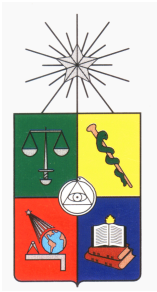

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERIA MATEMÁTICA

ANÁLISIS SUAVIZADO DEL ALGORITMO BOYER-MOORE-HORSPOOL

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

TOMÁS IGNACIO GONZÁLEZ SAAVEDRA

PROFESOR GUÍA:
SR. MARCOS ABRAHAM KIWI KRAUSKOPF

MIEMBROS DE LA COMISIÓN:
SR. JÉRÉMY FELIX BARBAY
SR. IVÁN RAPAPORT ZIMERMANN

SANTIAGO DE CHILE
JUNIO 2011

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO
POR: TOMÁS GONZÁLEZ SAAVEDRA
FECHA: 8/6/2011
PROF. GUÍA: MARCOS KIWI KRAUSKOPF

ANÁLISIS SUAVIZADO DEL ALGORITMO BOYER-MOORE-HORSPOOL

El objetivo del presente trabajo de título es estudiar el algoritmo para búsqueda de patrones en texto debido a Boyer-Moore-Horspool (BMH) bajo un enfoque de texto sometido a una ligera perturbación independiente carácter a carácter, de manera de determinar la esperanza del desempeño del algoritmo sometido a dicha perturbación.

El algoritmo BMH es una simplificación de Horspool al algoritmo que había sido propuesto anteriormente por Boyer y Moore (BM). En el mismo trabajo original de Horspool, él hace notar que su versión, en cuanto a tiempo de ejecución y comparaciones entre texto y patrón, aunque más sencilla y con peor desempeño de peor caso, tiene un desempeño comparable y a veces incluso superior al del algoritmo BM. Si bien diversos trabajos han hecho un buen análisis del algoritmo BMH especialmente en el caso promedio, no es claro que un análisis promedio sea una buena respuesta a la pregunta de por qué un algoritmo funciona bien en la práctica aunque su desempeño de peor caso sea deficiente.

Basados en el enfoque de análisis suavizado propuesto originalmente por Spielman y Teng pero adaptado para problemas de naturaleza discreta y más específicamente para texto, proponemos una perturbación bajo la cual analizar el desempeño esperado del algoritmo BMH cuando la entrada es sometida a una perturbación lo suficientemente pequeña como para no eliminar la influencia que la entrada ejