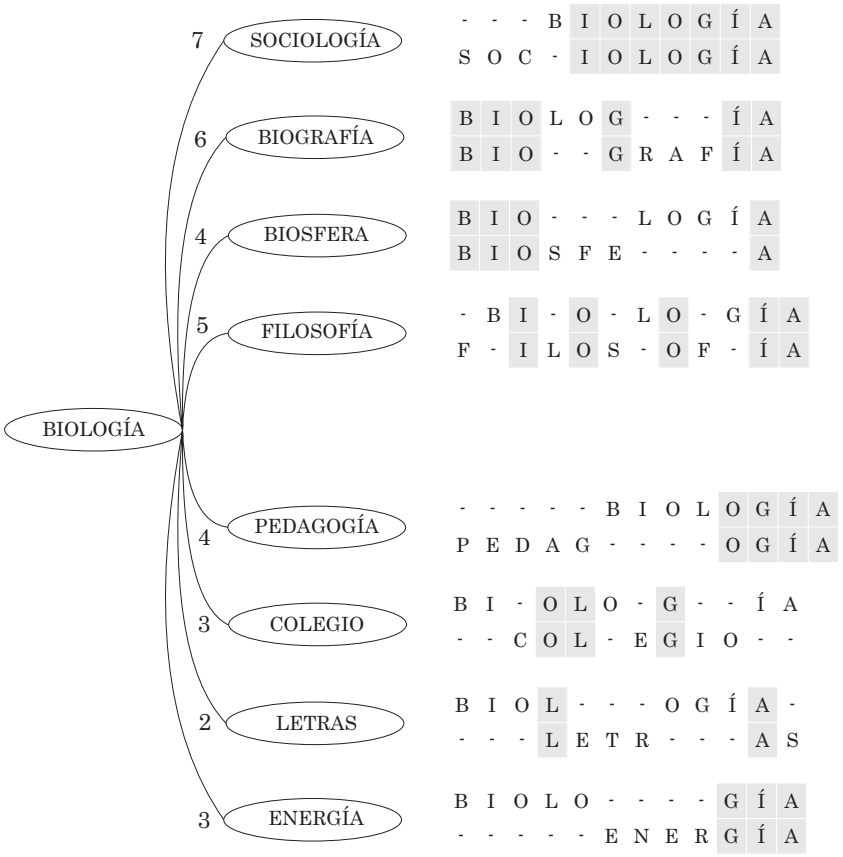


FIGURA 2.1: LCS para la secuencia "`BIOLOGÍA`" contra otras secuencias. Los números representan el LCS entre las secuencias, además, para cada par de secuencias se muestra un posible alineamiento. Se usa el carácter '-' como carácter vacío.

## 2.3 La subsecuencia común más larga como distancia

Si las secuencias son representadas como puntos en un espacio métrico, entonces una medida de distancia es una función asociada a un valor numérico con un par de secuencias determinando a una mayor distancia menor similitud y viceversa. Las medidas de distancia usualmente satisfacen los axiomas matemáticos de una métrica.

Si $M$ es un conjunto y $f$ una función $M \times M \rightarrow \Re^{+}$, $f$ es una métrica sobre $M$ si para todo $x, y, z \in M$ los siguientes axiomas se satisfacen:

$$\forall x,\ y,\ f(x,y) = 0 \Longleftrightarrow x = y \tag{2.1}$$

$$\forall x \neq y,\ f(x,y) > 0 \text{ [Positiva]} \tag{2.2}$$

$$\forall x,\ y,\ f(x,y) = f(y,x) \text{ [Simétrica]} \tag{2.3}$$

$$\forall x,\ y,\ z,\ f(x,y) \leq f(x,z) + f(z,y) \text{ [Inigualdad triangular]} \tag{2.4}$$

Si $x, y \in \Sigma$ y $d(x,y)$ es la distancia entre las dos secuencias que satisfacen los axiomas de la métrica, se define $d_{\mathcal{L}}(x,y)$ como la longitud de la subsecuencia común de máxima tamaño y considera $0 \leq d_{\mathcal{L}}(x,y) \leq |x|$ [25, 28].

Si el mínimo número de operaciones de inserción y eliminación necesarios para transformar $x$ en $y$, puede expresarse:

$$d(x,y) = m + n - 2 \times d_{\mathcal{L}}(x,y) \tag{2.5}$$

Entonces, la distancia de la subsecuencia común más larga de las secuencias $x$ y $y$ es:

$$d_{\mathcal{L}}(x,y) \ = \ \frac{m + n - d(x,y)}{2} \tag{2.6}$$

Se puede deducir como a mayor valor del LCS las secuencias son más similares.

**Ejemplo 2.2.** Sea $x$ =“`ATCTGAT`” y $y$ =“`TGCATA`”, entonces $d(x,y) = m + n - 2 \times d_{\mathcal{L}}(x,y) = 7 + 6 - 2 \times 4 = 5$

$$\begin{bmatrix} \texttt{A T - C - T G A T} \\ \texttt{- T G C A T - A -} \end{bmatrix} \text{ y } \begin{bmatrix} \texttt{A T C T G - A T -} \\ \texttt{- T - - G C A T A} \end{bmatrix}$$

En el primer alineamiento se puede observar el LCS$(x,y)$ =“`TCTA`” y en el segundo alineamiento el LCS$(x,y)$ =“`TGAT`” haciendo referencia a las coincidencias respectivas.

## 2.4 El problema de la subsecuencia común más larga restringida

El LCS no siempre es satisfactorio porque existen caracteres que están en las secuencias dadas y no aparecen en las subsecuencias comunes de máxima longitud. Entonces, para

hallar esta medida de similitud de manera apropiada, se adiciona una restricción forzando a las subsecuencias comunes de máxima longitud resultantes contengan los caracteres especificados, o encontrar la subsecuencia común más larga restringida.

**Definición 2.2 (*Subsecuencia común más larga restringida*).** – en inglés *Constrained Longest Common Subsequence* (CLCS) – Sean las secuencias $x$, $y$ y $z$ de longitud $m$, $n$ y $r$ ($m \geq n \geq r$) respectivamente, el CLCS de las secuencias $x$ y $y$ con respecto a $z$ consiste en encontrar un LCS de $x$ y $y$ tal que $z$ sea subsecuencia del LCS$(x, y)$. Por lo tanto se puede inferir $\mathcal{L}_z(x, y) \leq \mathcal{L}(x, y)$.

**Ejemplo 2.3.** El CLCS de la secuencia "`GENÉTICA`" contra otras secuencias con la secuencia de restricción "`TA`" se puede observar en la Figura 2.2.

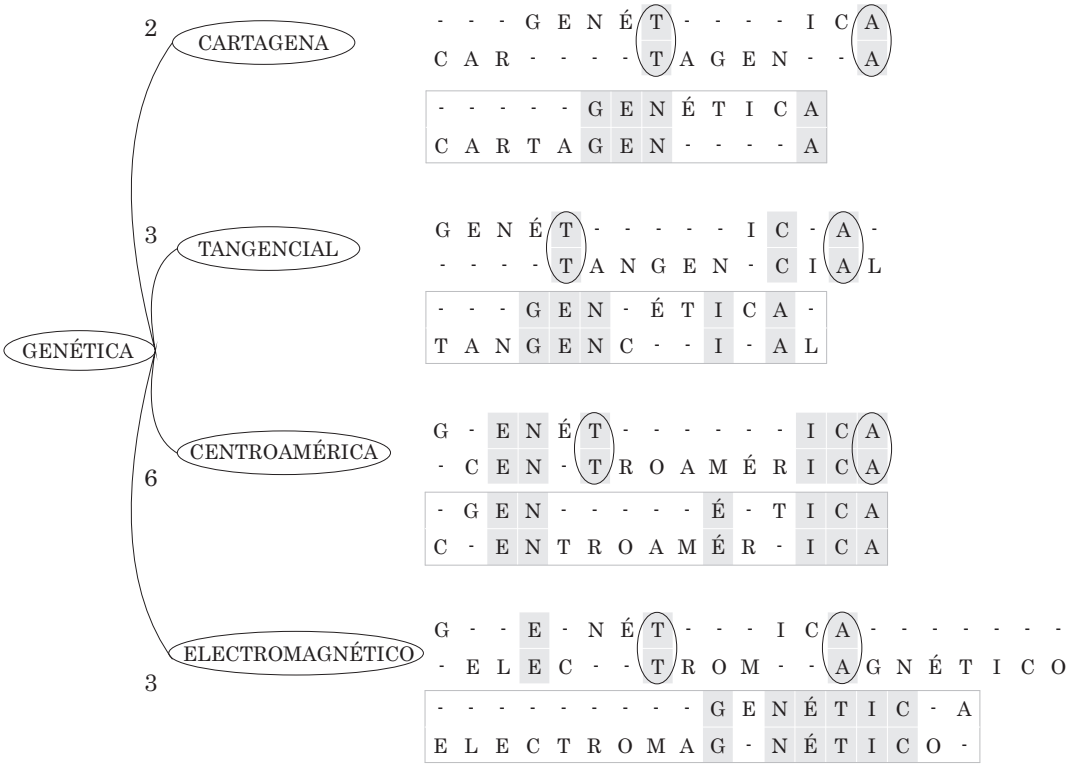


FIGURA 2.2: CLCS para la secuencia "`GENÉTICA`" contra otras secuencias con la secuencia de restricción "`TA`". Los números representan el CLCS entre las secuencias, además, para cada par de secuencias se muestra un posible alineamiento (enmarcado el LCS). Se usa el carácter '-' como carácter vacío.

## 2.5 Alineamiento de secuencias con restricción

El alineamiento de un par de secuencias con restricción – en inglés *Constrained Pairwise Sequence Alignment* (CPSA) – es un caso del CLCS el cual usa una función especial de

distancia [8], además, puede ser asociado a uno de los casos más sencillos del alineamiento de secuencias con motivos [3].

**Definición 2.3 (*Alineamiento de un par de secuencias*).** Sea $x$, $y \in \Sigma$, un alineamiento de estas dos secuencias se define como dos secuencias de igual longitud $x'$ y $y' \in \Sigma \cup \{\text{“–”}\}$ tal que $|x'| = |y'| = m'$, donde $m' \geq m, n$, y removiendo todos los caracteres “–” de $x'$ y $y'$ se obtiene $x$ y $y$, respectivamente, con la suposición de que no hay $x' = y' =$ “–” para algún $1 \leq i \leq m'$.

**Definición 2.4 (*Alineamiento de un par de secuencias con restricción*).** Sea $x, y, z \in \Sigma$, una solución es un alineamiento de $x$ y $y$ tal que para cada carácter de $z$ aparece una columna en el alineamiento y en el mismo orden de $z$, es decir, existe una lista de números enteros $\langle c_1, c_2, \ldots, c_r \rangle$ donde $1 \leq c_1 < c_2 < \ldots < c_r \leq m'$, y para todo $1 \leq k \leq r$, se tiene $x'_{c_k} = y'_{c_k} = p_k$.

**Definición 2.5 (*Puntaje de alineamiento de un par de secuencias*).** El puntaje de alineamiento de dos secuencias $x'$ y $y'$ de longitud $m'$ se define como $\Sigma_{1 \leq i \leq m'} \delta(x'_i, y'_i)$.

Es importante observar como el problema del LCS es el problema de alineamiento global cuando para la función de distancia los caracteres sin coincidencia y la inserción de espacios tienen el valor de 0 [21].

El alineamiento de un par de secuencias con restricción consiste en encontrar $x'$ y $y'$ con un mínimo puntaje de alineamiento para las secuencias dadas $x$ y $y$, la secuencia de restricción $z$ y una función de distancia $\delta(x', y')$ que mide la distancia entre dos caracteres.

El CLCS es equivalente al alineamiento de un par de secuencias con restricción de $x$, $y$ y $z$ usando la función de distancia (Ecuación 2.7) [8]. Esta función de distancia beneficia las coincidencias y no penaliza los caracteres sin coincidencia o la inserción de espacios. Por lo tanto, el puntaje del alineamiento son las coincidencias en $x$ y $y$ con respecto a $z$.

$$\delta(x', y') = \begin{cases} -1 & \text{si } x' = y' \text{ (coincidencia)} \\ 0 & \text{si } x' \neq y' \text{ (inserción, eliminación o reemplazo)} \end{cases} \tag{2.7}$$

# 3

# Técnicas de solución de la subsecuencia común más larga

Un algoritmo es una descripción que debe funcionar a fin de resolver un problema específico. Como tal, la palabra "algoritmo" es derivación del nombre al–Khwarizmi, árabe matemático del siglo 9 de la corte del califa Mamun de Bagdad. Muchos algoritmos comparten ideas similares para solucionar problemas, incluso problemas muy diferentes. Estas ideas similares son agrupadas como técnicas y son importantes a la hora de diseñar un algoritmo [21].

Este capítulo describe los algoritmos más simples para encontrar la longitud de la subsecuencia común más larga de dos secuencias ($\mathcal{L}(x, y)$) y recuperar al menos una solución ($\mathrm{LCS}(x, y)$) según algunas de las técnicas de diseño de algoritmos más importantes: la programación dinámica, los autómatas finitos, la programación dinámica escasa, divide y vencerás, y el paralelismo de bits.

## 3.1 Programación dinámica

La técnica de programación dinámica construye una solución del problema a partir de las soluciones de los subproblemas en los que se divide el problema original. Además, la programación dinámica organiza los cálculos para evitar que los algoritmos resuelvan el mismo subproblema varias veces principalmente cuando el número de subproblemas llega a ser demasiado grande y conllevar a un incremento de tiempo computacional [21].

El algoritmo más simple para encontrar el LCS desde la técnica de la programación dinámica emplea un tiempo y espacio computacional $O(mn)$. Si solo se necesita recuperar la longitud del LCS, la complejidad computacional en tiempo es $O(mn)$ y en espacio $O(n)$ con la fila actual y la fila previa almacenadas en memoria [22].

Usando la programación dinámica para encontrar el LCS, el algoritmo construye una matriz $L[0..m, 0..n]$ donde $L(i, j)$ representa el LCS para $x_1 \ldots i$ y $y_1 \ldots j$. Cada valor en la matriz se calcula basándose en los valores de las posiciones vecinas anteriores: izquierda, superior y diagonal superior izquierda, para ello, se utiliza la Ecuación 3.1 presentada por Hirschberg en [16]. Así, de la matriz de programación dinámica se obtiene $\mathcal{L}(x, y)$ correspondiente a $L(m, n)$.

$$L(i,j) = \begin{cases} 0 & \text{si } i = 0 \text{ o } j = 0 \\ L(i-1,\, j-1) + 1 & \text{si } x_i = y_j \\ \text{máx.}\,\{L(i-1,\, j), L(i,\, j-1)\} & \text{si } x_i \neq y_j \end{cases} \tag{3.1}$$

**Ejemplo 3.1.** Si dadas las secuencias $x$ = "`ACCAAG`" y $y$ = "`TCCACA`" se construye la matriz $L$ (véase Figura 3.1) y se puede establecer el tamaño del LCS entre ambas secuencias como 4, porque la celda $L(6, 6)$ tiene el valor 4.

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| | | ε | **T** | **C** | **C** | **A** | **C** | **A** |
| 0 | ε | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | **A** | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | **C** | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| 3 | **C** | 0 | 0 | 1 | 2 | 2 | 2 | 2 |
| 4 | **A** | 0 | 0 | 1 | 2 | 3 | 3 | 3 |
| 5 | **A** | 0 | 0 | 1 | 2 | 3 | 3 | 4 |
| 6 | **G** | 0 | 0 | 1 | 2 | 3 | 3 | 4 |

FIGURA 3.1: Construcción de la matriz $L$ con la técnica de programación dinámica para encontrar el LCS para las secuencias $x$ = "`ACCAAG`" y $y$ = "`TCCACA`". Las líneas sobresalientes enmarcan los límites y los números encerrados en círculos los puntos dominantes para cada límite. $\mathcal{L}(x, y) = 4$. LCS$(x, y)$ = "`CCAA`".

Para encontrar el LCS de las secuencias dadas, los conceptos de límites y puntos dominantes son usados [5]. El límite es la delimitación de un área sobre la matriz $L$ de números similares, en la Figura 3.1 los límites son representados por las líneas sobresalientes. Los puntos dominantes son las esquinas de los límites o los valores en los cuales todos los valores de las posiciones vecinas anteriores son menores a él. El LCS se puede recuperar recorriendo los límites de la parte inferior derecha hacia la parte superior izquierda de la matriz cruzando sobre los puntos dominantes. En el Ejemplo 3.1 se inicia con el límite máximo o el límite que contiene solo números 4 ubicando la posición $L(5, 6)$ (punto dominante) donde hay una coincidencia del carácter '`A`', después se avanza al límite de números 3 (posición $L(4, 4)$ con coincidencia '`A`'), luego al límite de números 2 (posición $L(3, 3)$ con coincidencia '`C`' y finalmente al límite de números 1 (posición $L(2, 2)$ con coincidencia '`C`'). Entonces, el LCS recuperado es "`CCAA`".

## 3.2 Autómatas finitos

Un autómata es un modelo computacional que consiste de una serie de estados, un estado inicial, un alfabeto de entrada y una función de transición mapeando los símbolos de entrada y los estados actuales a un próximo estado. El cálculo comienza en el estado inicial con una secuencia de entrada y los cambios a nuevos estados dependen de la función de transición.

Un grafo acíclico directo es una clase de autómata finito que representa los sufijos de una secuencia dada en los cuales cada arco esta etiquetado con un carácter o símbolo. Los caracteres o símbolos a lo largo de una ruta desde la raíz a un nodo son la subsecuencia la cual el nodo representa.

El grafo de subsecuencias acíclico directo – *Directed Acyclic Subsequence Graph* (DASG) – es un grafo análogo a un grafo de palabras acíclico directo – *Directed Acyclic Word Graph* (DAWG) – usando subsecuencias en vez de palabras o subsecuencias con símbolos de forma obligatoriamente contiguos (*substrings* en inglés). Un DASG acepta todas las subsecuencias que ocurren en una secuencia dada [11].

La matriz de programación dinámica se puede representar como un grafo donde los nodos son las celdas y los arcos representan las operaciones. El costo o peso de los arcos corresponde al costo de las operaciones. Para este nuevo planteamiento la idea

es encontrar la ruta más corta desde el nodo ubicado en la parte superior izquierda (posición $(0,0)$ en la matriz de programación dinámica) al nodo ubicado en la parte inferior derecha (posición $(m,n)$ en la matriz de programación dinámica). La suma total del costo de la ruta más corta en el grafo representa el costo mínimo de las operaciones necesarias para transformar $x$ en $y$ y el cálculo del LCS es el número de coincidencias sobre esta ruta [21].

El tiempo computacional del algoritmo para encontrar la ruta más corta en el grafo es $O(mn\log(mn))$.

**Ejemplo 3.2.** La Figura 3.2, encuentra el LCS con la técnica de autómatas para las secuencias $x =$ "`TCCACA`" y $y =$ "`ACCAAG`".

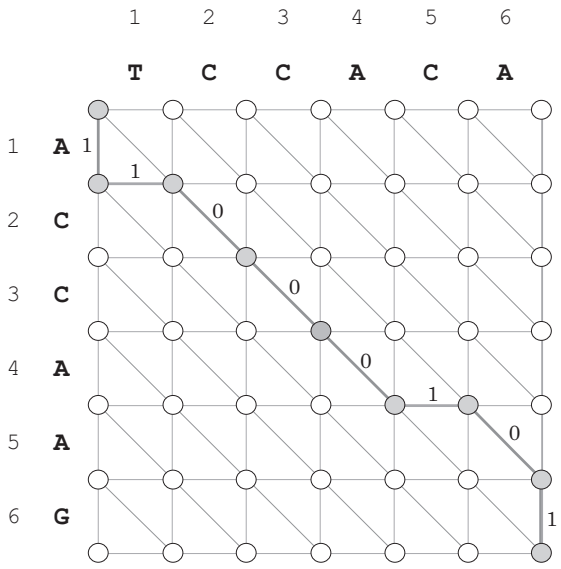


FIGURA 3.2: Grafo para las secuencias $x =$ "`ACCAAG`" y $y =$ "`TCCACA`". La línea sobresaliente es la ruta más corta desde la parte superior izquierda (posición $(0,0)$ en la matriz de programación dinámica) al nodo ubicado en la parte inferior derecha (posición $(m,n)$ en la matriz de programación dinámica). Los arcos diagonales representan coincidencias. $\mathcal{L}(x,y) = 4$.

## 3.3 Programación dinámica escasa

La programación dinámica escasa – en inglés *Sparse Dynamic Programming* – se basa en el hecho que en un conjunto de recurrencias de programación dinámica maximizando (o minimizando) alguna función objetivo y esta se calcule usando una matriz de programación dinámica, existe un número pequeño o escaso de posiciones en la matriz

contribuyendo a la optimización de la función objetivo. La ventaja de la técnica está en la reducción del tiempo empleado por los algoritmos, dada la dependencia de un conjunto pequeño de posiciones con respecto al conjunto total de posiciones de la matriz de programación dinámica para encontrar una solución.

En el caso del LCS, los algoritmos dependen de parámetros diferentes al tamaño de las secuencias dadas, como el número de coincidencias y la longitud del LCS. El conjunto pequeño o escaso de posiciones en la matriz de programación de dinámica necesario para resolver el problema del LCS está formado por las coincidencias $k$–dominantes de la matriz [28].

Uno de los principales algoritmos para calcular el LCS usando esta técnica es el algoritmo de Hunt–Szymanski [17], aunque se han realizado varios estudios empleando la técnica para solucionar el problema [4].

El algoritmo de Hunt–Szymanski consiste en calcular todos los puntos dominantes $\mathcal{Q}$ fila tras fila de arriba hacia abajo. El algoritmo necesita de un proceso de inicialización construyendo una lista ordenada de coincidencias $MATCHLIST(i)$[1] para cada $i$ que contiene la secuencia descendente de posiciones $j_1 < j_2 \ldots < j_d$ para las cuales $x_i = y_{j_q}$ para $q \in \{1..d\}$. Este proceso se puede lograr en $O(n \log n)$ en una estructura donde se ordenen los caracteres de $y$ de forma ascendente y la posición original de cada elemento en orden descendente. Luego, iterativamente para cada fila $i$ el algoritmo evalúa la función *threshold*[1] $T(i,k)$ correspondiente a la posición de $j$ donde la $k$–anticadena corta la fila $i$, representada en la Ecuación 3.2; los valores de la función $T$ pueden ser almacenados en un vector $THRESH$[1]. Para cada coincidencia, el algoritmo determina si este es $k$–dominante para algún $k$, proceso que puede ser logrado con una complejidad de tiempo $O(\log n)$ usando búsqueda binaria sobre el vector $THRESH$.

$$T(i,k) = \begin{cases} \text{el más pequeño } j \mid x_i = y_j \text{ y } T(i-1,k-1) < j \leq T(i-1,k) \\ T(i-1,k), \text{ si no existe tal } j \end{cases} \tag{3.2}$$

**Ejemplo 3.3.** Dadas las secuencias $x$ = "`ACCAAG`" y $y$ = "`TCCACA`" calcular la lista ordenada de coincidencias $MATCHLIST$ y el vector $THRESH$ con los valores de la función $T(i,k)$ (véase Figura 3.3). A continuación se muestra como se lleva a cabo

[1] nombre original en el trabajo de Hunt–Szymanski de 1977 [17]

la creación de la estructura sobre la secuencia $y$, primera etapa que contribuye en el preprocesamiento para la construcción de $MATCHLIST$.

$$\begin{array}{cc} \texttt{T C C A C A} & \\ & \Longrightarrow \\ \texttt{1 2 3 4 5 6} & \end{array} \quad \begin{array}{c} \texttt{A A C C C T} \\ \\ \texttt{6 4 5 3 2 1} \end{array}$$

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| | | **T** | **C** | **C** | **A** | **C** | **A** |
| 1 | **A** | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | **C** | 0 | 1 | 1 | 1 | 2 | 2 |
| 3 | **C** | 0 | 1 | 2 | 2 | 2 | 2 |
| 4 | **A** | 0 | 1 | 2 | 3 | 3 | 3 |
| 5 | **A** | 0 | 1 | 2 | 3 | 3 | 4 |
| 6 | **G** | 0 | 1 | 2 | 3 | 3 | 4 |

*MATCHLIST*

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 6 | 4 | |
| 2 | 5 | 3 | 2 |
| 3 | 5 | 3 | 2 |
| 4 | 6 | 4 | |
| 5 | 6 | 4 | |
| 6 | | | |

*THRESH*

| | $k$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 4 | - | - | - | - | - |
| 2 | 2 | 5 | - | - | - | - |
| 3 | 2 | 3 | - | - | - | - |
| 4 | 2 | 3 | 4 | - | - | - |
| 5 | 2 | 3 | 4 | 6 | - | - |
| 6 | 2 | 3 | 4 | 6 | - | - |

FIGURA 3.3: Matriz de programación dinámica para las secuencias $x =$"`ACCAAG`" y $y =$ "`TCCACA`", se muestran las anticadenas (líneas sobresalientes de color gris) para cada $k$. Además, el cálculo de $THRESH$ y $MATCHLIST$ para las secuencias dadas, $x$ y $y$. En $THRESH$ se puede observar el $\mathcal{L}(x, y)$, como el valor de $k$ donde $i = m$. En este caso, $\mathcal{L}(x, y) = 4$, porque el valor de $k$ para $i = m$ es 4.

La complejidad del algoritmo (véase Algoritmo 3.1) en tiempo es $O((\mathcal{M} + n) \log n)$ y en espacio es $O(\mathcal{M} + n)$ en muchos casos, pero no es favorable cuando $\mathcal{M}$ tiende a $mn$, convirtiéndose la complejidad de tiempo en $O(mn \log n)$, una complejidad de tiempo peor a la del algoritmo de programación dinámica [1].

## 3.4 Divide y vencerás

Divide y vencerás[2], hace referencia a la estrategia para tratar un gran problema que puede ser muy difícil de resolver. La estrategia consiste en dividir el problema en problemas más simples tantas veces cuanto sea necesario, o en otras palabras, se llega a la solución del problema original cuando la solución de las partes se torne lo suficientemente sencilla.

En ciencias de la computación la técnica divide y vencerás hace justamente lo anterior y como su nombre lo dice, la técnica procede en dos fases distintas: un fase de división en

[2]en latín, *Divide et vinces*, es una frase célebre de Julio César – (Roma, 100 a.C – 44 a.C.) militar y político cuya dictadura puso fin a la república en Roma –

**Algoritmo 3.1** Algoritmo de Hunt–Szymanski para recuperar el LCS

**Requiere:** $(x, y, m, n, MATCHLIST, C, p)$
1: $p \leftarrow 0$
2: $THRESH_0 \leftarrow 0$
3: $LINK_0 \leftarrow null$
4: **para** $i \leftarrow m$ **haga**
5: $\quad THRESH_i \leftarrow n + 1$
6: $\quad LINK_i \leftarrow null$
7: **para** $i \leftarrow m$ **haga**
8: $\quad$ **para cada** $j$ en $MATCHLIST_i$ **haga**
9: $\quad\quad$ encontrar $k \mid THRESH_{k-1} < j \leq THRESH_k$
10: $\quad\quad$ **si** $j < THRESH_k$ **entonces**
11: $\quad\quad\quad THRESH_k \leftarrow j$
12: $\quad\quad\quad LINK_k \leftarrow newnode(i, LINK_{k-1})$
13: $t \leftarrow$ el más largo $k \mid THRESH_k \neq n + 1$
14: $temp \leftarrow LINK_t$
15: **mientras que** $temp.next \neq null$ **haga**
16: $\quad p \leftarrow p + 1$
17: $\quad D_p \leftarrow x_{temp.i}$
18: $\quad temp \leftarrow temp.next$
19: **retornar** $C \leftarrow D^*$

la cual el algoritmo divide un problema en problemas más pequeños y los resuelve; y una fase de conquistar en la cual combina las soluciones para los problemas más pequeños para encontrar una solución al problema original. En la fase de dividir el problema original en subproblemas, usualmente si estos subproblemas son relativamente grandes, se aplicará de nuevo la división hasta lograr un punto en el cual los subproblemas sean lo adecuadamente pequeños para ser solucionados de forma sencilla [21].

Uno de los principales algoritmos basados en la técnica de divide y vencerás tiene como ventaja importante lograr una complejidad de espacio lineal en el cálculo del LCS, es el propuesto por Daniel Hirschberg [16] en 1975.

Hirschberg demostró como es posible recuperar el LCS con una complejidad de espacio en $O(m + n)$ y una complejidad de tiempo en $O(mn)$. El método depende del hecho de que $\mathcal{L}(x, y)$ debe ser igual a $\mathcal{L}(x^*, y^*)$. La principal idea del algoritmo de Hirschberg (Algoritmo 3.3) consiste en encontrar el punto medio de una "curva" del LCS y usar la técnica de divide y vencerás para reconstruir la ruta del LCS (véase Figura 3.4).

Para determinar de manera eficiente el punto medio de una curva del LCS se utiliza el algoritmo 3.2 de espacio lineal de la siguiente forma:

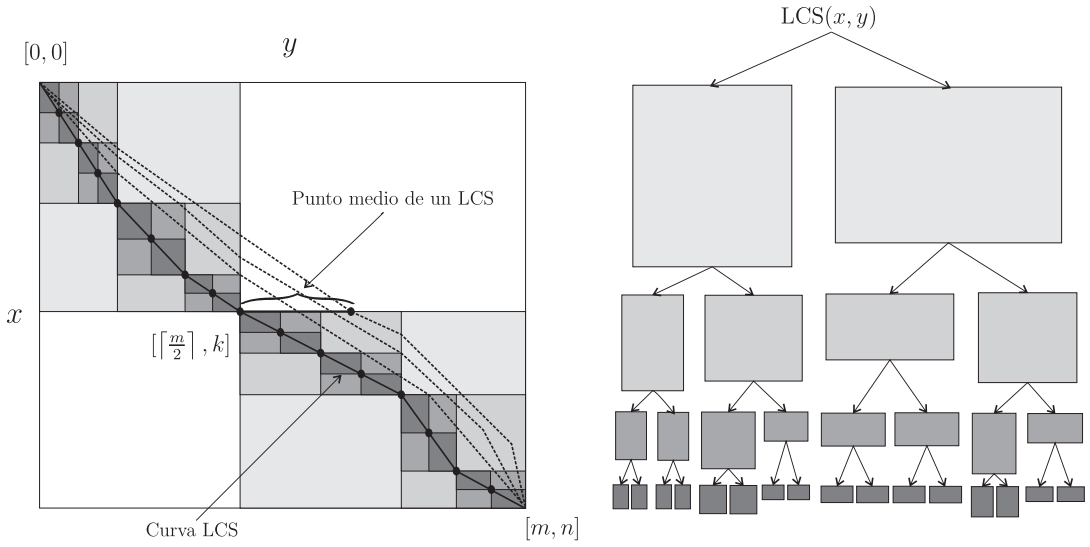


FIGURA 3.4: Representación de la técnica de divide y vencerás (izquierda) y los llamados recursivos (derecha).

- ⋆ Calcular la última fila de la matriz $L$ entre $x_{1..\lceil \frac{m}{2} \rceil}$ y $y$
- ⋆ Calcular la última fila de la matriz $L^*$ entre $x^*_{\lceil \frac{m}{2} \rceil + 1..m}$ y $y^*$
- ⋆ Calcular $k$ que maximiza $L'(k) = L(k) + L^*(n-k)$. Entonces, para $0 \leq k \leq n, L'(k) = \mathcal{L}(x, y)$

Después de haberse determinado el punto medio se puede dividir el problema original en dos subproblemas del mismo tipo. Estos dos subproblemas son $\text{LCS}(x_{1..\lceil \frac{m}{2} \rceil}, y_{1..k})$ y $\text{LCS}(x_{\lceil \frac{m}{2} \rceil + 1..m}, y_{k+1..n})$. La solución al problema original es la concatenación de la solución del primer subproblema seguido de la solución del segundo subproblema. Los subproblemas se resuelven de manera recursiva de la misma forma hasta convertirse en casos triviales.

**Ejemplo 3.4.** Dadas las secuencias $x =$ "`ACCAAG`" y $y =$ "`TCCACA`", se puede determinar el LCS de $x$ y $y$ usando el algoritmo de Hirschberg (véase Figura 3.5).

## 3.5 Paralelismo de bits

Esta técnica está basada en las operaciones sobre el conjunto de bits o palabra (*Word* en inglés) que puede ser procesada o transmitida en paralelo. El número de bits de

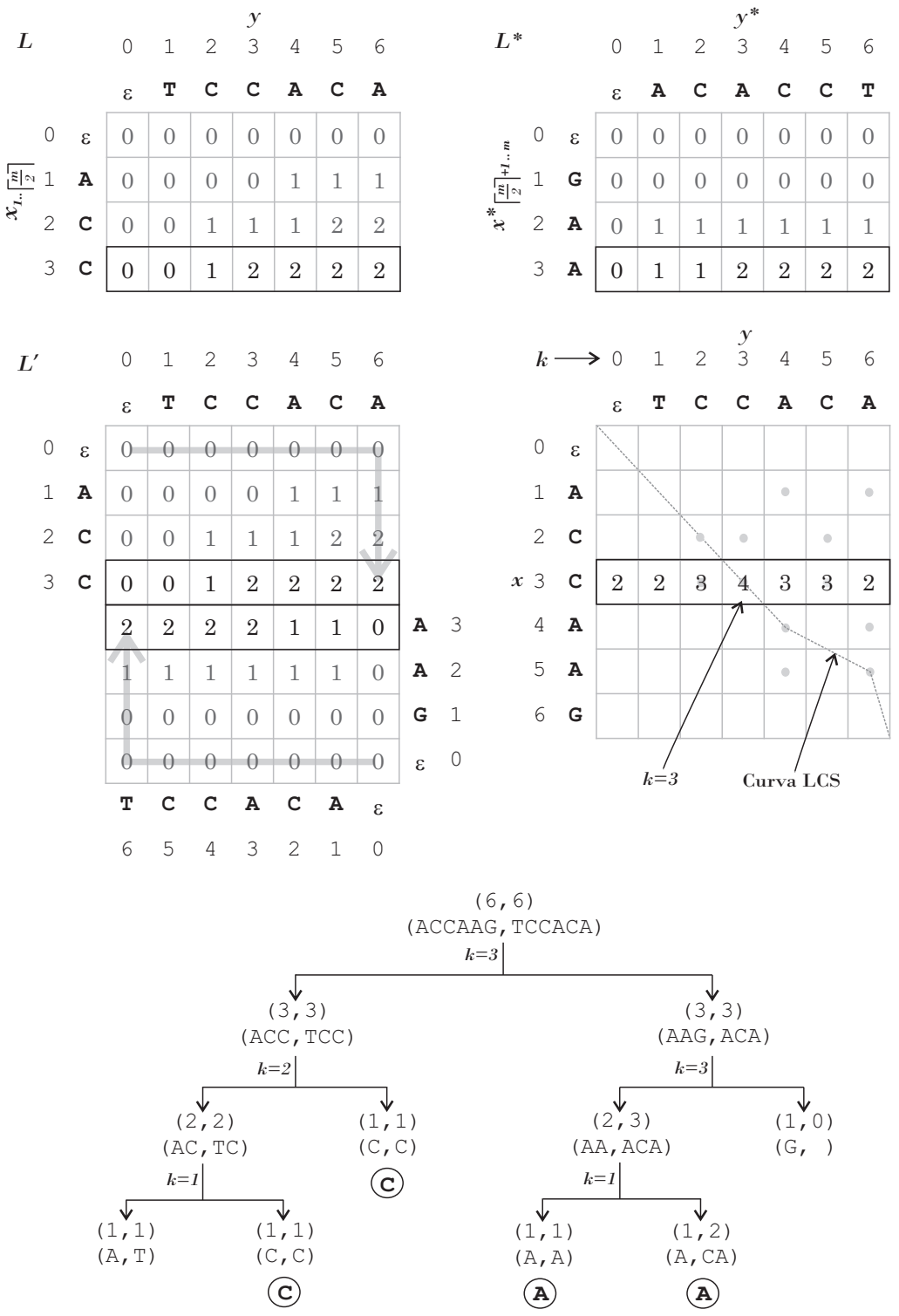


FIGURA 3.5: Las matrices representan el cálculo del punto medio de una "curva" del LCS para las secuencias $x =$"`ACCAAG`" y $y =$ "`TCCACA`" usando el algoritmo de Hirschberg. El árbol representa la técnica de divide y vencerás usando el mismo algoritmo y las secuencias dadas $x$ y $y$. Las letras dentro de los círculos representan el LCS de las secuencias $x$ y $y$, es decir, $\mathrm{LCS}(x, y) =$ "`CCAA`".

**Algoritmo 3.2** Algoritmo FindRow (Algoritmo de Hirschberg)

**Requiere:** $(x, y, m, n, L)$
1: $L_{0..n} \leftarrow 0$
2: **para** $i \leftarrow 1 \leq m$ **haga**
3:  **para** $j \leftarrow 1 \leq n$ **haga**
4:  **si** $x_i = y_j$ **entonces**
5:  $Lnew_j \leftarrow L_{j-1} + 1$
6:  **de lo contrario**
7:  $Lnew_j \leftarrow$máx.$(Lnew_{j-1} + 1, L_j)$
8: **para** $j \leftarrow 1 \leq n$ **haga**
9:  $L_j = Lnew_j$
10: **retornar** $C$

**Algoritmo 3.3** Algoritmo de Hirschberg para recuperar el LCS

**Requiere:** $H(x, y, m, n, C, p)$
1: **si** $n = 0$ **entonces**
2:  $p \leftarrow 0$
3: **de lo contrario**
4:  **si** $m = 1$ **entonces**
5:  **si** $x_i = y_j$ **entonces**
6:  $p \leftarrow 1$
7:  $C_1 \leftarrow x_1$
8:  **de lo contrario**
9:  $p \leftarrow 0$
10:  **de lo contrario**
11:  $i \leftarrow \lceil \frac{m}{2} \rceil$
12:  FindRow$(x, y, i, n, L)$
13:  FindRow$(x^*, y^*, m - i, n, L^*)$
14:  $k, max \leftarrow 0$
15:  **para** $l \leftarrow 0 \leq n$ **haga**
16:  **si** $L_l + L^*_{n-l} > max$ **entonces**
17:  $max \leftarrow L_l + L^*_{n-l}$
18:  $k \leftarrow l$
19:  H$(x, y, i, k, C, q)$
20:  H$(x_{i+1..m}, y_{k+1..n}, m - i, n - k, D, r)$
21:  $p \leftarrow q + r$
22:  **para** $l \leftarrow 1 \leq r$ **haga**
23:  $C_{q+l} \leftarrow D_l$
24: **retornar** $C$

la palabra es una importante característica de la arquitectura de los computadores. Actualmente los computadores tienen una palabra de tamaño de 32 o 64 bits.

El algoritmo en paralelismo de bits se usa para calcular eficientemente la matriz creada por programación dinámica. Cada columna de la matriz de programación dinámica se calcula de manera de paralelismo de bits bajo el supuesto $m \leq \omega$, donde $\omega$ es el número de bits de la palabra del computador [18]. El algoritmo de Crochemore *et al.* [10] para

recuperar el LCS emplea un tiempo y espacio computacional $O(mn/\omega)$. Para encontrar el tamaño del LCS, el algoritmo consume un tiempo computacional igual a recuperar el LCS, pero su espacio es $O(m/\omega)$ (véase Algoritmo 3.4).

El algoritmo primero construye una matriz $M$ y una matriz $M'$ (negación de la matriz $M$), formadas por vectores de bits para cada carácter de la secuencia (véase Figura 3.6). Si las secuencias $x$ y $y$ pertenecen a un alfabeto $\Sigma$ entonces $M(\Sigma)$ es definido como $M[\mu]_i = 1$ si $x_i = \mu$, de lo contrario es igual a 0. Es decir, los números 1 que aparecen en los vectores para cada símbolo es la ocurrencia del símbolo en la secuencia $x$.

El próximo paso del algoritmo es construir la matriz $V'$ con la Ecuación 3.3. Finalmente, el tamaño del LCS es el número de veces que un corrimiento (*carry* en inglés) ocurre.

$$V'_j = \begin{cases} 2^m - 1, & \text{para } j = 0 \\ (V'_{j-1} + (V'_{j-1} \text{ AND } M(Y_j)))\text{OR}(V'_{j-1} \text{ AND } M'(Y_j)), & \text{para } j \in 1 \ldots n \end{cases} \tag{3.3}$$

**Ejemplo 3.5.** La Figura 3.6, encuentra el LCS con la técnica de vectores de bits para las secuencias $x =$ "`ACCAAG`" y $y =$ "`TCCACA`" sobre el alfabeto $\Sigma = \{$`A,C,G,T`$\}$.

Matriz $V'$

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| | | $2^m$-1 | T | C | C | A | C | A |
| 1 | A | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | C | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 3 | C | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | A | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | A | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 6 | G | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | 1 | 1 | 1 | | 1 |

Matriz $M$

| | | A | C | G | T |
|---|---|---|---|---|---|
| 1 | A | 1 | 0 | 0 | 0 |
| 2 | C | 0 | 1 | 0 | 0 |
| 3 | C | 0 | 1 | 0 | 0 |
| 4 | A | 1 | 0 | 0 | 0 |
| 5 | A | 1 | 0 | 0 | 0 |
| 6 | G | 0 | 0 | 1 | 0 |

Matriz $M'$

| | | A | C | G | T |
|---|---|---|---|---|---|
| 1 | A | 0 | 1 | 1 | 1 |
| 2 | C | 1 | 0 | 1 | 1 |
| 3 | C | 1 | 0 | 1 | 1 |
| 4 | A | 0 | 1 | 1 | 1 |
| 5 | A | 0 | 1 | 1 | 1 |
| 6 | G | 1 | 1 | 0 | 1 |

FIGURA 3.6: Construcción de las matrices $V'$, $M$ y $M'$ para encontrar el LCS con la técnica vectores de bits dadas las secuencias $x =$ "`ACCAAG`" y $y =$ "`TCCACA`". $\mathcal{L}(x, y) = 4$.

Para recuperar el LCS usando la técnica de vectores de bits se parte del hecho que una *anticadena* contribuye exactamente con un punto de coincidencia $k$–dominante y se puede asegurar como el conjunto resultante de coincidencias seleccionadas conforman un LCS [28]. En la matriz de programación dinámica $L$ se puede identificar una coincidencia

$k$–dominante si y sólo si:

$$k = L(i,j) = L(i-1,j-1)+1 = L(i-1,j)+1 = L(i,j-1)+1 \tag{3.4}$$

y en $V'$ se puede ver como:

$$V'(j-1)_i = 1 \text{ y } V'(j)_i = 0 \tag{3.5}$$

Entonces, para hallar el conjunto de todos los puntos $k$–dominantes (matriz $K$), se recurre a encontrar todos los cambios de bit de 0 a 1 de manera horizontal de izquierda a derecha en la matriz $V'$, para ello se utiliza la Ecuación 3.6.

$$K_j = (V'_{j-1} \text{ XOR } V'_j) \text{ AND } V'_{j-1} \text{ para } j \in \{1 \ldots n\} \tag{3.6}$$

**Algoritmo 3.4** Algoritmo de vectores de bits para recuperar el LCS

**Requiere:** $(x, y, m, n, V', K)$
1: $mascara \leftarrow 2^m - 1$
2: **para** $i \leftarrow n$ **descienda a** 1 **haga**
3: $Vmascara \leftarrow V'_j$ & $mascara$
4: $Vmascara' \leftarrow Vmascara$ & $K_j$
5: $k = ultimobit(Vmascara)$
6: $l = ultimobit(Vmascara')$
7: **si** $k = l$ **y** $Vmascara' > 0$ **entonces**
8: $mascara \leftarrow 2^k - 1$
9: **retornar** $(k, j)$

**Ejemplo 3.6.** La Figura 3.7, recupera el LCS con la técnica de vectores de bits para las secuencias $x$ = "`ACCAAG`" y $y$ "`TCCACA`" sobre el alfabeto $\Sigma = \{$`A,C,G,T`$\}$. Se construye la matriz $K$ usando la matriz $V'$ (véase Figura 3.6).

Matriz $V'$

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| | | **T** | **C** | **C** | **A** | **C** | **A** |
| 1 | **A** | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | **C** | 1 | 0 | 0 | 1 | 0 | 0 |
| 3 | **C** | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | **A** | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | **A** | 1 | 1 | 1 | 1 | 1 | 0 |
| 6 | **G** | 1 | 1 | 1 | 1 | 1 | 1 |

(a.)

Matriz $K$

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| | | **T** | **C** | **C** | **A** | **C** | **A** |
| 1 | **A** | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | **C** | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | **C** | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | **A** | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | **A** | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | **G** | 0 | 0 | 0 | 0 | 0 | 0 |

(b.)

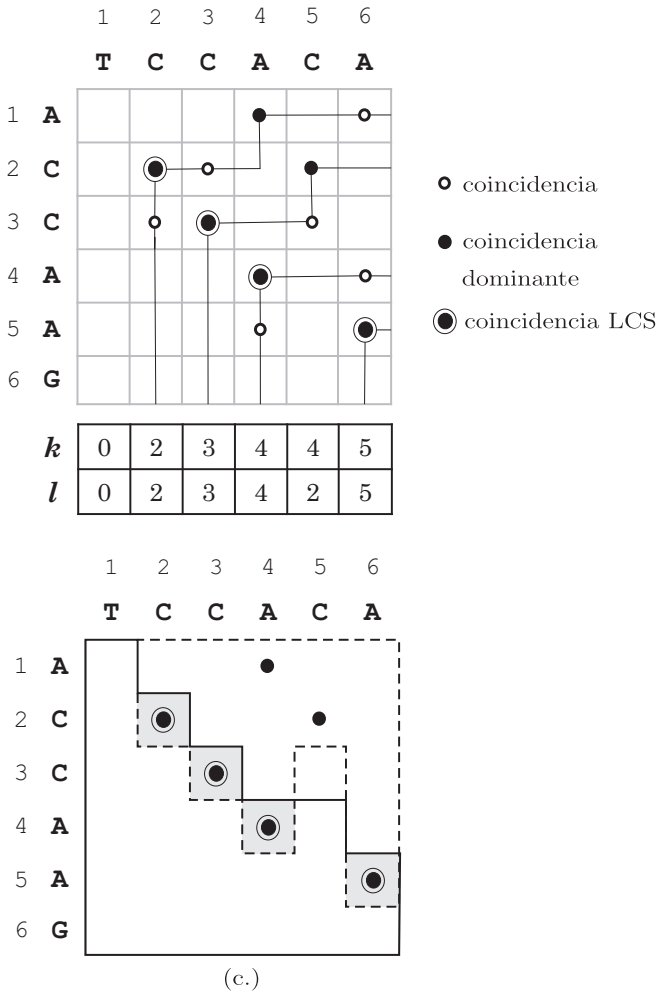


FIGURA 3.7: Construcción de las matrices (a.) $V'$ y (b.) $K$ para recuperar el LCS con la técnica vectores de bits dadas las secuencias $x$ = "`ACCAAG`" y $y$ = "`TCCACA`". En (c.) se presentan las coincidencias, las coincidencias dominantes y las coincidencias LCS del ejemplo. Además, se encuentran los valores obtenidos para $k$ y $l$ con su respectiva representación gráfica ($k$ línea punteada y $l$ línea continua). El LCS recuperado son los caracteres correspondientes a las intersecciones donde ocurren los cuadros de color gris. Entonces, el LCS$(x, y)$ = "`CCAA`".

# 4

# Algoritmos para solucionar el problema de la subsecuencia común más larga restringida

Dadas las secuencias $x, y$ y $z$, la subsecuencia común más larga restringida – *Constrained Longest Common Subsequence* (CLCS) – de las secuencias $x$ y $y$ respecto a $z$, consiste en encontrar un LCS de $x$ y $y$ tal que $z$ es una subsecuencia de este LCS.

**Ejemplo 4.1.** Para las secuencias $x$="`ACCAAG`", $y$="`TCCACA`" y $z$="`AC`", el LCS que contiene la secuencia $z$ es $\text{LCS}_z(x, y)$="`ACA`" de tamaño 3. Así sea el $\text{LCS}(x, y)$="`CCAA`" de tamaño 4 una subsecuencia de máximo tamaño, esta no contiene la secuencia $z$.

Los trabajos más representativos para solucionar el CLCS están basados en la técnica de programación dinámica, estos se resumen en la Tabla 4.1.

Tabla 4.1: Complejidad de los algoritmos para solucionar el CLCS

| Año | Complejidad | | Autor |
|---|---|---|---|
| | Tiempo | Espacio | |
| *2003* | $O(m^2n^2r)$ | $O(m^2n^2r)$ | Yin–Te Tsai [30] |
| *2003* | $O(mnr)$ | $O(mnr)$ | Peng Chao–Li [27] |
| *2004* | $O(mnr)$ | $O(mr)$ | Chin *et al.* [8] |
| *2005* | $O(mnr)$ | $O(mnr)$ | Arslan y Eğecioğlu [2] |
| *2006* | $O(mnr)$ | $O(mnr)$ | Wang Wei–Lun [31] |
| *2007* | $O(r(m\mathcal{L} + \mathcal{M}) + n)$ | $O(r\mathcal{M}+\text{máx.}(n, \sigma))$ | Deorowicz [13] |
| *2008* | $O(r\mathcal{R} \log\log n + n)$ | $\Theta(\text{máx.}\{\mathcal{R}, n\})$ | Iliopoulos y Rahman [20] |

A continuación se detalla el aporte de cada uno de los algoritmos para solucionar el CLCS.

## 4.1 Algoritmo de Yin–Te Tsai

Yin–Te Tsai en su artículo [30] introdujo el problema del CLCS y presentó un algoritmo basado en la programación dinámica. El algoritmo de Tsai usa la Ecuación 4.1 y la Ecuación 4.2 para encontrar el CLCS de las secuencias dadas. El algoritmo halla el máximo valor del LCS de todas las subsecuencias limitadas por las coincidencias de caracteres entre las dos secuencias y la secuencia de restricción. Esta operación se repite por cada carácter de la secuencia de restricción.

$$L_k(i,j) = \begin{cases} \mathcal{L}(x_{1..i-1}, y_{1..j-1}) + 1 & \text{si } k = 1 \text{ y} \\ & x_i = y_j = z_k \\ \text{máx.}_{\substack{1\le a<i \\ 1\le b<j}} \{L_{k-1}(a,b) + \mathcal{L}(x_{a+1..i-1}, y_{b+1..j-1}) + 1\} & \text{si } 2 \le k \le r \text{ y} \\ & x_i = y_j = z_k \\ -\infty & \text{otro caso.} \end{cases} \tag{4.1}$$

$$\mathcal{L}_z(x,y) = \text{máx.}_{\substack{1\le i\le m \\ 1\le j\le n}} \{L_r(i,j) + \mathcal{L}(x_{i+1..m}, y_{j+1..n})\} \tag{4.2}$$

El algoritmo de Tsai (véase Algoritmo 4.1) tiene una complejidad de tiempo y de espacio $O(m^2n^2r)$.

**Ejemplo 4.2.** Dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`" el $\text{LCS}_z(x,y)$ por el algoritmo de Tsai (véase Figuras 4.1 y 4.2) es el siguiente:

1. Para $k = 1$, $z_1 =$"`G`"; $L_1(1..m, 1..n) = -\infty$ excepto
   $L_1(4,1) = \mathcal{L}(1..3, 1..0) + 1 = 1$, $L_1(4,6) = \mathcal{L}(1..3, 1..5) + 1 = 2$, $L_1(4,11) = \mathcal{L}(1..3, 1..10) + 1 = 2$, $L_1(7,1) = \mathcal{L}(1..6, 1..0) + 1 = 1$, $L_1(7,6) = \mathcal{L}(1..6, 1..5) + 1 = 4$, $L_1(7,11) = \mathcal{L}(1..6, 1..10) + 1 = 5$, $L_1(10,1) = \mathcal{L}(1..9, 1..0) + 1 = 1$, $L_1(10,6) = \mathcal{L}(1..9, 1..5) + 1 = 4$, $L_1(10,11) = \mathcal{L}(1..9, 1..10) + 1 = 7$, $L_1(11,1) = \mathcal{L}(1..10, 1..0) + 1 = 1$, $L_1(11,6) = \mathcal{L}(1..10, 1..5) + 1 = 4$, $L_1(11,11) = \mathcal{L}(1..10, 10..10) + 1 = 7$

2. Para $k = 2$, $z_2 =$"`T`"; $L_2(1..m, 1..n) = -\infty$ excepto
   $L_2(12,2) = \text{máx.}\{ L_1(4,1) + \mathcal{L}(5..11, 2..1),\ L_1(7,1) + \mathcal{L}(8..11, 2..1),\ L_1(10,1) +$

**Algoritmo 4.1** Algoritmo de Tsai para encontrar el CLCS

**Requiere:** $(x, y, z, m, n, r)$
1: **para** $k < r$ **haga**
2: $L_k(1..m, 1..n) \leftarrow -\infty$
3: **para** $i < m$ **haga**
4: **para** $j < n$ **haga**
5: **si** $x_i = z_k$ **y** $y_j = z_k$ **entonces**
6: **si** $k = 1$ **entonces**
7: $L_k(i, j) \leftarrow \mathcal{L}(x_{1..i-1}, y_{1..j-1}) + 1$
8: **de lo contrario**
9: $aux, max \leftarrow 0$
10: **para** $a < m$ **haga**
11: **para** $b < n$ **haga**
12: **si** $i > a$ **y** $j > b$ **entonces**
13: $max \leftarrow L_{k-1}(a, b) + \mathcal{L}(x_{a+1..i-1}, y_{b+1..j-1}) + 1$
14: **si** $max > aux$ **entonces**
15: $aux \leftarrow max, L_k(i, j) \leftarrow max$
16: **para** $i < m$ **haga**
17: **para** $j < n$ **haga**
18: $\mathcal{T} \leftarrow L_r(i, j) + \mathcal{L}(x_{i+1..m}, y_{j+1..n})$
19: **retornar** $\mathcal{T}$ {Tamaño del CLCS}

$\mathcal{L}(11..11, 2..1)$, $L_1(11, 1) + \mathcal{L}(12..11, 2..1)$ } $+ 1 = 2$

$L_2(12, 12) =$ máx.{ $L_1(4, 1) + \mathcal{L}(5..11, 2..11)$, $L_1(4, 6) + \mathcal{L}(5..11, 7..11)$, $L_1(4, 11) + \mathcal{L}(5..11, 12..11)$, $L_1(7, 1) + \mathcal{L}(8..11, 2..11)$, $L_1(7, 6) + \mathcal{L}(8..11, , 7..11)$, $L_1(7, 11) + \mathcal{L}(8..11, 12..11)$, $L_1(10, 1) + \mathcal{L}(11..11, 2..11)$, $L_1(10, 6) + \mathcal{L}(11..11, 7..11)$, $L_1(10, 11) + \mathcal{L}(11..11, 12..11)$, $L_1(11, 1) + \mathcal{L}(12..11, 2..11)$, $L_1(11, 6) + \mathcal{L}(12..11, 7..11)$, $L_1(11, 11) + \mathcal{L}(12..11, 12..11)$ } $+ 1 = 8$

3. Para $k = 3$, $z_3 =$"`A`"; $L_3(1..m, 1..n) = -\infty$ excepto

   $L_3(14, 5) =$ máx.{ $L_2(12, 2) + \mathcal{L}(13..13, 3..4)$ } $+ 1 = 4$

   $L_3(14, 8) =$ máx.{ $L_2(12, 2) + \mathcal{L}(13..13, 3..7)$ } $+ 1 = 4$

   $L_3(14, 9) =$ máx.{ $L_2(12, 2) + \mathcal{L}(13..13, 3..8)$ } $+ 1 = 4$

4. $|\text{CLCS}_z(x, y)| =$ máx.{ $L_3(14, 5) + \mathcal{L}(15..15, 6..15)$, $L_3(14, 8) + \mathcal{L}(15..15, 9..15)$, $L_3(14, 9) + \mathcal{L}(15..15, 10..15)$ } $= 5$

## 4.2 Algoritmo de Peng Chao–Li

Chao–Li Peng [27] logró una reducción de complejidad de espacio y tiempo de $O(mnr)$ (véase Algoritmo 4.2) para solucionar el problema del CLCS empleando la Ecuación 4.3.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | G | T | C | C | A | G | C | A | A | C | G | T | C | C | C |
| 1 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 2 | T | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 3 | T | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 4 | G | 1 | - | - | - | - | 2 | - | - | - | - | 2 | - | - | - | - |
| 5 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 7 | G | 1 | - | - | - | - | 4 | - | - | - | - | 5 | - | - | - | - |
| 8 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 9 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 10 | G | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 11 | G | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 12 | T | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 13 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 14 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 15 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | G | T | C | C | A | G | C | A | A | C | G | T | C | C | C |
| 1 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 2 | T | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 3 | T | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 4 | G | 1 | - | - | - | - | 2 | - | - | - | - | 2 | - | - | - | - |
| 5 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 7 | G | 1 | - | - | - | - | 4 | - | - | - | - | 5 | - | - | - | - |
| 8 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 9 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 10 | G | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 11 | G | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 12 | T | - | 2 | - | - | - | - | - | - | - | - | - | 8 | - | - | - |
| 13 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 14 | A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 15 | C | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

FIGURA 4.1: Representación del algoritmo de Tsai para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x =$ "ATTGCAGACGGTCAC", $y =$ "GTCCAGCAACGTCCC" y $z =$ "GTA". Los círculos de color negro representan coincidencias en la etapa actual y los de color blanco en etapas anteriores. Dentro de los círculos negros aparecen los LCS de las correspondientes secuencias (límites de las áreas de color gris). El símbolo '-' representa $-\infty$. La primera tabla corresponde a $k = 1$ y la segunda tabla a $k = 2$

.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 2 | **T** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 3 | **T** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 4 | **G** | 1 | - | - | - | - | 2 | - | - | - | - | 2 | - | - | - | - |
| 5 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 7 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 5 | - | - | - | - |
| 8 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 9 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 10 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 11 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 12 | **T** | - | 2 | - | - | - | - | - | - | - | - | - | 8 | - | - | - |
| 13 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 14 | **A** | - | - | - | - | 4 | - | - | 4 | 4 | - | - | - | - | - | - |
| 15 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 2 | **T** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 3 | **T** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 4 | **G** | 1 | - | - | - | - | 2 | - | - | - | - | 2 | - | - | - | - |
| 5 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 7 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 5 | - | - | - | - |
| 8 | **A** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 9 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 10 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 11 | **G** | 1 | - | - | - | - | 4 | - | - | - | - | 7 | - | - | - | - |
| 12 | **T** | - | 2 | - | - | - | - | - | - | - | - | - | 8 | - | - | - |
| 13 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 14 | **A** | - | - | - | - | 4 | - | - | 4 | 4 | - | - | - | - | - | - |
| 15 | **C** | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 5 |

FIGURA 4.2: Representación del algoritmo de Tsai para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". Los círculos de color negro representan coincidencias en la etapa actual y los de color blanco en etapas anteriores. Dentro de los círculos negros aparecen los LCS de las correspondientes secuencias (límites de las áreas de color gris). El símbolo '-' representa $-\infty$. La primera tabla corresponde a $k = 3$ y la segunda tabla al resultado obtenido $\mathcal{L}_z(x, y) = 5$.

Sea $L_r(m,n) = \mathcal{L}_z(m,n)$, es decir, $L_r(m,n)$ representa el CLCS de la secuencia $x$ y $y$ con respecto a la secuencia $z$.

$$
\begin{aligned}
&L_k(i,0) = -\infty, 0 \leq i \leq m \text{ y } 1 \leq k \leq r \\
&L_k(0,j) = -\infty, 0 \leq j \leq n \text{ y } 1 \leq k \leq r \\
&L_k(i,j) = \begin{cases} L_{k-1}(i-1,j-1)+1 & \text{si } x_i = y_j = z_k \\ L_k(i-1,j-1)+1 & \text{si } x_i = y_j \text{ y } \\ & (k=0 \text{ o } x_i \neq z_k) \\ \text{máx.}\{L_k(i-1,j), L_k(i,j-1)\} & \text{si } x_i \neq y_j \end{cases} \\
&\text{donde } 1 \leq i \leq m, 1 \leq j \leq n \text{ y } 1 \leq k \leq r
\end{aligned} \tag{4.3}
$$

Si $|z| = 0$, es decir, si $z = \varepsilon$, entonces la matriz $L$ se convierte en una matriz de dos dimensiones de tamaño $(m+1) \times (n+1)$ y la ecuación 4.3 se transforma en la siguiente ecuación (4.4):

$$
\begin{aligned}
&L_0(i,0) = 0, 0 \leq i \leq m \\
&L_0(0,j) = 0, 0 \leq j \leq n \\
&L_0(i,j) = \begin{cases} L_0(i-1,j-1)+1 & \text{si } x_i = y_j \\ \text{máx.}\{L_0(i-1,j),\ L_0(i,j-1)\} & \text{si } x_i \neq y_j \end{cases} \\
&\text{donde } 1 \leq i \leq m, 1 \leq j \leq n
\end{aligned} \tag{4.4}
$$

La Ecuación 4.4 es exactamente la relación de recurrencia la cual se usa para encontrar el LCS$(x,y)$ por programación dinámica (véase ecuación 3.1). Así, se puede decir que el LCS resulta como un caso especial del CLCS cuando la secuencia de restricción es nula.

Como se requiere $r+1$ matrices $L$ de tamaño $(m+1) \times (n+1)$, se necesita $O(mnr)$ espacio. Para cada $L_k(i,j)$ el tiempo de cálculo es $O(1)$. Entonces, el resultado de la complejidad de tiempo del algoritmo para resolver el CLCS requiere $O((m+1) \times (n+1) \times (r+1) \times (1 \times 4)) = O(mnr)$.

**Ejemplo 4.3.** Las Figuras 4.3 y 4.4 presentan el LCS$_z(x,y)$ por el algoritmo de Peng para las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`".

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ε | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 0 | ε | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | **A** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | **T** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 3 | **T** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | **G** | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 5 | **C** | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 6 | **A** | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 7 | **G** | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| 8 | **A** | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 9 | **C** | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 |
| 10 | **G** | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 7 | 7 |
| 11 | **G** | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 7 | 7 |
| 12 | **T** | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 8 | 8 | 8 | 8 |
| 13 | **C** | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9 |
| 14 | **A** | 0 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 6 | 6 | 6 | 7 | 8 | 9 | 9 | 9 |
| 15 | **C** | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 10 | 10 |

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ε | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 0 | ε | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 1 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 2 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 3 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 4 | **G** | – | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 5 | **C** | – | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 6 | **A** | – | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 7 | **G** | – | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| 8 | **A** | – | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 9 | **C** | – | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 |
| 10 | **G** | – | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 7 | 7 |
| 11 | **G** | – | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 7 | 7 |
| 12 | **T** | – | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 8 | 8 | 8 | 8 |
| 13 | **C** | – | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9 |
| 14 | **A** | – | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 6 | 6 | 6 | 7 | 8 | 9 | 9 | 9 |
| 15 | **C** | – | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 10 | 10 |

FIGURA 4.3: Dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". En la tabla superior se presenta $k = 0$ donde los círculos de color blanco representan el LCS de las secuencias. Los círculos de color gris representan el CLCS de las secuencias dadas. La tabla inferior representa el $k = 1$.

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ε | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 0 | ε | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 1 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 2 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 3 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 4 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 5 | **C** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 6 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 7 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 8 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 9 | **C** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 10 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 11 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 12 | **T** | – | – | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 8 | 8 | 8 | 8 |
| 13 | **C** | – | – | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 8 | 9 | 9 | 9 |
| 14 | **A** | – | – | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 8 | 9 | 9 | 9 |
| 15 | **C** | – | – | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 8 | 9 | 10 | 10 |

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ε | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 0 | ε | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 1 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 2 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 3 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 4 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 5 | **C** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 6 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 7 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 8 | **A** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 9 | **C** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 10 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 11 | **G** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 12 | **T** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 13 | **C** | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 14 | **A** | – | – | – | – | – | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 15 | **C** | – | – | – | – | – | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

FIGURA 4.4: Dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". En la tabla superior se presenta $k = 2$ y la tabla inferior representa el $k = 3$. Los círculos de color gris representan el CLCS de las secuencias dadas. $\mathcal{L}_z(x, y) = 5$.

**Algoritmo 4.2** Algoritmo de Peng para encontrar el CLCS

**Requiere:** $(x, y, z, m, n, r)$
1: $k \leftarrow 0, i \leftarrow 1, j \leftarrow 1$
2: $L_0(1..m, 0) \leftarrow 0, L_0(0, 1..n) \leftarrow 0$
3: **para** $k \leq p$ **haga**
4:  **para** $i \leq m$ **haga**
5:  **para** $j \leq n$ **haga**
6:  **si** $k = 0$ **entonces**
7:  **si** $x_i = y_j$ **entonces**
8:  $L_0(i, j) \leftarrow L_0(i-1, j-1) + 1$
9:  **de lo contrario**
10:  $L_0(i, j) \leftarrow$ máx.$\{L_0(i-1, j), L_0(i, j-1)\}$
11:  **de lo contrario**
12:  **si** $x_i = z_k$ **y** $y_j = z_k$ **entonces**
13:  $L_k(i, j) \leftarrow L_{k-1}(i-1, j-1) + 1$
14:  **de lo contrario**
15:  **si** $x_i = y_j$ **entonces**
16:  $L_k(i, j) \leftarrow L_k(i-1, j-1)$
17:  **de lo contrario**
18:  $L_k(i, j) \leftarrow$ máx.$\{L_k(i-1, j), L_k(i, j-1)\}$
19: **retornar** $\mathcal{T} \leftarrow L_r(m, n)$ {Tamaño del CLCS}

## 4.3 Algoritmo de Chin *et al.*

Chin *et al.* [8] lograron igual complejidad de tiempo al algoritmo de Peng para solucionar el problema del CLCS (véase Ecuación 4.3). La diferencia presentada en la solución de Chin *et al.* es la demostración de usar el algoritmo de Hirschberg (véase Sección 3.4) reduciendo la complejidad espacial a $O(mr)$.

Según la Ecuación 4.3, $L_k(i, j)$ es $\mathcal{L}_z(x, y)$ para las secuencias $x_{1..i}$, $y_{1..j}$ y $z_{1..k}$ y usando la misma ecuación sea $L^*_k(i, j)$ como $\mathcal{L}^*_z(x, y)$ para las secuencias $x^*_{1..i}$, $y^*_{1..j}$ y $z^*_{1..k}$ representando las secuencias invertidas de $x$, $y$ y $z$, respectivamente.

El CLCS para las secuencias $x$, $y$ y $z$ consiste en dividir la secuencia $x$ en dos partes. Luego, se encuentra la matriz $L$ para la primera mitad de la secuencia $x$, $y$ y la secuencia de restricción $z$ *i.e.*, $L_{z_{1..k}}(x_{1..\lceil \frac{m}{2} \rceil}, y_{1..n})$ y $L^*_{z_{1..k}}(x_{\lceil \frac{m}{2} \rceil + 1..m}, y_{1..n})$ para la segunda mitad de la secuencia $x$, $y$ y la secuencia de restricción $z$. Entonces, $\mathcal{L}_z(x, y)$ será:

$$t = \text{máx.}_{\substack{0 \leq j \leq n \\ 0 \leq k \leq r}} \left\{ L_k(x_{\lceil \frac{m}{2} \rceil}, y_j) + L^*_{k+1}(x_{\lceil \frac{m}{2} \rceil + 1}, y_{j+1}) \right\} \tag{4.5}$$

Donde $t$ será máximo para algún $j'(0 \leq j' \leq n)$ y $k'(0 \leq k' \leq r)$, entonces se divide el problema original en dos subproblemas: $\mathcal{L}_{z_{1..k'}}(x_{1..\lceil \frac{m}{2} \rceil}, y_{1..j'})$ y $\mathcal{L}_{z_{k'+1..r}}(x_{\lceil \frac{m}{2} \rceil + 1..m}, y_{j'+1..n})$.

Estos dos subproblemas son resueltos recursivamente. La solución al problema original consiste en concatenar la solución del primer subproblema con la solución del segundo subproblema. El algoritmo en detalle puede verse en Algoritmo 4.3.

La Figura 4.5 representa el algoritmo propuesto por Chin *et al.* y la manera de la reducción de complejidad en espacio $O(mr)$ para solucionar el problema del CLCS con el uso del algoritmo de Hirschberg (véase Sección 3.4) encontrando el punto medio de una "curva" del CLCS.

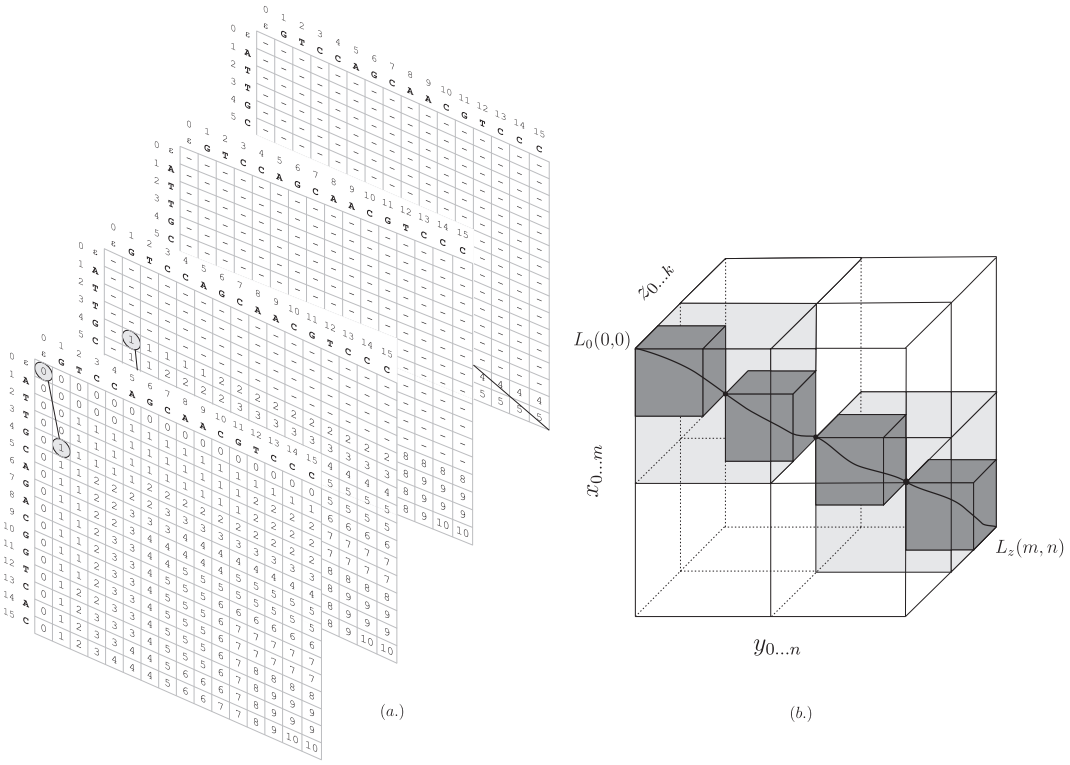


FIGURA 4.5: En (*a.*) se representa el algoritmo propuesto por Chin *et al.* El algoritmo de Chin *et al.* plantea una complejidad de tiempo igual al algoritmo de Peng. En (*b.*) se representa el uso del algoritmo de Hirschberg (véase Sección 3.4) en la solución del problema del CLCS.

## 4.4 Algoritmo de Arslan y Eğecioğlu

El trabajo de Arslan y Eğecioğlu [2] simplificó la recurrencia de Tsai [30] obteniendo la Ecuación 4.6. Aunque la Ecuación 4.6 calcula de forma diferente el CLCS con respecto a la ecuación de Peng, el algoritmo de Arslan y Eğecioğlu (véase Algoritmo 4.6) mantiene la complejidad de tiempo y espacio en $O(mnr)$.

**Algoritmo 4.3** Algoritmo de Chin *et al.* para encontrar el CLCS

**Requiere:** $Chin(x, y, z, m, n, r, \mathcal{S}, p)$
1: **si** $n = 0$ **entonces**
2: $\quad p \leftarrow 0$
3: **de lo contrario**
4: $\quad$ **si** $m = 1$ **entonces**
5: $\quad\quad$ **si** $\exists j$ con $x_i = y_j$ **entonces**
6: $\quad\quad\quad p \leftarrow 1$
7: $\quad\quad\quad \mathcal{S}_1 \leftarrow x_1$
8: $\quad\quad$ **de lo contrario**
9: $\quad\quad\quad p \leftarrow 0$
10: $\quad$ **de lo contrario**
11: $\quad\quad i \leftarrow \lceil \frac{m}{2} \rceil$
12: $\quad\quad L(x_{1..\lceil \frac{m}{2} \rceil}, y_{1..n}, z_{1..r})$
13: $\quad\quad L(x^*_{1..\lceil \frac{m}{2} \rceil}, y^*_{1..n}, z^*_{1..r})$
14: $\quad\quad t \leftarrow \text{máx.}_{\substack{0 \le j \le n \\ 0 \le k \le r}} \{L(x_{1..\lceil \frac{m}{2} \rceil}, y_{1..n}, z_{1..r}) + L(x^*_{\lceil \frac{m}{2} \rceil + 1..m}, y^*_{1..n}, z^*_{1..r})\}$
15: $\quad\quad Chin(x_{1..i}, y_{1..j}, z_{1..k}, i, j, k, \mathcal{S}, a)$
16: $\quad\quad Chin(x_{i+1..m}, y_{j+1..n}, z_{k+1..r}, m - i, n - j, r - k, \mathcal{D}, b)$
17: $\quad\quad p \leftarrow a + b$
18: $\quad\quad$ **para** $\ell \leftarrow 1 \le b$ **haga**
19: $\quad\quad\quad \mathcal{S}_{a+\ell} \leftarrow \mathcal{D}_\ell$
20: **retornar** $\mathcal{S}$ {Una secuencia CLCS}

El CLCS de las secuencias $x$, $y$ y $z$, expresado como $\mathcal{L}_z(x, y)$, se obtiene en $L_r(m, n)$.

$$
\begin{aligned}
&L_k(i, 0) = 0, 0 \le i \le m \text{ y } 0 \le k \le r \\
&L_k(0, j) = 0, 0 \le j \le n \text{ y } 0 \le k \le r \\
&L_k(i, j) = \text{máx.} \left\{ L'_k(i, j), L_k(i-1, j), L_k(i, j-1) \right\}, \\
&L'_k(i, j) = \text{máx.} \left\{ L''_k(i, j), L'''_k(i, j) \right\} \qquad (4.6) \\
&L''_k(i, j) = \begin{cases} L_{k-1}(i-1, j-1) + 1 & \text{si } (k = 0 \text{ o } (k > 0 \\ & \text{y } L_{k-1}(i-1, j-1) > 0)) \\ & \text{y } x_i = y_j = z_k, \\ 0 & \text{en otro caso} \end{cases} \\
&L'''_k(i, j) = \begin{cases} L_k(i-1, j-1) + 1 & \text{si } (k = 0 \text{ o} \\ & L_k(i-1, j-1) > 0) \\ & \text{y } x_i = y_j, \\ 0 & \text{en otro caso} \end{cases} \\
&\text{donde } 1 \le i \le m, 1 \le j \le n \text{ y } 0 \le k \le r
\end{aligned}
$$

**Algoritmo 4.4** Algoritmo $f_1$ (Algoritmo de Arslan)

**Requiere:** $(i, j, k, b)$
1: **si** $(k = 1$ **o** $(k > 1$ **y** $b > 1))$ **y** $(x_i = z_k$ **y** $y_j = z_k)$ **entonces**
2: **retornar** $b + 1$
3: **de lo contrario**
4: **retornar** $0$

**Algoritmo 4.5** Algoritmo $f_2$ (Algoritmo de Arslan)

**Requiere:** $(i, j, k, a)$
1: **si** $(k = 0$ **o** $a > 0)$ **y** $(x_i = y_j)$ **entonces**
2: **retornar** $a + 1$
3: **de lo contrario**
4: **retornar** $0$

**Algoritmo 4.6** Algoritmo de Arslan para encontrar el CLCS

**Requiere:** $(x, y, z, m, n, r)$
1: $k \leftarrow 0, i \leftarrow 1, j \leftarrow 1$
2: $L_0(1..m, 0) \leftarrow 0, L_0(0, 1..n) \leftarrow 0$
3: **para** $k \leq p$ **haga**
4: **para** $i \leq m$ **haga**
5: **para** $j \leq n$ **haga**
6: $a \leftarrow L_k(i - 1, j - 1)$
7: **si** $k > 0$ **entonces**
8: $b \leftarrow L_{k-1}(i - 1, j - 1)$
9: $L_k(i, j) \leftarrow$ máx.$\{f_1(i, j, k, b), f_2(i, j, k, a), L_k(i - 1, j), L_k(i, j - 1)\}$
10: **retornar** $\mathcal{T} \leftarrow L_r(m, n)$ {Tamaño del CLCS}

## 4.5 Algoritmo de Wang

Wang [31] observó lo innecesario de calcular toda la matriz de programación dinámica $L$ (véase Ecuación 4.2), debido a algunas de las áreas de la matriz que no impactan sobre el resultado final obtenido en $L_r(m, n)$. Las propiedades observadas para reducir el número de cálculos son:

- ⋆ Si $L_{k-1}(i, j) = L_k(i, j)$ entonces $L_{k-1}(i, j)$ nunca impacta sobre el resultado final
- ⋆ Si $L_k(i, j) = -\infty$ entonces los valores de $L_a(i, j)$ para $k < a \leq r$ son también $-\infty$

Entonces, en lugar de utilizar $r+1$ matrices $L$ de tamaño $(m+1) \times (n+1)$, el algoritmo utiliza una matriz $L$ de tamaño $(m + 1) \times (n + 1)$ donde para cada celda $L_0(i, j)$ $(0 \leq i \leq m,\ 0 \leq j \leq n)$ almacena una lista de únicamente las celdas necesarias para calcular $(i, j)$ en cada $k$.

Para aplicar las propiedades anteriormente mencionadas, el algoritmo de Wang construye la matriz $L_{ij}$, $0 \leq i \leq m$ y $0 \leq j \leq n$, como el conjunto de todos los elementos $(k, \mathcal{L}(i,j))$ para todo $0 \leq k \leq r$, donde el elemento $(w, \mathcal{L}(i,j))$ es eliminado, $0 \leq w \leq r$, si $\mathcal{L}_w(i,j) = -\infty$ o si hay otro elemento $(w+1, \mathcal{L}_{w+1}(i,j))$ donde $\mathcal{L}_w(i,j) = \mathcal{L}_{w+1}(i,j)$.

El algoritmo 4.7 representa el caso para $x_{1..m} = y_{1..n}$, el algoritmo 4.8 representa el caso para $x_{1..m} \neq y_{1..n}$ y el algoritmo 4.9 es la recuperación de un CLCS, algoritmos necesarios para el algoritmo completo de Wang (Algoritmo 4.10).

**Algoritmo 4.7** Algoritmo $Caso_1$ (Algoritmo de Wang)

**Requiere:** $(L_1)$
1: **mientras que** $L_1 \neq nulo$ **haga**
2: $(a,b) \in L_1$
3: **si** $a \neq r$ **y** $x_i = z_{a+1}$ **entonces**
4: añadir $(a+1, b+1)$ en $L$
5: eliminar $(a,b)$ de $L_1$
6: **de lo contrario**
7: **si** no $\exists(a,c)$ en $L_1$ **y** $c > b+1$ **entonces**
8: añadir $(a, b+1)$ en $L$
9: eliminar $(a,b)$ de $L_1$
10: **retornar** $L$

**Algoritmo 4.8** Algoritmo $Caso_2$ (Algoritmo de Wang)

**Requiere:** $(L_2, L_3)$
1: **mientras que** $L_2 \neq nulo$ **y** $L_3 \neq nulo$ **haga**
2: $(c,d) \in L_2, (e,f) \in L_3$
3: **si** $c = e$ **entonces**
4: añadir $(c,$ máx.$\{d,f\})$ en $L$
5: eliminar $(c,d)$ de $L_2$, $(e,f)$ de $L_3$
6: **de lo contrario**
7: **si** $c > e$ **entonces**
8: **si** $d < f$ **entonces**
9: añadir $(e,f)$ en $L$
10: eliminar $(e,f)$ de $L_3$
11: **de lo contrario**
12: **si** $d > f$ **entonces**
13: añadir $(c,d)$ en $L$
14: eliminar $(c,d)$ de $L_2$
15: **si** $L_2 \neq nulo$ **entonces**
16: añadir $L_2$ en $L$
17: **de lo contrario**
18: añadir $L_3$ en $L$
19: **retornar** $L$

La complejidad del algoritmo es $O(mnr)$ en tiempo y espacio, se debe por calcular la matriz $L_{ij}$ $(1 \leq i \leq m, 1 \leq j \leq n)$ y para cada celda de la matriz se tiene como

**Algoritmo 4.9** Algoritmo $RCLCS$ (Algoritmo de Wang)

**Requiere:** $RCLCS(i, j, k, p)$
1: **si** $p \neq 0$ **entonces**
2:  **si** $x_i = y_j$ **y** $y_j = z_k$ **entonces**
3:  $s \leftarrow x_i + s$
4:  $RCLCS(i-1, j-1, k-1, p-1)$
5:  **de lo contrario**
6:  **si** $x_i = y_j$ **y** $y_j \neq z_k$ **entonces**
7:  $s \leftarrow x_i + s$
8:  $RCLCS(i-1, j-1, k, p-1)$
9:  **de lo contrario**
10:  **si** $\exists(k, p)$ en $L(i-1, j)$ **entonces**
11:  $RCLCS(i-1, j, k, p)$
12:  **de lo contrario**
13:  $RCLCS(i, j-1, k, p)$
14: **retornar** $s$

**Algoritmo 4.10** Algoritmo de Wang para encontrar el CLCS

**Requiere:** $(x, y, z, m, n, r)$
1: $L(i, 0) \leftarrow L(0, j) \leftarrow 0$
2: **para** $i \leftarrow 1 \leq m$ **haga**
3:  **para** $j \leftarrow 1 \leq n$ **haga**
4:  **si** $x_i = y_j$ **entonces**
5:  $L(i, j) = Caso_1(L(i-1, j-1))$
6:  **de lo contrario**
7:  $L(i, j) = Caso_2(L(i-1, j), L(i, j-1))$
8: **si** $\exists\ r$ en $L(m, n)$ **entonces**
9:  $\mathcal{T} \leftarrow L_r(m, n)$ {Tamaño del CLCS}
10:  **para** $i \leftarrow 1 \leq m$ **haga**
11:  **para** $j \leftarrow 1 \leq n$ **haga**
12:  $\mathcal{S} = RCLCS(i, j, r, \mathcal{T})$
13: **retornar** $\mathcal{S}$ {Una secuencia CLCS}

máximo $r+1$ elementos. Estos elementos se pueden obtener empleando $O(r)$ al evaluar el elemento $L_{i-1,j-1}$ o los elementos $L_{i,j-1}$ y $L_{i-1,j}$ solo una vez.

**Ejemplo 4.4.** La Figura 4.6 presenta el $\text{LCS}_z(x, y)$ por el algoritmo de Wang para las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`".

De la misma manera He y Arslan [15] redujeron el número de cálculos y propusieron un método para resolver el problema del CPSA[1] usando espacio lineal.

La idea consiste en considerar el problema como la ruta óptima desde el punto $(0, 0, 0)$ hasta el punto $(m, n, r)$ en un grafo originado a partir de una matriz tridimensional.

[1] En la sección 2.5 se mostró como el CPSA es un caso especial del CLCS

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ε | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 0 | ε | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) |
| 1 | **A** | (0-0) | (0-0) | (0-0) | (0-0) | (0-0) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) |
| 2 | **T** | (0-0) | (0-0) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-2) | (0-2) | (0-2) | (0-2) |
| 3 | **T** | (0-0) | (0-0) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-1) | (0-2) | (0-2) | (0-2) | (0-2) |
| 4 | **G** | (0-0) | (1-1) | (1-1) | (1-1) | (1-1) | (1-1) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) | (1-2) |
| 5 | **C** | (0-0) | (1-1) | (1-1) | (1-2) | (1-2) | (1-2) | (1-2) | (1-3) | (1-3) | (1-3) | (1-3) | (1-3) | (1-3) | (1-3) | (1-3) | (1-3) |
| 6 | **A** | (0-0) | (1-1) | (1-1) | (1-2) | (1-2) | (1-3) | (1-3) | (1-3) | (1-4) | (1-4) | (1-4) | (1-4) | (1-4) | (1-4) | (1-4) | (1-4) |
| 7 | **G** | (0-0) | (1-1) | (1-1) | (1-2) | (1-2) | (1-3) | (1-4) | (1-4) | (1-4) | (1-4) | (1-4) | (1-5) | (1-5) | (1-5) | (1-5) | (1-5) |
| 8 | **A** | (0-0) | (1-1) | (1-1) | (1-2) | (1-2) | (1-3) | (1-4) | (1-4) | (1-5) | (1-5) | (1-5) | (1-5) | (1-5) | (1-5) | (1-5) | (1-5) |
| 9 | **C** | (0-0) | (1-1) | (1-1) | (1-2) | (1-3) | (1-3) | (1-4) | (1-5) | (1-5) | (1-5) | (1-6) | (1-6) | (1-6) | (1-6) | (1-6) | (1-6) |
| 10 | **G** | (0-0) | (1-1) | (1-1) | (1-2) | (1-3) | (1-3) | (1-4) | (1-5) | (1-5) | (1-5) | (1-6) | (1-7) | (1-7) | (1-7) | (1-7) | (1-7) |
| 11 | **G** | (0-0) | (1-1) | (1-1) | (1-2) | (1-3) | (1-3) | (1-4) | (1-5) | (1-5) | (1-5) | (1-6) | (1-7) | (1-7) | (1-7) | (1-7) | (1-7) |
| 12 | **T** | (0-0) | (1-1) | (2-2) | (2-2) | (1-3)<br>(2-2) | (1-3)<br>(2-2) | (1-4)<br>(2-2) | (1-5)<br>(2-2) | (1-5)<br>(2-2) | (1-5)<br>(2-2) | (1-6)<br>(2-2) | (1-7)<br>(2-2) | (2-8) | (2-8) | (2-8) | (2-8) |
| 13 | **C** | (0-0) | (1-1) | (2-2) | (2-3) | (2-3) | (2-3) | (1-4)<br>(2-3) | (1-5)<br>(2-3) | (1-5)<br>(2-3) | (1-5)<br>(2-3) | (1-6)<br>(2-3) | (1-7)<br>(2-3) | (2-8) | (2-9) | (2-9) | (2-9) |
| 14 | **A** | (0-0) | (1-1) | (2-2) | (2-3) | (2-3) | (3-4) | (3-4) | (1-5)<br>(3-4) | (1-6)<br>(3-4) | (1-6)<br>(3-4) | (1-6)<br>(3-4) | (1-7)<br>(3-4) | (2-8)<br>(3-4) | (2-9)<br>(3-4) | (2-9)<br>(3-4) | (2-9)<br>(3-4) |
| 15 | **C** | (0-0) | (1-1) | (2-2) | (2-3) | (2-4) | (3-4) | (3-4) | (3-5) | (1-6)<br>(3-5) | (1-6)<br>(3-5) | (1-7)<br>(3-5) | (1-7)<br>(3-5) | (2-8)<br>(3-5) | (2-9)<br>(3-5) | (2-10)<br>(3-5) | (2-10)<br>(3-5) |

FIGURA 4.6: Dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". En cada celda existe una lista de valores. Se usa ( – ) para mostrar el índice en $z$ y la correspondiente longitud de la subsecuencia común más larga restringida. La celda $L(m, n)$ contiene el $\mathcal{L}_z(x, y)$ en el último valor de la lista de esta celda, para el ejemplo es (3–5), es decir, $|z| = 3$ y $\mathcal{L}_z(x, y) = 5$.

La matriz tridimensional está formada por una matriz cuadrada (capa) de los índices $(0 \leq i \leq m, 0 \leq j \leq n)$ de las secuencias $x$ y $y$ por cada índice $(0 \leq k \leq r)$ de $z$, así la ruta del CLCS atraviesa cada capa donde $x_i = y_j = z_k$ (puntos de entrada de la capa $k$) satisfaciendo $z_k$ en la capa $k$.

Si se consideran dos capas adyacentes $(k-1)$ y $k$, para todos los puntos de entrada sobre la capa $k$ se tiene la longitud del CLCS para estos puntos es $L_{k-1}(i-1, j-1)+1$ y como $x_i = y_j$, el valor de la celda de $L_{k-1}(i, j)$ en la capa $(k-1)$ es también $L_{k-1}(i-1, j-1)+1$. Esto quiere decir de que todos los puntos de entrada en la capa $k$ tienen el mismo valor en la capa $(k-1)$. Los puntos de entrada en la capa $k$ pueden ser vistos como puntos de salida en la capa $(k-1)$. Lo anterior simboliza la ruta del CLCS deja la capa $(k-1)$ en un punto de salida y entra en la capa $k$ en el mismo punto sin cambiar su longitud.

Entonces, cada capa tiene un conjunto de puntos de entrada y un conjunto de puntos de salida, excepto la capa $k = 0$ que solo tiene un punto de entrada $(0, 0, 0)$ y la capa $k = r$ que solo tiene un punto de salida $(m, n, r)$. Como la ruta del CLCS ingresa en la capa $k$ en algún punto de entrada y deja la capa en algún punto de salida resulta innecesario calcular toda la capa $k$. Así por cada capa se puede precalcular los posibles puntos por donde una ruta óptima puede entrar y salir determinando una única región posible. Esta región se delimita por el primer punto de entrada o el punto con los mínimos valores para $i$ y $j$ entre todos los puntos de entrada y el último punto del conjunto de los puntos de salida de la capa.

Cada capa puede ser calculada en $O(n)$ espacio usando el algoritmo de Hirschberg (véase Sección 3.4) más el espacio de almacenamiento de todos los puntos de entrada sobre la capa actual o los puntos de salida de la capa previa. Si $\alpha$ es el número máximo total de puntos de entrada para cualquier capa, entonces la complejidad de espacio es $O(n + \alpha)$. Usualmente $\alpha$ es más pequeño que $mn$ por lo tanto el algoritmo en la práctica es $O(n)$ en espacio.

## 4.6 Algoritmo de Deorowicz

Sebastian Deorowicz [12, 13] presentó un nuevo algoritmo (Algoritmo 4.11) usando el algoritmo de Hunt–Szymanski (véase Sección 3.3). La Ecuación 4.7 es la recurrencia utilizada, principalmente tiene en cuenta el cálculo de las celdas donde existen coincidencias.

$$
\begin{aligned}
&L_0(i, 0) = 0, 0 \leq i \leq m \\
&L_0(0, j) = 0, 0 \leq j \leq n \\
&L_k(i, 0) = -\infty, 0 \leq i \leq m \text{ y } 1 \leq k \leq r \\
&L_k(0, j) = -\infty, 0 \leq j \leq n \text{ y } 1 \leq k \leq r \\
&L_k(i, j) = \begin{cases} \text{máx.}_{\substack{0 \leq i' < i \\ 0 \leq j' < j \\ x_{i'} = y_{j'}}} L_{k-1}(i', j') + 1 & \text{si } i, j, k > 0 \\ & \text{y } x_i = y_j = z_k, \\ \text{máx.}_{\substack{0 \leq i' < i \\ 0 \leq j' < j \\ x_{i'} = y_{j'}}} L_k(i', j') + 1 & \text{si } i, j > 0, x_i = y_j \\ & \text{y } (k = 0 \text{ o } x_i \neq z_k) \end{cases} \\
&\text{donde } 1 \leq i \leq m, 1 \leq j \leq n \text{ y } 0 \leq k \leq r
\end{aligned}
\tag{4.7}
$$

La complejidad de tiempo del algoritmo $O(r(m\mathcal{L} + \mathcal{M}) + n)$ y la complejidad de espacio es $O(r\mathcal{M}+\text{máx.}(n, \sigma))$, donde $x$ y $y$ son las secuencias dadas de longitud $m$ y $n$, respectivamente, $r$ representa el tamaño de la secuencia de restricción $z$, $\mathcal{L}$ representa la longitud del LCS$(x, y)$, $\mathcal{M}$ el número total de coincidencias entre las secuencias $x$ y $y$ y $\sigma$ el cardinal del alfabeto.

**Ejemplo 4.5.** Las Figuras 4.7 y 4.8 presentan el LCS$_z(x, y)$ por el algoritmo de Deorowicz para las secuencias $x$ = "`ATTGCAGACGGTCAC`", $y$ = "`GTCCAGCAACGTCCC`" y $z$ = "`GTA`".

## 4.7 Algoritmo de Iliopoulos y Rahman

Iliopoulos y Rahman [20] utilizaron la ecuación de Arslan y Eğecioğlu (véase ecuación 4.6) y la transformaron en la Ecuación 4.8:

$$
\begin{aligned}
L_k(i,j) &= \text{máx.}\left\{L'_k(i,j), L''_k(i,j), L_k(i-1,j), L_k(i,j-1)\right\}, \\
\mathcal{V}_1 &= \text{máx.}_{\substack{1\le i'<i \\ 1\le j'<j \\ x_{i'}=y_{j'}}} \left\{L_{k-1}(i',j')\right\} \\
L'_k(i,j) &= \begin{cases} \mathcal{V}_1 + 1 & \text{si } (k = 1 \text{ o } (k > 1 \text{ y } \mathcal{V}_1 > 0)) \\ & \text{y } x_i = y_j = z_k, \\ 0 & \text{en otro caso} \end{cases} \\
\mathcal{V}_2 &= \text{máx.}_{\substack{1\le i'<i \\ 1\le j'<j \\ x_{i'}=y_{j'}}} \left\{L_k(i',j')\right\} \\
L''_k(i,j) &= \begin{cases} 1 & \text{si } (i = 1 \text{ o } j = 1) \text{ y } x_i = y_j, \\ \mathcal{V}_2 + 1 & \text{si } (k = 0 \text{ o } \mathcal{V}_2 > 0) \text{ y } x_i = y_j, \\ 0 & \text{en otro caso} \end{cases} \\
&\text{donde } 1 \le i \le m, 1 \le j \le n \text{ y } 0 \le k \le r
\end{aligned}
\tag{4.8}
$$

Usando la estructura *Bounded Heap* [6] los valores $\mathcal{V}_1$ y $\mathcal{V}_2$ pueden ser calculados con una complejidad de tiempo de $O(\log\log n)$ y espacio $O(n)$. Esta estructura de datos está basada en un árbol van Emde Boas[2] para lograr tal complejidad.

[2]También se conoce como cola de prioridad van Emde Boas o árbol vEB

**Algoritmo 4.11** Algoritmo de Deorowicz para encontrar el CLCS

**Requiere:** $(x, y, z, m, n, r)$
1: **para** $i \leftarrow 1 \leq \sigma$ **haga**
2: $V^n(i) \leftarrow 0$
3: **para** $i \leftarrow 1 \leq n$ **haga**
4: $V^p(x_i, V^n(x_i)) \leftarrow i;\ V^n(x_i) \leftarrow V^n(x_i) + 1$
5: **para** $k \leftarrow 0 \leq r$ **haga**
6: **si** $k = 0$ **entonces**
7: $L^a(0).\ell \leftarrow 0$
8: **de lo contrario**
9: $L^a(0).\ell \leftarrow -\infty$
10: $L^a(0).i \leftarrow -\infty;\ L^a(0).j \leftarrow -\infty$
11: $N^a \leftarrow 1;\ N^b \leftarrow 0$
12: **para** $j \leftarrow 1 \leq m$ **haga**
13: $L^b(N^b) \leftarrow L^a(0);\ N^b \leftarrow N^b + 1;\ p \leftarrow 1$
14: **para** $s \leftarrow 0 \leq V^n(y_j) - 1$ **haga**
15: $i \leftarrow V^p(y_j, s)$
16: **mientras que** $p < N^a$ **haga**
17: **si** $L^a(p).i \geq i$ **entonces**
18: **salir del mientras**
19: **de lo contrario**
20: **si** $L^b(N^b - 1).\ell < L^a(p).\ell$ **y** $L^a(p).\ell > 0$ **entonces**
21: $L^b(N^b) \leftarrow L^a(p);\ N^b \leftarrow N^b + 1$
22: $p \leftarrow p + 1$
23: **si** $k > 0$ **y** $y_j = z_k$ **entonces**
24: $v \leftarrow T(j, i);\ T(j, i) \leftarrow L^a(p - 1).\ell + 1$
25: **de lo contrario**
26: $v \leftarrow L^a(p - 1).\ell + 1;\ T(j, i) \leftarrow v$
27: $F_k(j, i).i \leftarrow L^a(p - 1).i;\ F_k(j, i).j \leftarrow L^a(p - 1).j$
28: **si** $L^b(N^b - 1).\ell < v$ **entonces**
29: $L^b(N^b).i \leftarrow i;\ L^b(N^b).j \leftarrow j;\ L^b(N^b).\ell \leftarrow v;\ N^b \leftarrow N^b + 1$
30: **mientras que** $p < N^a$ **y** $L^b(N^b - 1).\ell \geq L^a(p).\ell$ **haga**
31: $p \leftarrow p + 1$
32: **mientras que** $p < N^a$ **haga**
33: $L^b(N^b) \leftarrow L^a(p);\ N^b \leftarrow N^b + 1;\ p \leftarrow p + 1$
34: $L^a \leftarrow L^b;\ N^a \leftarrow N^b;\ N^b \leftarrow 0$
35: **retornar** $\mathcal{T} \leftarrow L^a(N^a - 1).\ell$ {Tamaño del CLCS}
36: $i \leftarrow L^a(N^a - 1).i;\ j \leftarrow L^a(N^a - 1).j;\ k \leftarrow r$
37: **para** $\mathcal{T} \geq 1$ **haga**
38: $\mathcal{S}(\mathcal{T}) \leftarrow y_j$
39: **si** $k > 0$ **y** $y_j = z_k$ **entonces**
40: $k \leftarrow k - 1;$
41: $i' \leftarrow F_k(j, i).i;\ j' \leftarrow F_k(j, i).j$
42: $i \leftarrow i';\ j \leftarrow j'$
43: **retornar** $\mathcal{S}$ {Una secuencia CLCS}

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | | | | | 1 | | | 1 | 1 | | | | | | |
| 2 | **T** | | 1 | | | | | | | | | | 2 | | | |
| 3 | **T** | | 1 | | | | | | | | | | 2 | | | |
| 4 | **G** | 1 | | | | | 2 | | | | | 2 | | | | |
| 5 | **C** | | | 2 | 2 | | | 3 | | | 3 | | | 3 | 3 | 3 |
| 6 | **A** | | | | | 3 | | | 4 | 4 | | | | | | |
| 7 | **G** | 1 | | | | | 4 | | | | | 5 | | | | |
| 8 | **A** | | | | | 3 | | | 5 | 5 | | | | | | |
| 9 | **C** | | | 2 | 3 | | | 5 | | | 6 | | | 6 | 6 | 6 |
| 10 | **G** | 1 | | | | | 4 | | | | | 7 | | | | |
| 11 | **G** | 1 | | | | | 4 | | | | | 7 | | | | |
| 12 | **T** | | 2 | | | | | | | | | | 8 | | | |
| 13 | **C** | | | 3 | 3 | | | 5 | | | 6 | | | 9 | 9 | 9 |
| 14 | **A** | | | | | 4 | | | 6 | 6 | | | | | | |
| 15 | **C** | | | 3 | 4 | | | 5 | | | 7 | | | 9 | 10 | 10 |

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | | | | | - | | | - | - | | | | | | |
| 2 | **T** | | - | | | | | | | | | | - | | | |
| 3 | **T** | | - | | | | | | | | | | - | | | |
| 4 | **G** | - | | | | | - | | | | | - | | | | |
| 5 | **C** | | | 2 | 2 | | | 3 | | | 3 | | | 3 | 3 | 3 |
| 6 | **A** | | | | | 3 | | | 4 | 4 | | | | | | |
| 7 | **G** | - | | | | | 4 | | | | | 5 | | | | |
| 8 | **A** | | | | | 3 | | | 5 | 5 | | | | | | |
| 9 | **C** | | | 2 | 3 | | | 5 | | | 6 | | | 6 | 6 | 6 |
| 10 | **G** | - | | | | | 4 | | | | | 7 | | | | |
| 11 | **G** | - | | | | | 4 | | | | | 7 | | | | |
| 12 | **T** | | 2 | | | | | | | | | | 8 | | | |
| 13 | **C** | | | 3 | 3 | | | 5 | | | 6 | | | 9 | 9 | 9 |
| 14 | **A** | | | | | 4 | | | 6 | 6 | | | | | | |
| 15 | **C** | | | 3 | 4 | | | 5 | | | 7 | | | 9 | 10 | 10 |

FIGURA 4.7: Representación del algoritmo de Deorowicz para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". Los números representan el cálculo sobre las celdas que son coincidencias y tenidas en cuenta para calcular el CLCS de las secuencias dadas. Las celdas con el símbolo '-' representan $-\infty$ y las celdas sin símbolo alguno no existen. La primera tabla corresponde a $k = 0$ y la segunda tabla a $k = 1$

.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | | | | | - | | | - | - | | | | | | |
| 2 | **T** | | - | | | | | | | | | | - | | | |
| 3 | **T** | | - | | | | | | | | | | - | | | |
| 4 | **G** | - | | | | | - | | | | | - | | | | |
| 5 | **C** | | | - | - | | | - | | | - | | | - | - | - |
| 6 | **A** | | | | | - | | | - | - | | | | | | |
| 7 | **G** | - | | | | | - | | | | | - | | | | |
| 8 | **A** | | | | | - | | | - | - | | | | | | |
| 9 | **C** | | | - | - | | | - | | | - | | | - | - | - |
| 10 | **G** | - | | | | | - | | | | | - | | | | |
| 11 | **G** | - | | | | | - | | | | | - | | | | |
| 12 | **T** | | - | | | | | | | | | | - | | | |
| 13 | **C** | | | 3 | 3 | | | 3 | | | 3 | | | 9 | 9 | 9 |
| 14 | **A** | | | | | 4 | | | 4 | 4 | | | | | | |
| 15 | **C** | | | 3 | 4 | | | 5 | | | 5 | | | 9 | 10 | 10 |

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **G** | **T** | **C** | **C** | **A** | **G** | **C** | **A** | **A** | **C** | **G** | **T** | **C** | **C** | **C** |
| 1 | **A** | | | | | - | | | - | - | | | | | | |
| 2 | **T** | | - | | | | | | | | | | - | | | |
| 3 | **T** | | - | | | | | | | | | | - | | | |
| 4 | **G** | - | | | | | - | | | | | - | | | | |
| 5 | **C** | | | - | - | | | - | | | - | | | - | - | - |
| 6 | **A** | | | | | - | | | - | - | | | | | | |
| 7 | **G** | - | | | | | - | | | | | - | | | | |
| 8 | **A** | | | | | - | | | - | - | | | | | | |
| 9 | **C** | | | - | - | | | - | | | - | | | - | - | - |
| 10 | **G** | - | | | | | - | | | | | - | | | | |
| 11 | **G** | - | | | | | - | | | | | - | | | | |
| 12 | **T** | | - | | | | | | | | | | - | | | |
| 13 | **C** | | | - | - | | | - | | | - | | | - | - | - |
| 14 | **A** | | | | | - | | | - | - | | | | | | |
| 15 | **C** | | | - | - | | | 5 | | | 5 | | | 5 | 5 | 5 |

FIGURA 4.8: Representación del algoritmo de Deorowicz para hallar $\mathcal{L}_z(x, y)$ dadas las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`". Los números representan el cálculo sobre las celdas que son coincidencias y tenidas en cuenta para calcular el CLCS de las secuencias dadas. Las celdas con el símbolo '-' representan $-\infty$ y las celdas sin símbolo alguno no existen. La primera tabla corresponde a $k = 2$ y la segunda tabla a $k = 3$

.

El algoritmo de Iliopoulos y Rahman (Algoritmo 4.12) utiliza las siguientes operaciones soportadas por la estructura *Bounded Heap*:

- ✓ Insertar $(\mathcal{H}, pos, val, dato)$: Inserta en la estructura $\mathcal{H}$ la posición $pos$ con el valor $val$ y la información asociada $dato$.
- ✓ IncrementarValor $(\mathcal{H}, pos, val, dato)$: Si $\mathcal{H}$ no contiene la posición $pos$, se ejecuta Insertar. De lo contrario, cambia los valores de las posiciones por el máximo valor entre $val$ y $val'$, donde $val'$ es el valor anterior.
- ✓ MaximoValor $(\mathcal{H}, pos)$: Retorna el elemento con el máximo valor entre todos los elementos de la estructura $\mathcal{H}$ con la posición más pequeña que $pos$. Si la estructura $\mathcal{H}$ no contiene algún elemento con la posición más pequeña que $pos$, retorna 0.

**Algoritmo 4.12** Algoritmo de Iliopoulos y Rahman para encontrar el CLCS

**Requiere:** $(\mathcal{M})$
1: $C.Val = 0$
2: $C.Sec = \epsilon$
3: $\mathcal{H}_0^{-1} = \epsilon$
4: $\mathcal{H}_0^0 = \epsilon$
5: **para** $i \leftarrow 1 \leq n$ **haga**
6:  **para** $k \leftarrow 0 \leq p$ **haga**
7:  $\mathcal{H}_i^k = \mathcal{H}_{i-1}^k$
8:  **para todo** $(i, j) \in \mathcal{M}_i$ **haga**
9:  $max_1 =$ MaximoVal $(\mathcal{H}_{i-1}^{k-1}, j)$
10:  $max_2 =$ MaximoVal $(\mathcal{H}_{i-1}^{k}, j)$
11:  $Val_2 = 0$
12:  **si** $((k = 0)$ **o** $(max_2 > 0))$ **entonces**
13:  $Val_2 = max_2$
14:  **si** $x_i = z_k$ **entonces**
15:  $Val_1 = 0$
16:  **si** $(k = 1)$ **o** $((k > 1)$ **y** $(max_1 > 0))$ **entonces**
17:  $Val_1 = max_1$
18:  $maxresult =$ máx. $\{Val_1, Val_2\}$
19:  $\mathcal{L}_k(i, j) = maxresult + 1$
20:  **si** $C.Val < \mathcal{L}_k(i, j)$ **entonces**
21:  $C.Val = \mathcal{L}_k(i, j)$
22:  $C.Sec = (i, j, k)$
23:  IncrementarValor $(\mathcal{H}_i^k, j, \mathcal{L}_k(i, j), (i, j, k))$
24:  Eliminar $\mathcal{H}_{i-1}^{k-1}$
25: **retornar** $C$ $\{\mathcal{T} \leftarrow C.Val$ Tamaño de CLCS y $\mathcal{S} \leftarrow C.Sec$ Una secuencia CLCS$\}$

**Ejemplo 4.6.** La Figura 4.9 representa como se construye el CLCS usando algoritmo de Iliopoulos y Rahman para las secuencias $x =$ "`ATTGCAGACGGTCAC`", $y =$ "`GTCCAGCAACGTCCC`" y $z =$ "`GTA`" para $k = 1$.

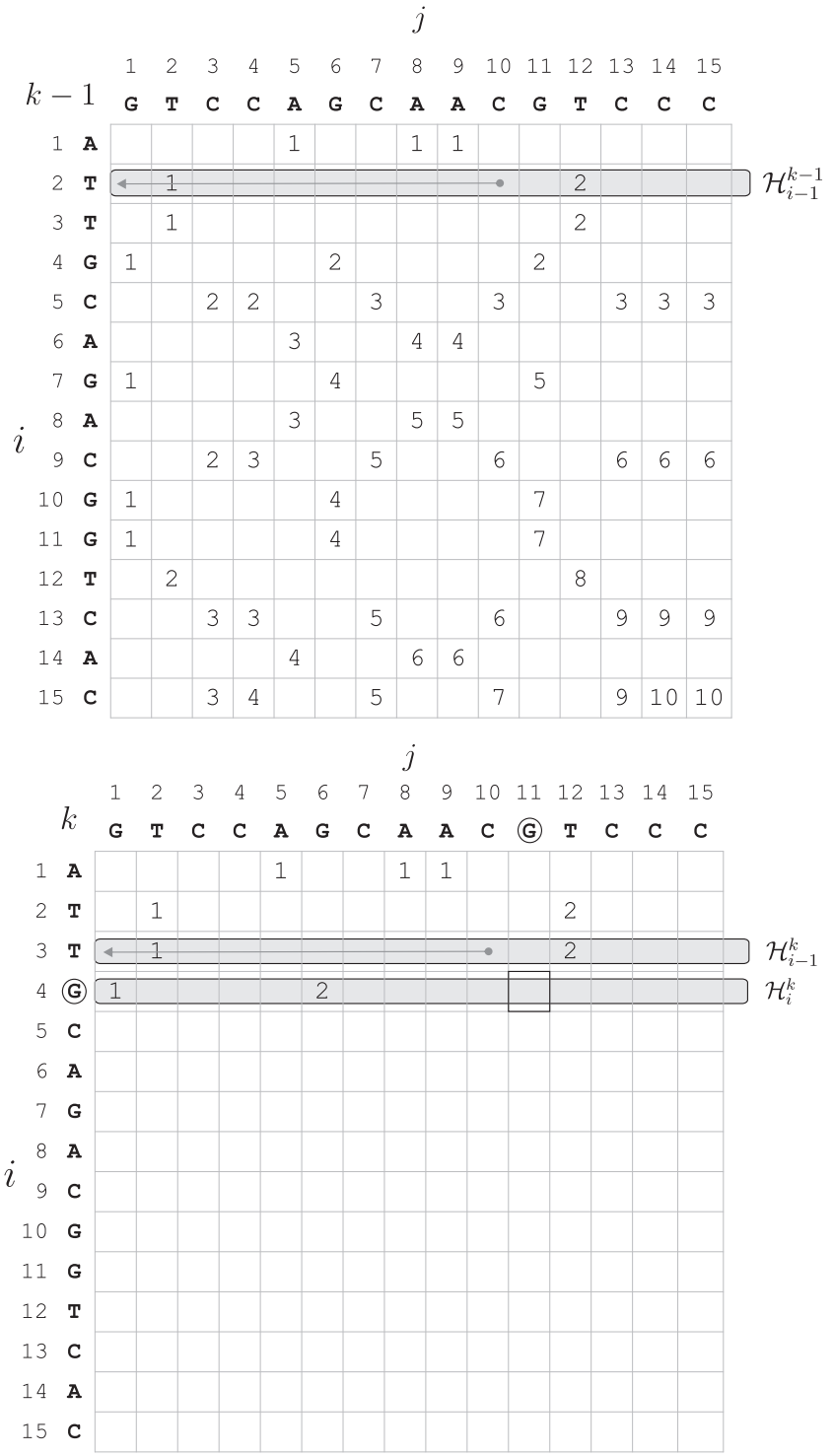


FIGURA 4.9: Representación del funcionamiento del algoritmo de Iliopoulos y Rahman sobre las secuencias $x =$ "ATTGCAGACGGTCAC", $y =$ "GTCCAGCAACGTCCC" y $z =$"GTA" para $k = 1$. Las áreas de color gris representan las estructuras *BoundedHeap* utilizadas por el algoritmo. Enmarcado aparece el valor a calcular (posición $L_1(4, 11)$). Las flechas representan la región donde se evalúa la función Maximovalor en las estructuras.

La complejidad del algoritmo en tiempo esta determinada por la etapa de preprocesamiento, para encontrar $\mathcal{M}$ con el orden requerido es de $O(\mathcal{R} \log \log n + n)$ [19]. En la etapa posterior del algoritmo es evidente observar que por cada coincidencia de $\mathcal{M}$ se utilizan las operaciones IncrementarValor y Maximovalor, estas operaciones son soportadas por la estructura en tiempo amortizado $O(\log \log n)$ [6]. Este procedimiento se repite $r$ veces. Así la complejidad total del algoritmo en tiempo está dada por $O(r\mathcal{R} \log \log n + n)$, teniendo en cuenta a $r\mathcal{R}$ como un valor más pequeño que $mnr$, de lo contrario en el peor caso la complejidad del algoritmo en tiempo es $\Omega(mnr)$. La complejidad en espacio del algoritmo también esta determinado por la etapa de preprocesamiento, dado por $\Theta(\text{máx.}\{\mathcal{R}, n\})$, debido a la etapa principal del algoritmo solo se necesita $O(n)$ para almacenar tres estructuras $BoundedHeap$ y la secuencia $z$.

# 5

# Nuevo Algoritmo

En este capítulo se presenta un nuevo algoritmo para encontrar la subsecuencia común más larga restringida aplicando el concepto de dominancia. El algoritmo propuesto en la práctica presenta un tiempo comparable con los demás algoritmos, pero, en algunos casos es peor a los ya existentes. El algoritmo propuesto no solo retorna la longitud del CLCS, sino también el conjunto de los CLCS dominantes. Adicionalmente, no es difícil mostrar como el algoritmo puede retornar todos los CLCS de un problema si la función de dominancia es ligeramente cambiada (*i.e.* usar '$<$' en vez de tener '$\leq$' en la función de dominancia).

**Definición 5.1 (*Dominancia entre dos subsecuencias comunes*).** Sea $x$ y $y$ dos secuencias ($m \leq n$) y sea $P$ y $Q$ dos subsecuencias comunes de $h$–longitud de $x$ y $y$, con $P$ y $Q$ ocurriendo en $x$ en los índices $(i_{x_1} \ldots i_{x_h})$ y $(j_{x_1} \ldots j_{x_h})$, respectivamente y de la misma forma, $P$ y $Q$ ocurriendo en $y$ en los índices $(i_{y_1} \ldots i_{y_h})$ y $(j_{y_1} \ldots j_{y_h})$, respectivamente. $s_p$ y $s_q$ son variables binarias asociadas con las subsecuencias $P$ y $Q$, respectivamente. Entonces $P$ domina a $Q$, denotado por $P \prec Q$, si y solo si:

$$i_{xh} \leq j_{xh}, \;\; i_{yh} \leq j_{yh} \quad \text{y} \quad s_p < s_q$$

## 5.1 Algoritmo

El algoritmo se puede resumir en dos pasos:

⋆ ***Paso 1*** **[Calcular coincidencias y encontrar $\mathcal{D}_1$]** Para entender el objetivo de este paso, se puede iniciar pensando en una matriz inicial como la que se muestra en

la Figura 5.1 (a)($i$). En esta matriz las filas se representan por $x_i$, $1 \leq i \leq m$ y las columnas como $y_j$, $1 \leq j \leq n$. Se procede ahora a calcular todas las coincidencias por filas – en inglés *row–wise* – de abajo hacia arriba, de la siguiente forma: se recorre $x$ (de izquierda a derecha) creando $List_x[\beta]$ que contiene todas las posiciones $i$ de $x$ donde el símbolo $\beta$ fue encontrado. Luego, se recorre $y$ (de abajo hacia arriba) y por cada símbolo $\beta$ encontrado en la posición $j$, se une $j$ a todas las posiciones $i$ almacenadas en $List_x[\beta]$ y se adiciona la tripleta $\delta = (j, i, \beta)$ a $\Delta$. Hay que tener en cuenta que para cada $\delta \in \Delta$, se crea un apuntador $w_\delta$ (apunta a la primera posición de $z$ ($z_0$)) y una variable $s_\delta$ que toma el valor de cero si la componente $\beta$ de $\delta$ es igual a $z_0$ o uno en otro caso. Si $s_\delta = 0$ entonces, $w_\delta$ apuntará a $z_1$. Adicionalmente, $\Delta$ es usado para calcular el conjunto $\mathcal{D}_i$ de coincidencias dominantes.

⋆ ***Paso 2*** **[Encontrar las subsecuencias comunes dominantes]** La principal idea es usar $\mathcal{D}_{i-1}$ para calcular el conjunto $\mathcal{D}_i$ de subsecuencias comunes de $i$–longitud con la siguiente condición invariante de estabilidad: *Ningún elemento (subsecuencia común) en $\mathcal{D}_{i-1}$ domina a otro elemento en $\mathcal{D}_i$.* Para asegurar esta condición, la Definición 5.1 es usada para seleccionar los elementos pertenecientes a $\mathcal{D}_i$.

El Paso 1 se ejecuta una sola vez en el algoritmo, mientras el Paso 2 es parte de un proceso iterativo que finaliza cuando el conjunto $D_i$ no puede ser construido. Se puede ver como los elementos $\mathcal{D}_i$ son las coincidencias dominantes extraídas desde $\Delta$ y los elementos $\mathcal{D}_i$ con $i > 1$ son subsecuencias dominantes generadas al concatenar cada $d \in \mathcal{D}_{i-1}$ con los elementos $\delta \in \Delta$. Una concatenación de un $d$ y $\delta$, denotado como $d\delta$, es insertado en $\mathcal{D}_i$ si y solo si: $i$) la fila y la columna del último carácter de la subsecuencia $d$ es más pequeña que la fila y la columna $\delta$, respectivamente, $ii$) esta concatenación no es dominada por ningún elemento perteneciente a $\mathcal{D}_i$. Cada $d \in \mathcal{D}$ tiene un apuntador $w_d$ y una variable $s_d$. La variable $s_d$ tiene el valor de cero si la componente $\beta$ de $\delta$ es igual a la componente de $z$ apuntada por $w_{d-1}$, o uno de cualquier otro modo. Si $s_\delta = 0$ entonces, $w_\delta$ apuntará al próximo componente de $z$, o si $s_\delta = 1$ entonces $w_{d-1} = w_d$. Finalmente, $s_d$ es actualizado usando una operación lógica OR sobre $s_{d-1}$ y $s_d$. Una vez el Paso 2 se cumple satisfactoriamente en la iteración $i$, el algoritmo tiene ya construido el conjunto $\mathcal{D}_i$, donde todos los elementos $d \in \mathcal{D}$ son las subsecuencias comunes dominantes de $i$–longitud entre $x$ y $y$. Hay que tener en cuenta que en la iteración $i = r$ el algoritmo descubre que habrá una solución al problema CLCS si $w_d$ apunta a $z_{r+1}$ y no habrá una solución si $w_d$ apunta a otro elemento. Cuando el Paso 2 no se lleva a cabo en la

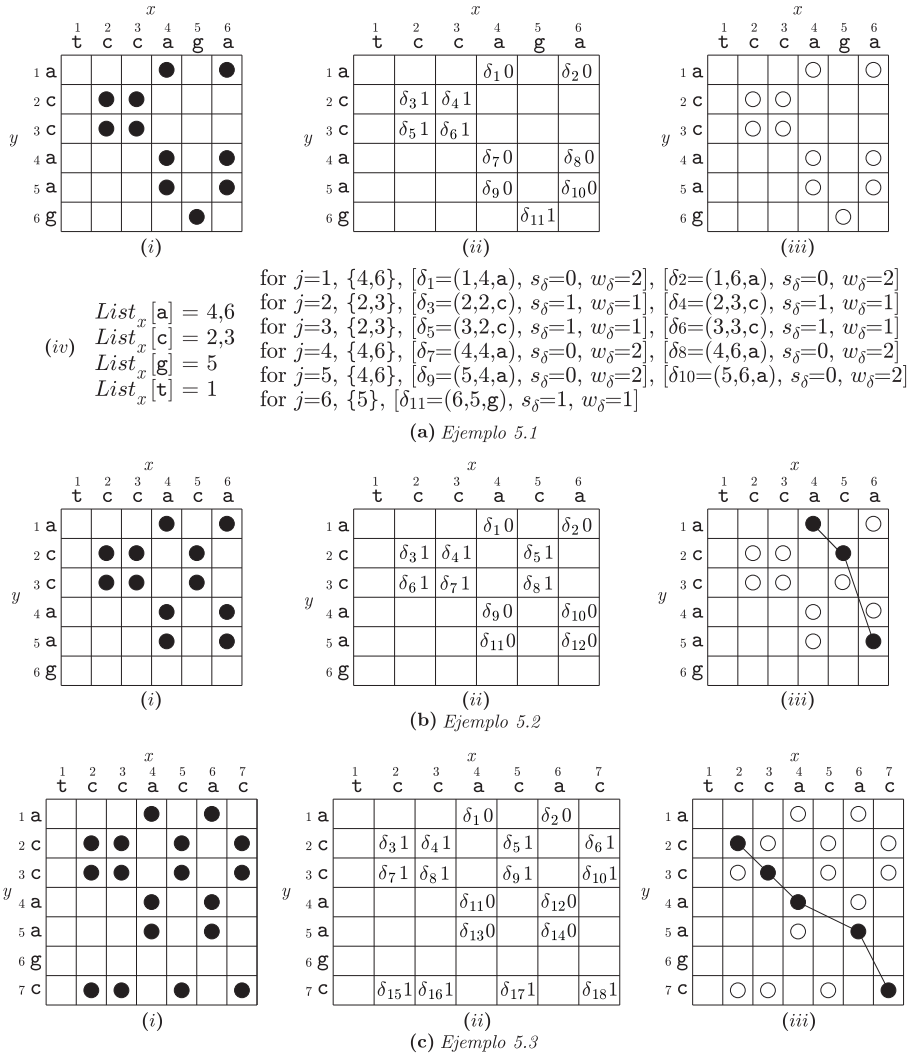


FIGURA 5.1: Ejemplo 5.1, 5.2 y 5.3.($i$) una matriz de representación de las secuencias $x$ y $y$, ($ii$) resultado de las subsecuencias comunes de tamaño 1, ($iii$) el CLCS reportado por el algoritmo propuesto, ($iv$) ruta de cálculo del paso 1.

iteración $i$ ($i.e.$, $\mathcal{D}_i = \emptyset$), todas las subsecuencias más largas comunes restringidas fueron almacenadas en el conjunto $\mathcal{D}$, donde al menos un $w_f$ se encuentra apuntando a $z_{r+1}$. El Algoritmo 5.1 resume todas las ideas anteriores. Para entender el algoritmo es necesario definir los siguientes procedimientos:

✓ *Procedimiento* Concatenar($d, \delta$): Retorna *true* si la subsecuencia $d$ puede ser concatenada con la subsecuencia $\delta$, ($i.e.$ la fila y la columna de el último carácter de la subsecuencia $d$ es más pequeño que la fila y la columna $\delta$, respectivamente), o *false* de cualquier otro modo.

✓ *Procedimiento* Dominancia($d\delta, \mathcal{D}$): Retorna *true* si la subsecuencia $d\delta$ es dominada por algún elemento $d \in \mathcal{D}$, o *false* de cualquier otro modo.

✓ *Procedimiento* NoDominancia($d\delta, \mathcal{D}$): Elimina todos los elementos de $\mathcal{D}$ que son dominados por $d$.

✓ *Procedimiento* Confiabilidad($\mathcal{D}$): Retorna *true* si al menos un $w_d$ apunta a $z_{r+1}$ en $\mathcal{D}$, o *false* de cualquier otro modo.

**Algoritmo 5.1** Nuevo Algoritmo CLCS

**Requiere:** $(x, y, z, \Sigma)$ $(m, n, r, \Delta, \mathcal{D}_1 \leftarrow 0, \beta \in \Sigma)$
**Retornar:** $\mathcal{D}$

1: **para** $i \leftarrow 1$ **a** $n$ **haga**
2: $List_X[x_i]$.add($i$)
3: **para** $j \leftarrow 1$ **a** $m$ **haga**
4: $\beta \leftarrow y_j$
5: **para todo** $k \in List_X[\beta]$ **haga**
6: $\delta \leftarrow (j,k,\beta)$;
7: **si** $\beta = z_1$ **entonces**
8: $w_\delta \leftarrow 2$; $s_\delta \leftarrow 0$;
9: **de lo contrario**
10: $w_\delta \leftarrow 1$; $s_\delta \leftarrow 1$;
11: $\Delta$.add($\delta$,$w_\delta$,$s_\delta$);
12: **si** $w_\delta = r + 1$ **entonces** $\eta \leftarrow i$
13: **si** !Dominado($\delta, \mathcal{D}_1$) **entonces**
14: $\mathcal{D}_1$.add($\delta, w_\delta, s_\delta$)
15: $i \leftarrow 2$
16: **mientras que** *true* **haga**
17: **para todo** $d \in \mathcal{D}_{i-1}$ **haga**
18: **para todo** $\delta \in \Delta$ **haga**
19: **si** Concatenar($d, \delta$) $AND$ !Dominado($d\delta, \mathcal{D}_i$) **entonces**
20: **si** $\delta_\beta = *w_d$ **entonces**
21: $w_d \leftarrow w_d + 1$; $s_d \leftarrow 0$
22: **de lo contrario**
23: $s_d \leftarrow 1$
24: **si** $w_d = r + 1$ **entonces** $\eta \leftarrow i$
25: $s_f \leftarrow s_d$ $OR$ $s_f$
26: $\mathcal{D}_i$.add($d\delta, w_d, s_d$);
27: NoDominado($d\delta, \mathcal{D}_i$)
28: **si** $i = r$ **entonces**
29: $flag \leftarrow$Confiabilidad($\mathcal{D}_i$)
30: **si** $flag = false$ **entonces retornar** $\emptyset$
31: **si** $\mathcal{D}_i = \emptyset$ **entonces**
32: **si** $i = 1$ **entonces retornar** $\Delta$
33: **de lo contrario** $\mathcal{D}_\eta$
34: $i \leftarrow i + 1$

## 5.2 Ejemplos

A continuación se muestran tres diferentes ejemplos profundamente estudiados en los que el algoritmo fue aplicado. Estos ejemplos en particular, representan diferentes situaciones en las cuales el problema del CLCS puede aparecer.

**Ejemplo 5.1 (No se encuentra el CLCS($x, y$)).** Dadas las secuencias $x =$"`tccaga`", $y =$"`accaag`" y $z =$"`ac`" (véase Figura 5.1 (**a**)($i - iv$)) muestra todas las coincidencias $\delta \in \Delta$ de $x$ y $y$ encontradas usando el procedimiento descrito en el Paso 1 (véase Algoritmo 5.1, líneas 1–11). Adicionalmente, el conjunto $\mathcal{D}_1$ es construido usando el Algoritmo 5.1 en la línea 13 y en la línea 14 y también se encontró la variable $s_\delta$ y el apuntador $w_\delta$. Para propósitos de claridad, el apuntador $w$ se muestra como un índice sobre $z$. La Figura 5.2 reporta todas las iteraciones durante el Paso 2. En la segunda iteración (*i.e.* $i = 2$), el conjunto $\mathcal{D}_2$ es construido usando el Algoritmo 5.1 desde la línea 17 a 27. En esta iteración la condición de igualdad entre la iteración y longitud de $z$ es evaluada *true* (véase Algoritmo 5.1, línea 28). Entonces, dado que el elemento $w_d = r + 1 = 3$ no fue encontrado, el algoritmo retorna un conjunto vacío $\emptyset$ (véase Algoritmo 5.1, línea 30). Es importante mencionar que el algoritmo es capaz de determinar confiablemente la solución del problema en una etapa temprana de ejecución (*i.e.* iteración $r$).

| $\Delta$ | | | | | |
|---|---|---|---|---|---|
| $\delta_1$ | a | [1/4](0,2) | $\delta_7$ | a | [4/4](0,2) |
| $\delta_2$ | a | [1/6](0,2) | $\delta_8$ | a | [4/6](0,2) |
| $\delta_3$ | c | [2/2](1,1) | $\delta_9$ | a | [5/4](0,2) |
| $\delta_4$ | c | [2/3](1,1) | $\delta_{10}$ | a | [5/6](0,2) |
| $\delta_5$ | c | [3/2](1,1) | $\delta_{11}$ | g | [6/5](1,1) |
| $\delta_6$ | c | [3/3](1,1) | | | |

| | $\mathcal{D}_1$ | | |
|---|---|---|---|
| iteration 1 | $\delta_1$ | a | [1/4](0,2) |
| | $\delta_2$ | a | [1/6](0,2) |
| | $\delta_3$ | c | [2/2](1,1) |
| | $\delta_7$ | a | [4/4](0,2) |
| | $\delta_8$ | a | [4/6](0,2) |
| | $\delta_9$ | a | [5/4](0,2) |
| | $\delta_{10}$ | a | [5/6](0,2) |

| | $\mathcal{D}_2 = \mathsf{Concatenate}(\Delta, \mathcal{D}_1)$ | | |
|---|---|---|---|
| iteration 2 | $\delta_1\delta_8$ | aa | [1,4/4,6](1,2) |
| | $\delta_1\delta_{11}$ | ag | [1,6/4,5](1,2) |
| | $\delta_3\delta_6$ | cc | [2,3/2,3](1,1) |

FIGURA 5.2: Ruta de cálculo de Ejemplo 5.1. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia.

**Ejemplo 5.2 (El CLCS($x, y$) encontrado no es igual al LCS($x, y$)).** Dadas las secuencias $x =$"`tccaca`", $y =$"`accaag`" y $z =$"`ac`" (véase Figura 5.1 (**b**)($i - ii$)) hace

referencia al Paso 1. La Figura 5.3 reporta todas las iteraciones del Paso 2 y la Figura 5.1 (**b**)(*iii*) ilustra el CLCS reportado por el algoritmo. En el Paso 1 (véase Algoritmo 5.1, líneas 1–11) se encontró las coincidencias $\delta \in \Delta$ de $x$,$y$ y las variables $s_\delta$ y el apuntador $w_\delta$ (véase Figura 5.1 (**b**)$(i - ii)$). Adicionalmente, el conjunto $\mathcal{D}_1$ es construido usando el Algoritmo 5.1 desde la línea 13 a 14. La Figura 5.3 reporta todas las iteraciones durante el Paso 2. En las próximas iteraciones (*i.e.* $i = 2, i = 3, i = 4$), los conjuntos $\mathcal{D}_2$, $\mathcal{D}_3$ y $\mathcal{D}_4$ son construidos usando el Algoritmo 5.1 desde la línea 17 a 27. En la iteración 5, el conjunto $\mathcal{D}_5$ es vacío, resultado de ejecutar el Algoritmo 5.1 líneas 17 a 27. Así, el algoritmo identifica el conjunto $\mathcal{D}_3$ como el último conjunto logrado donde $w_d = r+1$ (véase Algoritmo 5.1, línea 30). Específicamente, el algoritmo descubre en $\mathcal{D}_3$ una subsecuencia con un apuntador $w_d = r+1$. Esta subsecuencia representa $\text{LCS}_z(x, y)$. Para la subsecuencia reportada en $\mathcal{D}_4$ (`ccaa`), note como $z$ no es subsecuencia, así que no hay $w_{\mathcal{D}_4} = r+1$. Entonces, esta subsecuencia no puede ser reportada como el CLCS. Particularmente, el algoritmo es capaz de encontrar un CLCS, incluso si no es igual al $\text{LCS}(x, y)$, identificando el último conjunto donde se encontró $w_{\mathcal{D}_i} = r + 1$.

| $\Delta$ | | | | | |
|---|---|---|---|---|---|
| $\delta_1$ | a | [1/4](0,2) | $\delta_7$ | c | [3/3](1,1) |
| $\delta_2$ | a | [1/6](0,2) | $\delta_8$ | c | [3/5](1,1) |
| $\delta_3$ | c | [2/2](1,1) | $\delta_9$ | a | [4/4](0,2) |
| $\delta_4$ | c | [2/3](1,1) | $\delta_{10}$ | a | [4/6](0,2) |
| $\delta_5$ | c | [2/5](1,1) | $\delta_{11}$ | a | [5/4](0,2) |
| $\delta_6$ | c | [3/2](1,1) | $\delta_{12}$ | a | [5/6](0,2) |

iteration 1

| $\mathcal{D}_1$ | | |
|---|---|---|
| $\delta_1$ | a | [1/4](0,2) |
| $\delta_2$ | a | [1/6](0,2) |
| $\delta_3$ | c | [2/2](1,1) |
| $\delta_9$ | a | [4/4](0,2) |
| $\delta_{10}$ | a | [4/6](0,2) |
| $\delta_{11}$ | a | [5/4](0,2) |
| $\delta_{12}$ | a | [5/6](0,2) |

iteration 2

| $\mathcal{D}_2 = \text{Concatenate}(\Delta,\mathcal{D}_1)$ | | |
|---|---|---|
| $\delta_1\delta_5$ | ac | [1,2/4,5](0,3) |
| $\delta_1\delta_8$ | ac | [1,3/4,5](0,3) |
| $\delta_1\delta_{10}$ | aa | [1,4/4,6](1,2) |
| $\delta_3\delta_7$ | cc | [2,3/2,3](1,1) |

iteration 3

| $\mathcal{D}_3 = \text{Concatenate}(\Delta,\mathcal{D}_2)$ | | |
|---|---|---|
| $\delta_1\delta_5\delta_{10}$ | aca | [1,2,4/4,5,6](1,3) ✓ |
| $\delta_3\delta_7\delta_9$ | cca | [2,3,4/2,3,4](1,2) |

iteration 4

| $\mathcal{D}_4 = \text{Concatenate}(\Delta,\mathcal{D}_3)$ | | |
|---|---|---|
| $\delta_3\delta_7\delta_9\delta_{12}$ | ccaa | [2,3,4,5/2,3,4,6](1,2) |

FIGURA 5.3: Ruta de cálculo de Ejemplo 5.2. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia. El CLCS encontrado es señalado por el visto.

**Ejemplo 5.3 (El CLCS$(x, y)$ encontrado es igual al LCS$(x, y)$).** Dadas las secuencias $x =$"`tccacac`", $y =$"`accaagc`" y $z =$"`ac`", (véase Figura 5.1 (**c**)$(i - ii)$) hace referencia al Paso 1. La Figura 5.4 reporta todas las iteraciones del Paso 2 y la Figura 5.1 **c**$(iii)$ ilustra el CLCS reportado por el algoritmo. En el Paso 1 (véase Algoritmo 5.1, líneas 1–11) se encontraron las coincidencias $\delta \in \Delta$ de $x$,$y$ y las variables $s_\delta$ y el apuntador $w_\delta$ (véase Figura 5.1 **c**$(iii)$. La Figura 5.4 reporta los conjuntos $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, $\mathcal{D}_4$, $\mathcal{D}_5$ y $\mathcal{D}_6$. Note como el conjunto $\mathcal{D}_6$ es vacío una vez el ciclo principal termina. Así, el algoritmo identifica el conjunto $\mathcal{D}_5$ como el último conjunto donde $w_d = r + 1$ (véase Algoritmo 5.1, línea 24). La Figura 5.1 describe en una forma más sistemática el CLCS encontrado por el Algoritmo 5.1. En este ejemplo, el LCS$(x, y)$ es la misma subsecuencia reportada en $\mathcal{D}_5$ (`ccaac`). Note como $z$ es una subsecuencia de (`ccaac`), *i.e.* hay un $w_{\mathcal{D}_5} = r + 1$. Entonces, el LCS$(x, y)$ reportado es el mismo LCS$_z(x, y)$.

| $\Delta$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\delta_1$ | a | [1/4](0,2) | $\delta_7$ | c | [3/2](1,1) | $\delta_{13}$ | a | [5/4](0,2) |
| $\delta_2$ | a | [1/6](0,2) | $\delta_8$ | c | [3/3](1,1) | $\delta_{14}$ | a | [5/6](0,2) |
| $\delta_3$ | c | [2/2](1,1) | $\delta_9$ | c | [3/5](1,1) | $\delta_{15}$ | c | [7/2](1,1) |
| $\delta_4$ | c | [2/3](1,1) | $\delta_{10}$ | c | [3/7](1,1) | $\delta_{16}$ | c | [7/3](1,1) |
| $\delta_5$ | c | [2/5](1,1) | $\delta_{11}$ | a | [4/4](0,2) | $\delta_{17}$ | c | [7/5](1,1) |
| $\delta_6$ | c | [2/7](1,1) | $\delta_{12}$ | a | [4/6](0,2) | $\delta_{18}$ | c | [7/7](1,1) |

iteration 1

| $\mathcal{D}_1$ | | |
|---|---|---|
| $\delta_1$ | a | [1/4](0,2) |
| $\delta_2$ | a | [1/6](0,2) |
| $\delta_3$ | c | [2/2](1,1) |
| $\delta_{11}$ | a | [4/4](0,2) |
| $\delta_{12}$ | a | [4/6](0,2) |
| $\delta_{13}$ | a | [5/4](0,2) |
| $\delta_{14}$ | a | [5/6](0,2) |

iteration 2

| $\mathcal{D}_2 =$ Concatenate($\Delta$,$\mathcal{D}_1$) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\delta_1\delta_5$ | ac | [1,2/4,5](0,3) | $\delta_1\delta_{18}$ | ac | [1,7/4,7](0,3) | $\delta_{11}\delta_{18}$ | ac | [4,7/4,7](0,3) |
| $\delta_1\delta_6$ | ac | [1,2/4,7](0,3) | $\delta_2\delta_6$ | ac | [1,2/6,7](0,3) | $\delta_{12}\delta_{18}$ | ac | [4,7/6,7](0,3) |
| $\delta_1\delta_9$ | ac | [1,3/4,5](0,3) | $\delta_2\delta_{10}$ | ac | [1,3/6,7](0,3) | $\delta_{13}\delta_{17}$ | ac | [5,7/4,5](0,3) |
| $\delta_1\delta_{10}$ | ac | [1,3/4,7](0,3) | $\delta_2\delta_{18}$ | ac | [1,7/6,7](0,3) | $\delta_{13}\delta_{18}$ | ac | [5,7/4,7](0,3) |
| $\delta_1\delta_{12}$ | aa | [1,4/4,6](1,2) | $\delta_3\delta_8$ | cc | [2,3/2,3](1,1) | $\delta_{14}\delta_{18}$ | ac | [5,7/6,7](0,3) |
| $\delta_1\delta_{17}$ | ac | [1,7/4,5](0,3) | $\delta_{11}\delta_{17}$ | ac | [4,7/4,5](0,3) | | | |

iteration 3

| $\mathcal{D}_3 =$ Concatenate($\Delta$,$\mathcal{D}_2$) | | |
|---|---|---|
| $\delta_1\delta_5\delta_{10}$ | acc | [1,2,3/4,5,7](1,3) |
| $\delta_1\delta_5\delta_{12}$ | aca | [1,2,4/4,5,6](1,3) |
| $\delta_3\delta_8\delta_{11}$ | cca | [2,3,4/2,3,4](1,2) |

iteration 4

| $\mathcal{D}_4 =$ Concatenate($\Delta$,$\mathcal{D}_3$) | | |
|---|---|---|
| $\delta_1\delta_5\delta_{12}\delta_{18}$ | acac | [1,2,4,7/4,5,6,7](1,3) |
| $\delta_3\delta_8\delta_{11}\delta_{14}$ | ccaa | [2,3,4,5/2,3,4,6](1,2) |
| $\delta_3\delta_8\delta_{11}\delta_{17}$ | ccac | [2,3,4,7/2,3,4,5](1,3) |

iteration 5

| $\mathcal{D}_5 =$ Concatenate($\Delta$,$\mathcal{D}_4$) | | |
|---|---|---|
| $\delta_3\delta_8\delta_{11}\delta_{14}\delta_{18}$ | ccaac | [2,3,4,5,7/2,3,4,6,7](1,3) ✓ |

FIGURA 5.4: Ruta de cálculo de Ejemplo 5.3. Obsérvese como [ / ] se usa para mostrar los índices en $Y/X$ donde la subsecuencia ocurre, y ( , ) es usado para mostrar los valores de $s$ y $w$ de la subsecuencia. El CLCS encontrado es señalado por el visto.

## 5.3 Complejidad

El Paso 1 consiste en encontrar todas las coincidencias entre las secuencias $x$ y $y$. Esto se puede determinar fácilmente almacenando en $List_x[\beta]$ las apariencias de todos los símbolos del alfabeto en $x$. Esta lista puede ser implementada en $O(\text{máx.}(m, n))$ espacio y tiempo. El conjunto $\Delta$ puede ser implementado en $O(\mathcal{R})$ espacio y $O(m\sigma)$ tiempo. El Paso 2, por otra parte, es calculado en las líneas 17–27 (véase Algoritmo 5.1), este paso puede ser implementado en $O(\mathcal{L}|\Delta||\mathcal{D}|)$, donde $|\Delta|$ representa el número de coincidencias entre las secuencias $x$ y $y$, $|\mathcal{D}|$ es el tamaño del conjunto resultante después de aplicar el concepto de dominancia y $\mathcal{L}$ es la longitud del LCS calculado. Claramente la complejidad total del algoritmo es determinada por el Paso 2. Por lo tanto, la complejidad total de tiempo del algoritmo es limitada por $O(\mathcal{L}|\Delta||\mathcal{D}|)$ y la complejidad total de espacio es limitada por $O(|\Delta| + |\mathcal{D}|)$.

# 6

# Experimentación

En este capítulo se presenta la comparación experimental de los algoritmos. El objetivo de la experimentación es medir el tiempo de ejecución de los algoritmos mencionados en los capítulos anteriores (Capítulos 4 y 5).

Todos los algoritmos implementados tienen un nivel similar de optimización. Los resultados obtenidos ofrecen:

- ✓ Una representación gráfica del comportamiento práctico de cada algoritmo para casos específicos.
- ✓ Exploración de la influencia del tamaño de los alfabetos sobre el tiempo de ejecución de cada algoritmo.
- ✓ Exploración sobre la variación del tamaño de la secuencia de restricción en cada algoritmo.
- ✓ Exploración del comportamiento de los algoritmos cuando el tamaño de las secuencias aumenta y los tamaños de la secuencia de restricción y del alfabeto se mantienen.
- ✓ Los resultados de tiempo de ejecución para los algoritmos incluyen el cálculo del tamaño del CLCS y la recuperación de al menos una de las secuencias CLCS.
- ✓ Evaluación con secuencias generadas de forma aleatoria y secuencias extraídas de una colección de textos (`http://pizzachili.dcc.uchile.cl/index.html`).

Los algoritmos fueron implementados en C++ usando específicamente la librería STL – *Standard Template Library* –. La Tabla 6.1 resume los programas desarrollados. Los programas fueron compilados sin la opción de optimización en el compilador g++ versión 4.3.2. Las ejecuciones se realizaron en una máquina con sistema operativo Linux, procesador Intel Core(TM)2 Duo de 2GHz y 3GB de RAM.

Tabla 6.1: Nombre de los archivos C++

| N. | Algoritmo | Archivo C++ |
|---|---|---|
| *1* | Tsai | Tsai.cpp |
| *2* | Arslan y Eğecioğlu | Arslan.cpp |
| *3* | Chin *et al.* | Chin.cpp |
| *4* | Peng Chao-Li | Peng.cpp |
| *5* | Wang | Wang.cpp |
| *6* | Deorowicz | Deo.cpp |
| *7* | Iliopoulos y Rahman | Ilio.cpp |
| *8* | Nuevo algoritmo | NCLCS.cpp |

Algunos de los detalles relevantes en la implementación de los algoritmos son:

- ⋆ La implementación del árbol van Emde Boas o árbol vEB fue extraída del enlace: `http://www.itu.dk/people/kokholm/veb/`, posteriormente se modificó para el algoritmo de Iliopoulos y Rahman (Sección 4.7).
- ⋆ Se utilizó la estructura vector del STL de C++ debido al acceso aleatorio a elementos en complejidad constante y a las operaciones de inserción y eliminación de elementos al final del vector en tiempo constante amortizado[1]

## 6.1 Procedimiento

Los conjuntos de datos utilizados fueron de dos tipos: secuencias generadas de forma aleatoria uniforme y secuencias pertenecientes a una colección de textos. Las secuencias generadas de forma aleatoria uniforme pertenecen a alfabetos predefinidos de tamaño 4, 10, 20, 32, 48 y 64 caracteres y las secuencias extraídas de la colección de textos (`http://pizzachili.dcc.uchile.cl/texts.html`) pertenecen a los alfabetos de "*DNA*", "*PROTEINS*", "*PITCHES*" y "*ENGLISH*" (la Tabla 6.2 muestra sus características).

[1] Solter, Nicholas A. y Kleper, Scott J. *Professional C++ (Programmer to Programmer)*. Wrox Press Ltd. 2005.

Tabla 6.2: Colección de textos

| Colección | $\sigma$ | Mb. | $H_0$ | $H_1$ | $H_2$ |
|---|---|---|---|---|---|
| *DNA* | *16* | *50* | *1.982* | *1.935* | *1.920* |
| *PROTEINS* | *25* | *50* | *4.195* | *4.173* | *4.146* |
| *PITCHES* | *133* | *50* | *5.633* | *4.734* | *4.139* |
| *ENGLISH* | *215* | *100* | *4.529* | *3.606* | *2.922* |

Mb. = Tamaño en Megabytes de la colección.
$H_k$ = Entropía empírica de orden $k$, medida en número de bits por símbolo de entrada.

Para estimar el valor esperado de una medición asociada a la ejecución de un algoritmo, se ejecutó 30 veces cada uno y se tomó el promedio de las mediciones asociadas.

En la Figura 6.1 se resumen los valores utilizados por las variables en la experimentación.

| | | *Variables fijas* | $\sigma$ | *Variables fijas* | $r$ | *Variables fijas* | $m{=}n$ | *Variables fijas* | $n$ |
|---|---|---|---|---|---|---|---|---|---|
| *Tipos de secuencias* | *Aleatoria* | Figura 6.2 | | Figura 6.3 | | Figura 6.4 | | Figura 6.5 | |
| | | $r = 16$<br>$m = n$ (a.) *100* (b.) *300* (c.) *500* (d.) *1000* | *4*<br>*10*<br>*20*<br>*32*<br>*48*<br>*64* | $\sigma = 20$<br>$m = n$ (a.) *100* (b.) *300* (c.) *500* (d.) *1000* | *4*<br>*8*<br>*16*<br>*24*<br>*32*<br>*64* | $\sigma = 20$<br>$r =$ (a.) *8* (b.) *16* (c.) *24* (d.) *64* | *100*<br>*300*<br>*500*<br>*800*<br>*1000* | $r = 64$<br>$\sigma =$ (a.) *4* (b.) *20* (c.) *32* (d.) *64*<br>$m = 1000$ | *100*<br>*300*<br>*500*<br>*800*<br>*1000* |
| | *Colección de textos* | Figura 6.6 | | Figura 6.7 | | Figura 6.8 | | Figura 6.9 | |
| | | $r = 16$<br>$m = n$ (a.) *100* (b.) *300* (c.) *500* (d.) *1000* | *16*<br>*25*<br>*133*<br>*215* | $\sigma = 215$<br>$m = n$ (a.) *100* (b.) *300* (c.) *500* (d.) *1000* | *4*<br>*8*<br>*16*<br>*24*<br>*32*<br>*64* | $\sigma = 215$<br>$r =$ (a.) *8* (b.) *16* (c.) *24* (d.) *64* | *100*<br>*300*<br>*500*<br>*800*<br>*1000* | $r = 64$<br>$\sigma =$ (a.) *16* (b.) *25* (c.) *133* (d.) *215*<br>$m = 1000$ | *100*<br>*300*<br>*500*<br>*800*<br>*1000* |

Figura 6.1: Resumen de los valores utilizados por las variables en la experimentación.

## 6.2 Resultados

Los resultados obtenidos con el algoritmo de Tsai son demasiado grandes con respecto a los tiempos de los demás algoritmos. Por consiguiente el algoritmo de Tsai no se incluye en las gráficas de comportamiento porque hace perder el detalle de los demás algoritmos

debido a la escala. Sin embargo, gracias a estos resultados se deduce experimentalmente como resulta impráctico el algoritmo y se corrobora su bajo desempeño por la elevada complejidad teórica que tiene.

Una de las características de las secuencias extraídas de la colección de textos es el cambio de su entropía para cada una de las ejecuciones. Por eso a los resultados obtenidos para este tipo de pruebas se incluye el promedio de la entropía de las correspondientes secuencias (Figura 6.6 – Tabla A.1), (Figura 6.7 – Tabla A.2), (Figura 6.8 – Tabla A.3) y (Figura 6.9 – Tabla A.4).

Las soluciones reportadas para algunos de los algoritmos son iguales a 0 milisegundos, lo que construye curvas de comportamiento no visibles en las gráficas (véase Figuras 6.2 (a.), 6.3 (a.), 6.6 (a.), 6.7 (a.)).

Es importante identificar dos grupos de algoritmos. Estos grupos son: los algoritmos basados en "secuencias" (Tsai, Ars, Chin, Peng, Wang) y algoritmos basados en "coincidencias" (Deo, Ilio, NCLCS). Los primeros tienen una tendencia con respecto a la longitud de las secuencias de entrada y los segundos por su parte, se basan en las coincidencias encontradas entre las secuencias.

### 6.2.1 Resultados con secuencias generadas de forma aleatoria

En la Figura 6.2 se observa el comportamiento de los algoritmos con respecto al tamaño del alfabeto. En esta figura se presentan cuatro pruebas diferentes variando el tamaño de las secuencias. Es evidente notar para los algoritmos basados en "secuencias" como el tamaño del alfabeto mantiene una tendencia de tiempo constante. Para los algoritmos basados en "coincidencias" los tiempos decrecen debido a la reducción del número de coincidencias en la medida que el tamaño del alfabeto se hace mayor. Además, existe una relación de incidencia en el tiempo de ejecución entre el tamaño del alfabeto y el tamaño de las secuencias, *e.g.* el algoritmo de Wang (Wang) tiene un tiempo bajo con respecto a todos los demás algoritmos para secuencias de tamaño menor a 1000 pertenecientes a alfabetos de tamaño inferiores a 20 y su comportamiento empieza a ser desfavorable para secuencias de tamaño mayor a 300 pertenecientes a alfabetos superiores de tamaño 20. El algoritmo propuesto (NCLCS) muestra la tendencia de decrecimiento de los algoritmos basados en "coincidencias" pero no es el mejor para este caso. El algoritmo de Iliopoulos

(Ilio) para alfabetos de tamaño mayor a 20 independiente del tamaño de las secuencias fue el algoritmo con el mejor desempeño.

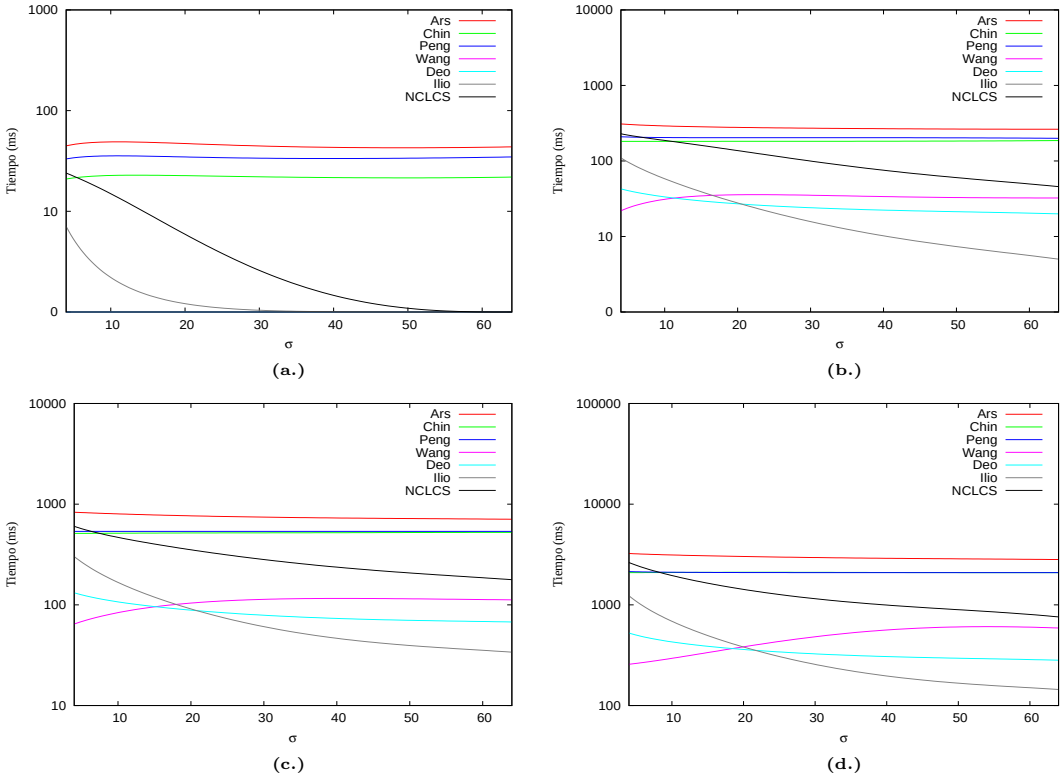


FIGURA 6.2: Comportamiento de los algoritmos variando el tamaño del alfabeto ($\sigma$), con las variables fijas $r = 16$ y **(a.)** $m = n = 100$, **(b.)** $m = n = 300$, **(c.)** $m = n = 500$ y **(d.)** $m = n = 1000$

La Figura 6.3 muestra la incidencia del tamaño de la secuencia de restricción en el comportamiento de los algoritmos. En las ejecuciones llevadas a cabo, se estableció como tamaño del alfabeto 20, debido a la importancia en la bioinformática del alineamiento de proteínas en la cual aparece el CLCS [15]. Las secuencias utilizadas para estos tipos de alineamientos están formadas del alfabeto de las proteínas, alfabeto de tamaño 20. Desde luego se debe tener en cuenta como las secuencias generadas de forma aleatoria tienen una frecuencia de aminoácidos diferente a las secuencias de la realidad, pero los resultados proporcionan un buen referente. En los resultados obtenidos es obvio observar la tendencia de aumento de tiempo para todos los algoritmos cuando crece la secuencia de restricción a excepción del algoritmo de Wang (Wang), este algoritmo tiene una tendencia constante y su mejor desempeño cuando la secuencia de restricción es de tamaño superior a 24. Con la tendencia constante del algoritmo de Wang (Wang), puede afirmarse que no existe influencia del crecimiento en el tamaño de la secuencia de restricción en solucionar el problema del CLCS, aunque este algoritmo no obtenga

el mejor desempeño en todos los casos. El algoritmo de Deorowicz (Deo) emplea menor tiempo en solucionar el problema cuando el tamaño de la secuencia de restricción es inferior a 24. El algoritmo propuesto (NCLCS) es deficiente para alfabetos de tamaño 20 y como se nota, el aumento de tamaño de las tres secuencias afecta de manera considerable el tiempo de respuesta.

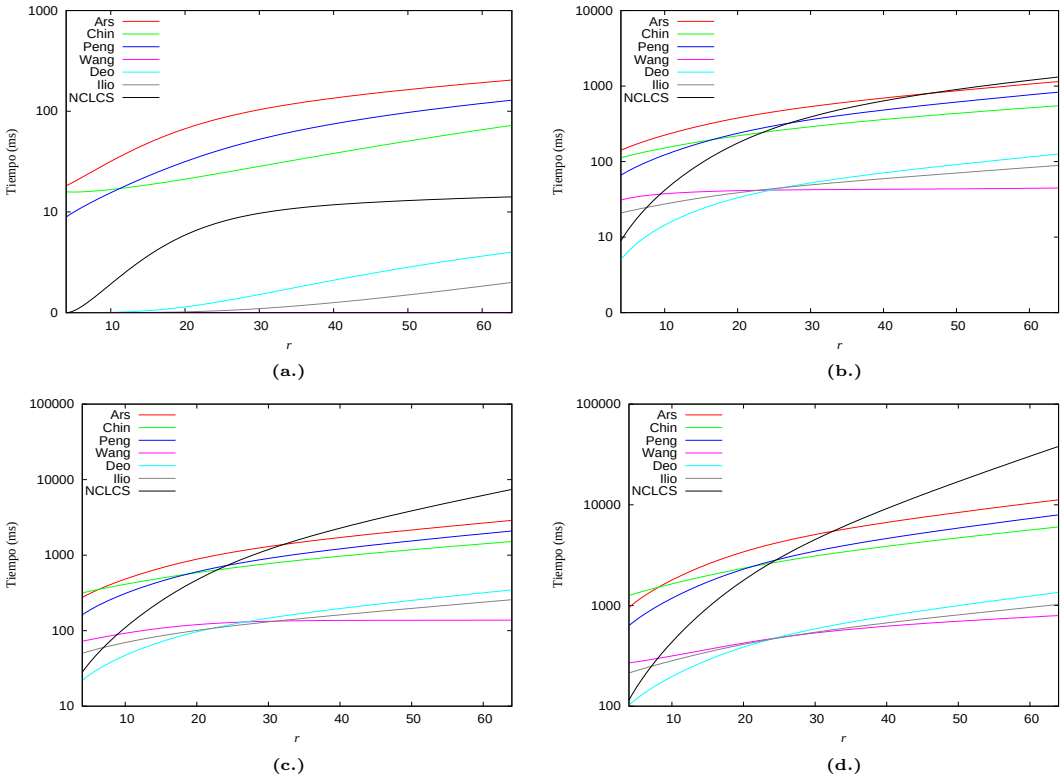


FIGURA 6.3: Comportamiento de los algoritmos variando el tamaño de la secuencia de restricción ($r$), con las variables fijas $\sigma = 20$ y **(a.)** $m = n = 100$, **(b.)** $m = n = 300$, **(c.)** $m = n = 500$ y **(d.)** $m = n = 1000$

El siguiente caso es observar el comportamiento de los algoritmos cuando las secuencias aumentan y el tamaño de la secuencia de restricción y el tamaño del alfabeto se mantienen (véase Figura 6.4). El tamaño del alfabeto se mantiene en 20 en justificación a la selección del alfabeto para la prueba anterior. Los resultados obtenidos presentan una tendencia similar para todos los algoritmos. El algoritmo propuesto (NCLCS) tiene un comportamiento similar a los algoritmos (Wang, Deo, Ilio) cuando la secuencia de restricción es de tamaño 8. Además, se puede afirmar de nuevo como el algoritmo de Deorowicz (Deo) es recomendado cuando la secuencia de restricción es inferior a un tamaño de 24 y como el algoritmo de Wang (Wang) se mantiene constante, su comportamiento no se ve perturbado por el aumento en el tamaño de la secuencia de restricción y como es favorable cuando el tamaño de la secuencia de restricción es superior a 24.

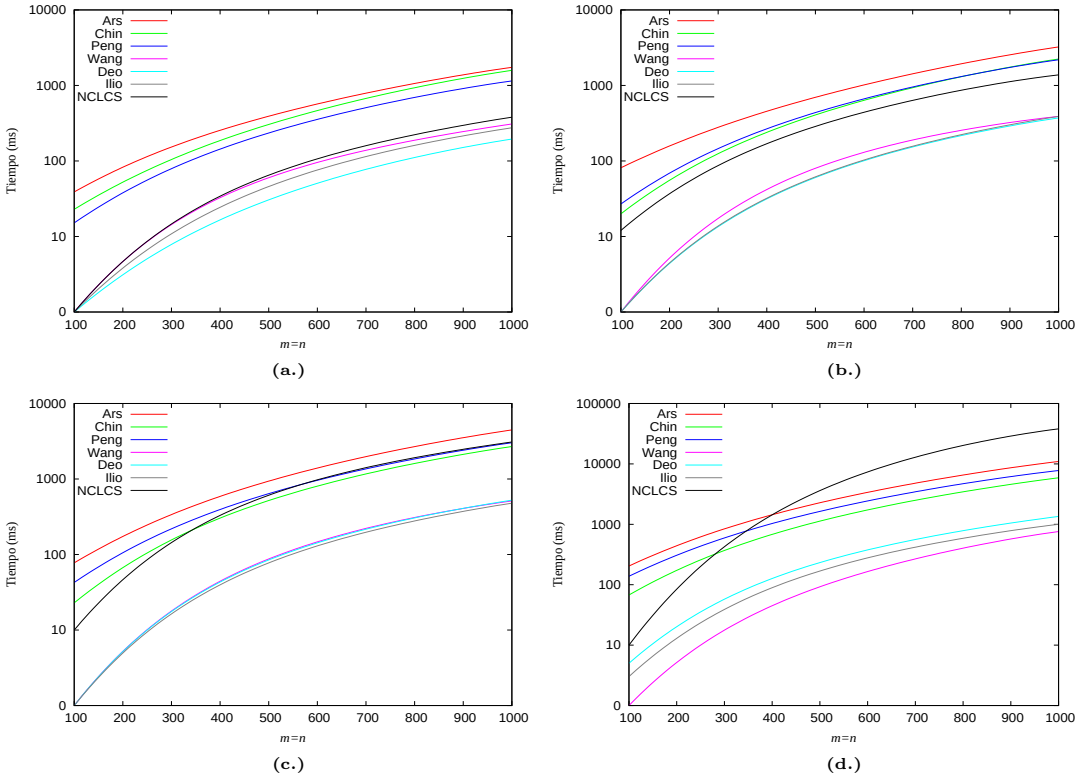


FIGURA 6.4: Comportamiento de los algoritmos variando el tamaño de las secuencias $x$ y $y$ ($m = n$), con las variables fijas $\sigma = 20$ y (a.) $r = 8$, (b.) $r = 16$, (c.) $r = 24$, (d.) $r = 64$

El último caso pretende mostrar el comportamiento de los algoritmos cuando una de las secuencias aumenta y se mantienen el tamaño de la otra secuencia, el tamaño de la secuencia de restricción y el tamaño del alfabeto. Se puede observar (véase Figura 6.5) un comportamiento muy similar de todos los algoritmos, pero se destaca la disminución del tiempo en la medida que el tamaño del alfabeto aumenta para el algoritmo propuesto (NCLCS) y para el algoritmo de Iliopoulos (Ilio). Sobre el algoritmo de Wang (Wang) se pueden confirmar dos cosas, primero, el algoritmo presenta un buen desempeño cuando el alfabeto tiene un tamaño igual o inferior a 20 y segundo, la tendencia del algoritmo no sufre un cambio cuando todas las secuencias aumentan de tamaño.

### 6.2.2 Resultados con secuencias de una colección de textos

Las pruebas realizadas con las secuencias extraídas de la colección de textos son del mismo tipo a las ejecutadas con las secuencias aleatorias. De esta manera, la Figura 6.6 representa el comportamiento de los algoritmos cuando varia el tamaño del alfabeto. Se observa como el tamaño del alfabeto no incide de manera determinante en el

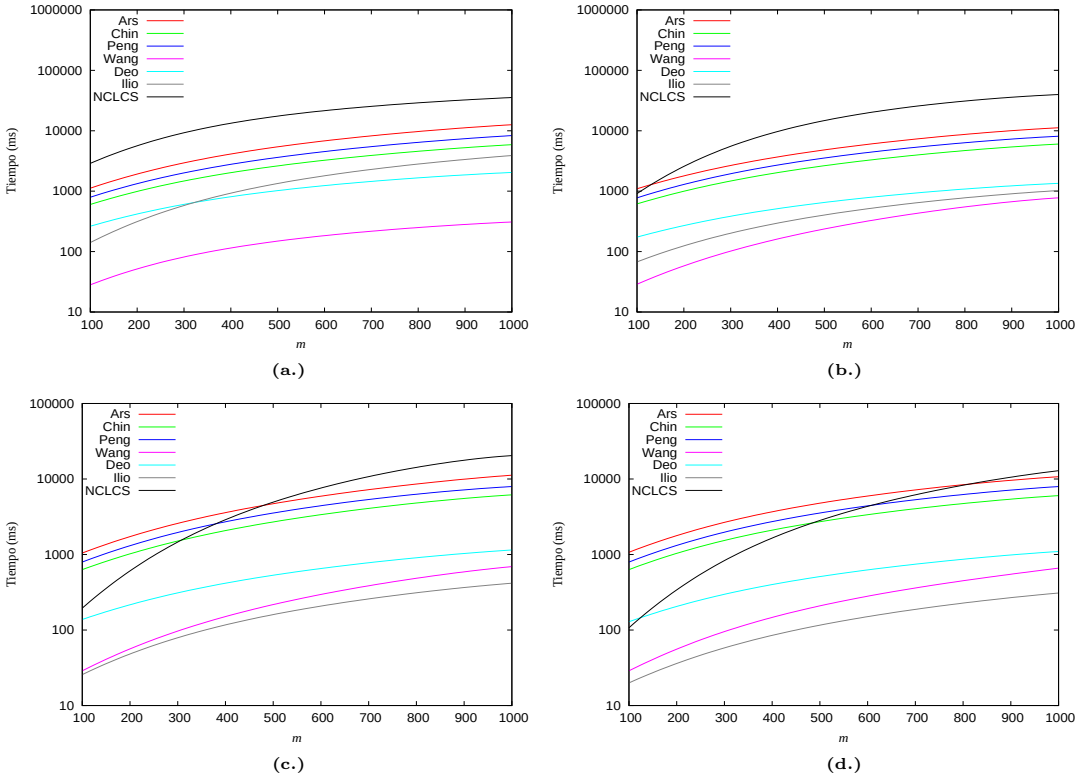


FIGURA 6.5: Comportamiento de los algoritmos variando el tamaño de la secuencia $y$ $(n)$, con las variables fijas $m = 1000$, $r = 64$ y **(a.)** $\sigma = 8$, **(b.)** $\sigma = 20$, **(c.)** $\sigma = 32$, **(d.)** $\sigma = 64$

comportamiento de los algoritmos (Ars, Chin, Peng). De igual forma como ocurre con los resultados de las secuencias generadas de forma aleatoria los algoritmos (Deo, Ilio, NCLCS) muestran la tendencia decreciente de los algoritmos basados en "coincidencias". Existen dos hechos importantes con respecto a la prueba sobre las secuencias generadas de forma aleatoria: el algoritmo de Deorowicz (Deo) mejora su desempeño para este tipo de secuencias incluso mejorando al algoritmo de Iliopoulos (Ilio) y el algoritmo de Wang (Wang) aumenta su desempeño óptimo con respecto a todos los demás algoritmos en alfabetos de tamaño inferior a 50.

El segundo caso es el comportamiento de los algoritmos variando la secuencia de restricción. El alfabeto seleccionado en este caso se basa en el hecho que cuando el tamaño del alfabeto aumenta, los algoritmos basados en "coincidencias" tienen un mejor desempeño en comparación con los algoritmos basados en "secuencias", es así como de la colección de textos el alfabeto de mayor tamaño (215) pertenece a "*ENGLISH*" (Tabla 6.2). Los resultados (véase Figura 6.7) muestran la tendencia similar de los algoritmos (Ars, Chin, Peng). El comportamiento del algoritmo de Wang (Wang) es constante. Los

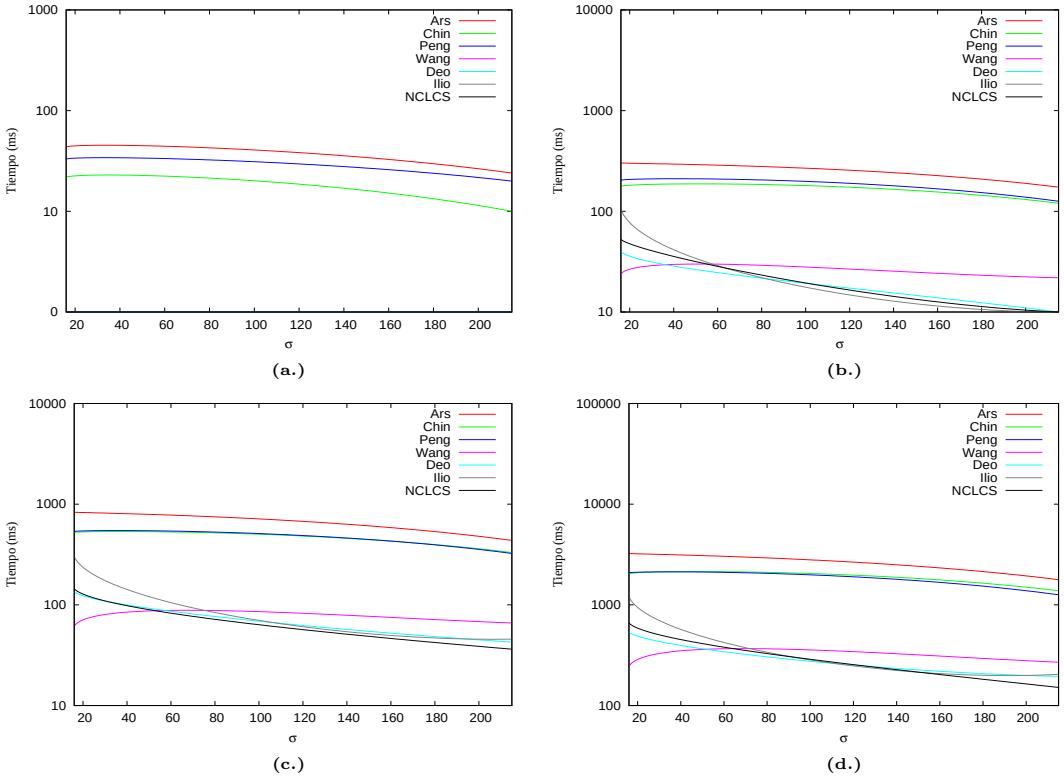


FIGURA 6.6: Comportamiento de los algoritmos variando el tamaño del alfabeto ($\sigma$), con las variables fijas $r = 16$ y (a.) $m = n = 100$, (b.) $m = n = 300$, (c.) $m = n = 500$ y (d.) $m = n = 1000$

mejores tiempos sobre las pruebas realizadas pueden notarse en tres secciones, si la secuencia de restricción es menor a 24 el algoritmo con mejor desempeño es el algoritmo propuesto (NCLCS), si la secuencia de restricción está entre un tamaño 24 y 32 el algoritmo de Iliopoulos (Ilio) es el indicado y si la secuencia de restricción supera el tamaño de 32, el algoritmo de Wang (Wang) tiene el mejor desempeño.

Si se observa la Figura 6.8 sobre el comportamiento de los algoritmos cuando los tamaños de las secuencias aumentan y se mantienen los tamaños del alfabeto y de la secuencia de restricción, tenemos una tendencia similar de todos los algoritmos. Pero, se confirma el mejor desempeño del algoritmo propuesto (NCLCS) con respecto a los demás algoritmos cuando la secuencia de restricción es pequeña, *e.g.* una secuencia de restricción de tamaño 8 (véase Figura 6.8 (a.)). Se recomienda el algoritmo de Wang (Wang) por su tendencia constante y cuando la secuencia de restricción es de tamaño grande, *e.g.* una secuencia de restricción de tamaño 64 (véase Figura 6.8 (d.)).

Al igual que la última prueba realizada sobre las secuencias generadas de forma aleatoria, en la cual aparte de mantener el tamaño de la secuencia de restricción y el tamaño del

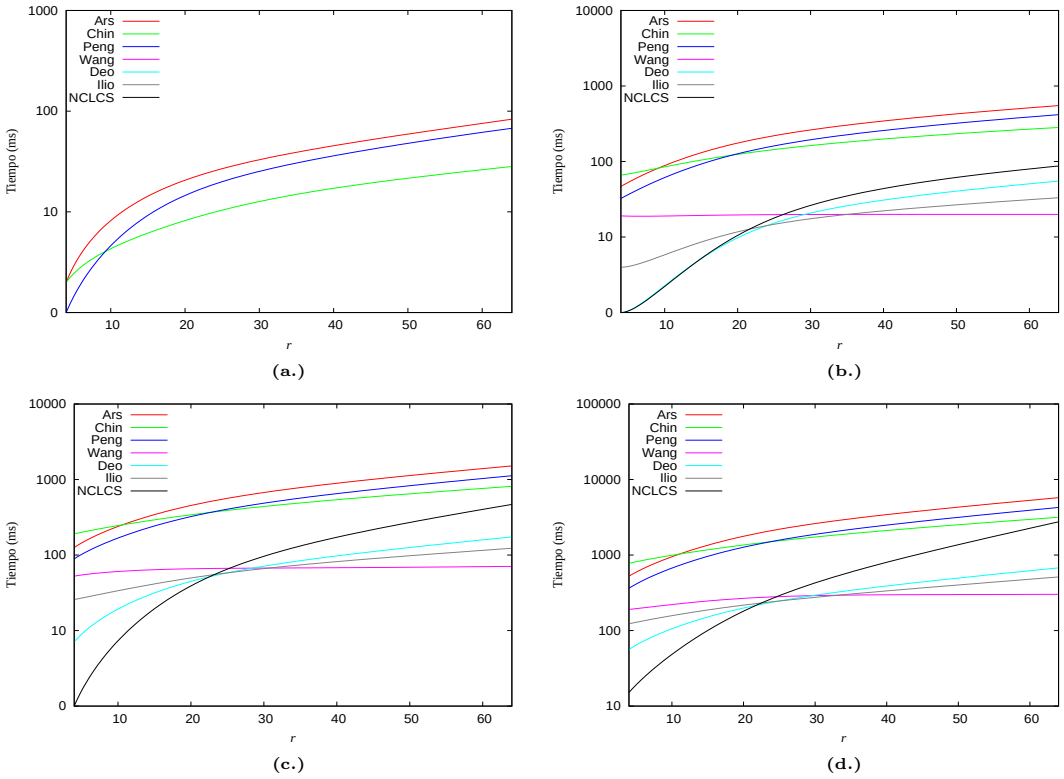


FIGURA 6.7: Comportamiento de los algoritmos variando el tamaño de la secuencia de restricción ($r$), con las variables fijas $\sigma = 215$ y **(a.)** $m = n = 100$, **(b.)** $m = n = 300$, **(c.)** $m = n = 500$ y **(d.)** $m = n = 1000$

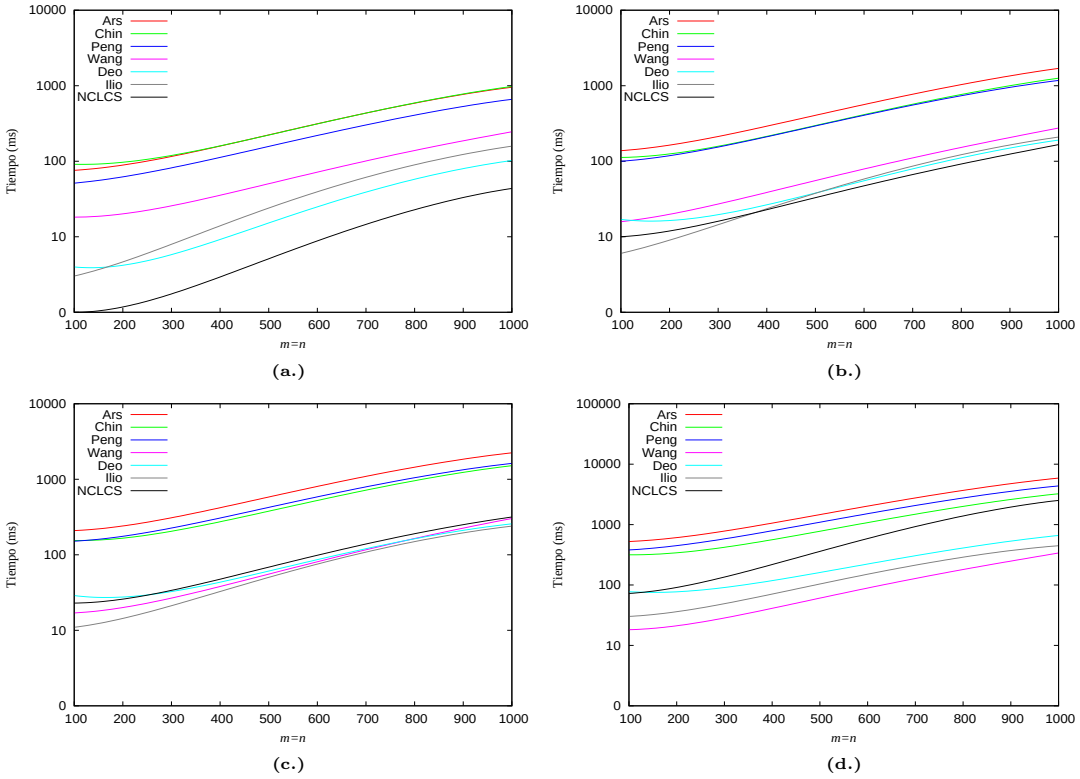


FIGURA 6.8: Comportamiento de los algoritmos variando el tamaño de las secuencias $x$ y $y$ ($m = n$), con las variables fijas $\sigma = 215$ y **(a.)** $r = 8$, **(b.)** $r = 16$, **(c.)** $r = 24$, **(d.)** $r = 64$

alfabeto se mantiene el tamaño de una de las secuencias mientras la otra secuencia varia su tamaño, los resultados obtenidos sobre las secuencias extraídas de la colección de textos se pueden observar en la Figura 6.9. Los resultados presentan una tendencia similar de todos los algoritmos, pero es evidente el comportamiento favorable del algoritmo de Wang (Wang) en los casos presentados. Además, los algoritmos de Deorowicz (Deo) e Iliopoulos (Ilio) tienden a presentar tiempos favorables cuando el tamaño del alfabeto aumenta y ocurre lo mismo de forma ostensible en el algoritmo propuesto (NCLCS).

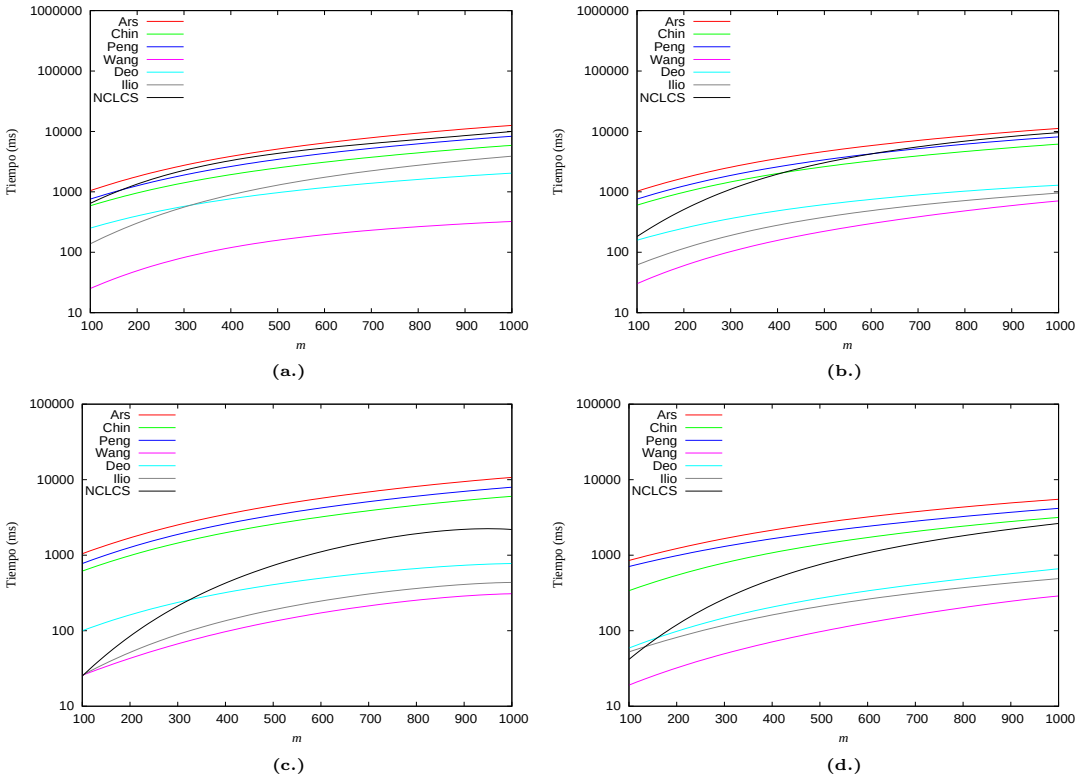


FIGURA 6.9: Comparación de los algoritmos seleccionados variando el tamaño de la secuencia $y$ $(n)$, con las variables fijas $m = 1000$, $r = 64$ y **(a.)** $\sigma = 16$, **(b.)** $\sigma = 25$, **(c.)** $\sigma = 133$, **(d.)** $\sigma = 215$

# 7

# Conclusiones y Trabajo Futuro

Esta tesis contribuyó con un estudio detallado de los algoritmos más importantes para solucionar el problema de la subsecuencia común más larga restringida en parte a un amplio conjunto de experimentos. Este problema básicamente consiste en determinar una subsecuencia común de máximo tamaño a dos secuencias dadas y que esta sea a su vez una supersecuencia de una tercera secuencia o secuencia de restricción.

Gracias a la implementación homogénea llevada a cabo, se logró plantear un conjunto de aseveraciones generales basadas en los comportamientos prácticos obtenidos de las distintas pruebas con los algoritmos. Este conjunto es: se confirmó desde un punto de vista experimental como el algoritmo de Tsai resulta impráctico. Los algoritmos de Arslan, Chin y Peng permanecieron ligados a un comportamiento dependiente al crecimiento de las secuencias y tampoco son funcionales en la práctica. El algoritmo de Wang es eficiente para secuencias pertenecientes a alfabetos pequeños (1 a 20 caracteres). Por lo tanto, este algoritmo es una buena elección para ser usado en aplicaciones con alfabetos de ADN. También, este algoritmo presenta la ventaja en alfabetos grandes (superiores a 20 caracteres) junto con secuencias de restricción mayores a 30 caracteres.

El algoritmo de Deorowicz tiene el mejor desempeño específicamente en las secuencias pertenecientes al alfabeto de las proteínas y secuencias de restricción inferiores a 20 caracteres.

Pese a tener una complejidad atractiva por estar basado en una estructura de datos eficiente, el algoritmo de Iliopoulos y Rahman no obtuvo el mejor desempeño en los casos propuestos.

Era de esperarse que los algoritmos basados en las coincidencias de las secuencias fuesen los más rápidos en todos los casos, pero, la etapa necesaria de preprocesamiento pesó en su comportamiento. En general, los resultados prácticos determinaron la importancia de la combinación del tamaño del alfabeto y del tamaño de las secuencias en el comportamiento de los algoritmos.

Como complemento, en esta tesis se propuso un algoritmo basado en el concepto de dominancia y comparable a los ya desarrollados, incluso este algoritmo mostró la mejor tendencia cuando el alfabeto contiene un número superior de 200 caracteres y la secuencia de restricción tiene un tamaño menor a 20 caracteres. Además, el algoritmo tiene la ventaja de recuperar todas las subsecuencias comunes más largas restringidas con respecto a los otros algoritmos.

Esta tesis también proporcionó una colección importante de algoritmos implementados que pueden formar una biblioteca. De la misma manera, este tipo de investigaciones pueden ayudar a acortar la brecha entre el aspecto teórico y práctico del comportamiento de los algoritmos.

Como trabajo futuro se deja abierta la posibilidad de aplicar nuevas técnicas de programación en la solución del problema de la subsecuencia común más larga restringida. Actualmente este trabajo está dirigido hacia el paradigma de la computación paralela como alternativa en la búsqueda de reducción de tiempos de ejecución en tipos de experimentos más robustos.

# A

# Anexos

Tabla A.1: Entropía promedio de las secuencias para la Figura 6.6

| Tipo | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_0$ | 1.748 | 1.779 | 1.233 | 1.867 | 1.825 | 1.132 | 1.877 | 1.566 | 1.228 | 1.899 | 1.906 | 1.178 |
| $H_1$ | 1.404 | 1.433 | 0.451 | 1.756 | 1.707 | 0.482 | 1.809 | 1.788 | 0.428 | 1.854 | 1.864 | 0.503 |
| $H_2$ | 0.819 | 0.825 | 0.133 | 1.380 | 1.344 | 0.141 | 1.566 | 1.526 | 0.116 | 1.723 | 1.743 | 0.081 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ |
| $H_0$ | 2.227 | 2.227 | 0.594 | 3.185 | 3.163 | 0.596 | 3.428 | 3.441 | 0.697 | 3.635 | 3.657 | 0.529 |
| $H_1$ | 0.382 | 0.361 | 0.075 | 0.811 | 0.816 | 0.049 | 1.171 | 1.147 | 0.058 | 1.572 | 1.566 | 0.058 |
| $H_2$ | 0.050 | 0.033 | 0 | 0.070 | 0.084 | 0 | 0.150 | 0.154 | 0.008 | 0.246 | 0.223 | 0 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ |
| $H_0$ | 1.833 | 1.697 | 0.587 | 2.184 | 2.397 | 0.592 | 2.577 | 2.580 | 0.770 | 2.947 | 2.967 | 0.686 |
| $H_1$ | 0.509 | 0.402 | 0.119 | 0.833 | 0.794 | 0.141 | 0.999 | 0.983 | 0.172 | 1.289 | 1.101 | 0.157 |
| $H_2$ | 0.165 | 0.108 | 0.053 | 0.328 | 0.240 | 0.041 | 0.385 | 0.340 | 0.044 | 0.476 | 0.333 | 0.047 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ |
| $H_0$ | 1.747 | 1.841 | 0.514 | 2.529 | 2.492 | 0.476 | 2.754 | 2.698 | 0.490 | 3.057 | 2.933 | 0.452 |
| $H_1$ | 0.356 | 0.384 | 0.041 | 0.787 | 0.777 | 0.016 | 1.041 | 1.003 | 0.041 | 1.422 | 1.360 | 0.016 |
| $H_2$ | 0.099 | 0.103 | 0 | 0.227 | 0.252 | 0 | 0.353 | 0.336 | 0.008 | 0.511 | 0.505 | 0.008 |

Tabla A.2: Entropía promedio de las secuencias para la Figura 6.7

| Tipo | $x_{100}$ | $y_{100}$ | $z_4$ | $x_{100}$ | $y_{100}$ | $z_8$ | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{100}$ | $y_{100}$ | $z_{24}$ | $x_{100}$ | $y_{100}$ | $z_{32}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_0$ | 1.886 | 1.845 | 0 | 1.791 | 1.813 | 0.256 | 1.776 | 1.815 | 0.508 | 1.748 | 1.817 | 0.769 | 1.872 | 1.812 | 0.907 |
| $H_1$ | 0.416 | 0.389 | 0 | 0.380 | 0.337 | 0 | 0.361 | 0.337 | 0.052 | 0.421 | 0.333 | 0.107 | 0.416 | 0.388 | 0.151 |
| $H_2$ | 0.107 | 0.082 | 0 | 0.075 | 0.074 | 0 | 0.091 | 0.093 | 0.028 | 0.111 | 0.084 | 0.016 | 0.098 | 0.094 | 0.079 |
| Tipo | $x_{300}$ | $y_{300}$ | $z_4$ | $x_{300}$ | $y_{300}$ | $z_8$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{24}$ | $x_{300}$ | $y_{300}$ | $z_{32}$ |
| $H_0$ | 2.489 | 2.522 | 0 | 2.538 | 2.525 | 0.222 | 2.537 | 2.554 | 0.535 | 2.493 | 2.322 | 0.843 | 2.524 | 2.540 | 0.890 |
| $H_1$ | 0.738 | 0.794 | 0 | 0.845 | 0.815 | 0 | 0.802 | 0.807 | 0.041 | 0.783 | 0.731 | 0.116 | 0.787 | 0.810 | 0.138 |
| $H_2$ | 0.212 | 0.229 | 0 | 0.244 | 0.251 | 0 | 0.242 | 0.230 | 0.008 | 0.246 | 0.243 | 0.005 | 0.222 | 0.229 | 0.058 |
| Tipo | $x_{500}$ | $y_{500}$ | $z_4$ | $x_{500}$ | $y_{500}$ | $z_8$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{24}$ | $x_{500}$ | $y_{500}$ | $z_{32}$ |
| $H_0$ | 2.780 | 2.807 | 0.033 | 2.780 | 2.789 | 0.166 | 2.769 | 2.812 | 0.537 | 2.811 | 2.707 | 0.773 | 2.804 | 2.734 | 1.010 |
| $H_1$ | 1.067 | 1.045 | 0 | 1.066 | 1.064 | 0.166 | 1.055 | 1.069 | 0.066 | 1.075 | 0.976 | 0.074 | 1.084 | 1.032 | 0.139 |
| $H_2$ | 0.353 | 0.341 | 0 | 0.353 | 0.356 | 0 | 0.344 | 0.345 | 0.008 | 0.332 | 0.317 | 0.016 | 0.334 | 0.340 | 0.030 |
| Tipo | $x_{1000}$ | $y_{1000}$ | $z_4$ | $x_{1000}$ | $y_{1000}$ | $z_8$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{24}$ | $x_{1000}$ | $y_{1000}$ | $z_{32}$ |
| $H_0$ | 3.003 | 3.038 | 0.033 | 3.050 | 2.988 | 0.200 | 3.026 | 2.990 | 0.560 | 2.826 | 3.067 | 0.850 | 3.006 | 3.006 | 1.084 |
| $H_1$ | 1.354 | 1.379 | 0 | 1.404 | 1.366 | 0 | 1.398 | 1.410 | 0.033 | 1.290 | 1.397 | 0.052 | 1.368 | 1.376 | 0.101 |
| $H_2$ | 0.494 | 0.495 | 0 | 0.515 | 0.517 | 0 | 0.521 | 0.527 | 0.016 | 0.516 | 0.472 | 0.005 | 0.505 | 0.510 | 0.004 |

Tabla A.3: Entropía promedio de las secuencias para la Figura 6.8

| Tipo | $x_{100}$ | $y_{100}$ | $z_8$ | $x_{300}$ | $y_{300}$ | $z_8$ | $x_{500}$ | $y_{500}$ | $z_8$ | $x_{800}$ | $y_{800}$ | $z_8$ | $x_{1000}$ | $y_{1000}$ | $z_8$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_0$ | 1.825 | 1.788 | 0.266 | 2.497 | 2.495 | 0.150 | 2.739 | 2.774 | 0.216 | 2.937 | 2.934 | 0.206 | 2.990 | 3.034 | 0.266 |
| $H_1$ | 0.331 | 0.376 | 0 | 0.808 | 0.828 | 0 | 1.029 | 1.057 | 0 | 1.228 | 1.300 | 0 | 1.425 | 1.388 | 0 |
| $H_2$ | 0.102 | 0.093 | 0 | 0.246 | 0.258 | 0 | 0.337 | 0.332 | 0 | 0.434 | 0.452 | 0 | 0.554 | 0.507 | 0 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{16}$ | $x_{300}$ | $y_{300}$ | $z_{16}$ | $x_{500}$ | $y_{500}$ | $z_{16}$ | $x_{800}$ | $y_{800}$ | $z_{16}$ | $x_{1000}$ | $y_{1000}$ | $z_{16}$ |
| $H_0$ | 1.840 | 1.849 | 0.536 | 2.515 | 2.556 | 0.509 | 2.805 | 2.770 | 0.538 | 2.943 | 2.980 | 0.506 | 2.986 | 3.031 | 0.488 |
| $H_1$ | 0.369 | 0.338 | 0.041 | 0.823 | 0.816 | 0.058 | 1.085 | 1.026 | 0.033 | 1.255 | 1.298 | 0.041 | 1.363 | 1.417 | 0.025 |
| $H_2$ | 0.082 | 0.072 | 0 | 0.248 | 0.243 | 0.016 | 0.336 | 0.318 | 0 | 0.433 | 0.442 | 0 | 0.522 | 0.516 | 0 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{24}$ | $x_{300}$ | $y_{300}$ | $z_{24}$ | $x_{500}$ | $y_{500}$ | $z_{24}$ | $x_{800}$ | $y_{800}$ | $z_{24}$ | $x_{1000}$ | $y_{1000}$ | $z_{24}$ |
| $H_0$ | 1.821 | 1.762 | 0.777 | 2.530 | 2.551 | 0.747 | 2.782 | 2.796 | 0.716 | 2.930 | 2.917 | 0.781 | 3.047 | 2.989 | 0.716 |
| $H_1$ | 0.366 | 0.383 | 0.126 | 0.839 | 0.777 | 0.070 | 1.071 | 1.070 | 0.099 | 1.216 | 1.221 | 0.104 | 1.412 | 1.389 | 0.095 |
| $H_2$ | 0.091 | 0.129 | 0.016 | 0.256 | 0.240 | 0.027 | 0.340 | 0.337 | 0.016 | 0.430 | 0.423 | 0.018 | 0.500 | 0.542 | 0.028 |
| Tipo | $x_{100}$ | $y_{100}$ | $z_{64}$ | $x_{300}$ | $y_{300}$ | $z_{64}$ | $x_{500}$ | $y_{500}$ | $z_{64}$ | $x_{800}$ | $y_{800}$ | $z_{64}$ | $x_{1000}$ | $y_{1000}$ | $z_{64}$ |
| $H_0$ | 1.868 | 1.833 | 1.474 | 2.373 | 2.478 | 1.517 | 2.745 | 2.776 | 1.581 | 2.896 | 2.942 | 1.527 | 3.042 | 3.050 | 1.430 |
| $H_1$ | 0.401 | 0.407 | 0.217 | 0.736 | 0.819 | 0.224 | 0.985 | 1.020 | 0.249 | 1.221 | 1.290 | 0.311 | 1.404 | 1.367 | 0.226 |
| $H_2$ | 0.116 | 0.088 | 0.069 | 0.215 | 0.246 | 0.050 | 0.314 | 0.330 | 0.034 | 0.425 | 0.444 | 0.079 | 0.502 | 0.507 | 0.056 |

Tabla A.4: Entropía promedio de las secuencias para la Figura 6.9

| Tipo | $x_{100}$ | $y_{1000}$ | $z_{64}$ | $x_{300}$ | $y_{1000}$ | $z_{64}$ | $x_{500}$ | $y_{1000}$ | $z_{64}$ | $x_{800}$ | $y_{1000}$ | $z_{64}$ | $x_{1000}$ | $y_{1000}$ | $z_{64}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_0$ | *1.771* | *1.901* | *1.705* | *1.867* | *1.886* | *1.706* | *1.868* | *1.893* | *1.694* | *1.893* | *1.890* | *1.724* | *1.897* | *1.912* | *1.675* |
| $H_1$ | *1.476* | *1.864* | *1.267* | *1.758* | *1.849* | *1.258* | *1.791* | *1.856* | *1.240* | *1.845* | *1.851* | *1.273* | *1.851* | *1.871* | *1.204* |
| $H_2$ | *0.857* | *1.740* | *0.606* | *1.394* | *1.720* | *0.580* | *1.559* | *1.730* | *0.606* | *1.683* | *1.731* | *0.607* | *1.717* | *1.746* | *0.617* |
| Tipo | $x_{100}$ | $y_{1000}$ | $z_{64}$ | $x_{300}$ | $y_{1000}$ | $z_{64}$ | $x_{500}$ | $y_{1000}$ | $z_{64}$ | $x_{800}$ | $y_{1000}$ | $z_{64}$ | $x_{1000}$ | $y_{1000}$ | $z_{64}$ |
| $H_0$ | *2.254* | *3.776* | *1.782* | *3.140* | *3.648* | *1.798* | *3.359* | *3.710* | *1.849* | *3.615* | *3.649* | *1.802* | *3.571* | *3.729* | *1.825* |
| $H_1$ | *0.380* | *1.671* | *0.240* | *0.816* | *1.541* | *0.280* | *1.167* | *1.635* | *0.274* | *1.525* | *1.494* | *0.257* | *1.429* | *1.582* | *0.220* |
| $H_2$ | *0.031* | *0.214* | *0.021* | *0.090* | *0.220* | *0.042* | *0.169* | *0.229* | *0.037* | *0.219* | *0.218* | *0.036* | *0.204* | *0.208* | *0.028* |
| Tipo | $x_{100}$ | $y_{1000}$ | $z_{64}$ | $x_{300}$ | $y_{1000}$ | $z_{64}$ | $x_{500}$ | $y_{1000}$ | $z_{64}$ | $x_{800}$ | $y_{1000}$ | $z_{64}$ | $x_{1000}$ | $y_{1000}$ | $z_{64}$ |
| $H_0$ | *1.668* | *3.253* | *1.466* | *2.446* | *2.919* | *1.518* | *2.755* | *2.516* | *1.341* | *3.159* | *2.949* | *1.434* | *1.862* | *2.969* | *1.568* |
| $H_1$ | *0.460* | *1.185* | *0.340* | *0.762* | *1.176* | *0.368* | *0.895* | *1.038* | *0.322* | *1.184* | *1.226* | *0.375* | *0.393* | *1.322* | *0.238* |
| $H_2$ | *0.151* | *0.321* | *0.091* | *0.255* | *0.421* | *0.114* | *0.263* | *0.422* | *0.088* | *0.356* | *0.439* | *0.145* | *0.100* | *0.477* | *0.061* |
| Tipo | $x_{100}$ | $y_{1000}$ | $z_{64}$ | $x_{300}$ | $y_{1000}$ | $z_{64}$ | $x_{500}$ | $y_{1000}$ | $z_{64}$ | $x_{800}$ | $y_{1000}$ | $z_{64}$ | $x_{1000}$ | $y_{1000}$ | $z_{64}$ |
| $H_0$ | *1.862* | *2.969* | *1.568* | *2.570* | *3.043* | *1.482* | *2.650* | *3.010* | *1.410* | *2.945* | *3.054* | *1.474* | *3.026* | *3.016* | *1.468* |
| $H_1$ | *0.393* | *1.322* | *0.238* | *0.832* | *1.416* | *0.232* | *1.046* | *1.413* | *0.242* | *1.320* | *1.417* | *0.224* | *1.407* | *1.385* | *0.260* |
| $H_2$ | *0.100* | *0.477* | *0.061* | *0.237* | *0.521* | *0.060* | *0.347* | *0.530* | *0.076* | *0.462* | *0.524* | *0.052* | *0.520* | *0.494* | *0.083* |

# Bibliografía


[1] Apostolico, Alberto: ≪String editing and longest common subsequences≫. *Handbook of formal languages, linear modeling: background and application*, (1997), **2**, pp. 361 – 398.

[2] Arslan, Abdullah N. y Eğecioğlu, Ömer: ≪Algorithms for the Constrained Longest Common Subsequence≫. *International Journal of Foundations of Computer Science*, (2005), **16**, pp. 1099 – 1109.

[3] Asbrink, Olof: *Algorithms for DNA alignment based on motifs.* Tesis de Maestría, Instituto Royal de Tecnología, Suecia, (2000).

[4] Baker, Brenda S. y Giancarlo, Raffaele: ≪Sparse Dynamic Programming for Longest Common Subsequence from Fragments≫. *Journal of Algorithms*, (2002), **42(2)**, pp. 231 – 254.

[5] Bergroth, L.; Hakonen, H. y Raita, T.: ≪A Survey of Longest Common Subsequence Algorithms≫. En: *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE 2000)*, pp. 39 – 48. IEEE Computer Society, Washington D.C., USA, (2000).

[6] Brodal, Gerth; Kaligosi, Kanela; Katriel, Irit y Kutz, Martin: ≪Faster Algorithms for Computing Longest Common Increasing Subsequences≫. En: *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006*, volumen 4009 de *Lecture Notes in Computer Science*, pp. 330 – 341. Springer, Barcelona, España, (2006).

[7] Chen, Jian Bein: *A Survey of the Longest Common Subsequence Problem and Its Related Problems.* Tesis de Maestría, Universidad Nacional Sun Yat – Sen, Taiwan, (2005).

[8] Chin, Francis; De Santis, Alfredo; Ferrara, Anna; Ho, N.L. y Kim, S.K.: ≪A simple algorithm for the constrained sequence problems≫. *Information Processing Letters*, (2004), **90(4)**, pp. 175 – 179.

[9] Cohen, Jaques: ≪Bioinformatics – An Introduction for Computing Scientists≫. *ACM Computing Surveys*, (2004), **36(2)**, pp. 122 – 158.

[10] Crochemore, Maxime; Iliopoulos, Costas; Pinzón, Yoan y Reid, James: ≪A Fast and Practical Bit-Vector Algorithm for the Longest Common Subsequence Problem≫. *Information Processing Letters*, (2001), **80(6)**, pp. 279 – 285.

[11] Crochemore, Maxime; Melichar, Borivoj y Tronícek, Zdenek: ≪Directed acyclic subsequence graph: overview≫. *Journal of Discrete Algorithms*, (2003), **1(3 – 4)**, pp. 255 – 280.

[12] Deorowicz, S. y Obstój, J.: ≪Constrained Longest Common Subsequence Computing Algorithms in Practice≫. *Informe técnico*, Silesian University of Technology, Polonia, (2009).

[13] Deorowicz, Sebastian: ≪Fast algorithm for constrained longest common subsequence≫. *Theoretical and Applied Informatics*, (2007), **19(2)**, pp. 91 – 102.

[14] Gotthilf, Zvi y Lewenstein, Moshe: ≪Approximating Constrained LCS≫. En: *String Processing and Information Retrieval (SPIRE 2007)*, pp. 164 – 172. Springer, Santiago, Chile, (2007).

[15] He, Dan y Arslan, Abdullah: ≪A Space – Efficient Algorithm for the Constrained Pairwise Sequence Alignment Problem≫. *Genome Informatics*, (2005), **16(2)**, pp. 237 – 246.

[16] Hirschberg, D. S.: ≪A linear space algorithm for computing maximal common subsequences≫. *Communications of ACM*, (1975), **18**, pp. 341 – 343.

[17] Hunt, J. W. y Szymanski, T. G.: ≪A fast algorithm for computing longest common subsequence≫. *Communications of ACM*, (1977), **20(5)**, pp. 350 – 353.

[18] Hyyrö, Heikki: ≪Bit – parallel LCS – length computation revisited≫. En: *Proceedings 15th Australasian Workshop on Combinatorial Algorithms (AWOCA)*, pp. 16 – 27, (2004).

[19] Iliopoulos, Costas S. y Rahman, M. Sohel: ≪Algorithms for computing variants of the longest common subsequence problem≫. *Theorical Computer Science*, (2008), **395(2 − 3)**, pp. 255 – 267.

[20] ——: ≪New Efficient Algorithms for LCS and Constrained LCS Problem≫. *Information Processing Letters*, (2008), **106(1)**, pp. 13 – 18.

[21] Jones, Neil C: *An introduction to bioinformatics algorithms.* MIT Press, Cambridge, MA, (2004). ISBN 0262101068.

[22] Kumar, Vootla Sudheer: *On The Longest Common Subsequence Problem – General and Variants.* Tesis de Maestría, Instituto de Tecnología de la India, (2006).

[23] Navarro, Gonzalo: ≪A Guided Tour To Approximate String Matching≫. *ACM Computing Surveys*, (2001), **33(1)**, pp. 31 – 88.

[24] Needleman, Saul B. y Wunsch, Cristian D.: ≪A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins≫. *Journal of Molecular Biology*, (1970), **48**, pp. 443 – 453.

[25] Nicolas, François: *Alignement, séquence consensus, recherche de similarités : complexité et approximabilit.* Tesis doctoral, Universidad de Montpellier II, Francia, (2005).

[26] Ning, Kang; Kee, Ng Hoong y Leong, Hon: ≪Finding Patterns in Biological Sequences by Longest Common Subsequences and Shortest Common Subsequences≫. En: *Proceedings of the Sixth IEEE Symposium on BionInformatics and BioEngineering (BIBE 2006)*, pp. 53 – 60. IEEE Computer Society, Washington D.C., USA, (2006).

[27] Peng, Chao Li: *An Approach for Solving the Constrained Longest Common Subsequence Problem.* Tesis de Maestría, Universidad Nacional Sun Yat – Sen, Taiwan, (2003).

[28] Pinzón, Yoan: *Approximate Strings Algorithms Using Bit – Vector Techniques.* Tesis doctoral, Universidad King's College, Reino Unido, (2001).

[29] Smith, T. F. y Waterman, M. S.: ≪Identification of Common Molecular Subsequences≫. *Journal of Molecular Biology*, (1981), **147**, pp. 195 – 197.

[30] Tsai, Yin Te: ≪The constrained longest common subsequence≫. *Information Processing Letters*, (2003), **88(4)**, pp. 173 – 176.

[31] Wang, Wei-Lun: *Longest Common Subsequence with Constraints.* Tesis de Maestría, Universidad Nacional Chi–Nan, China, (2006).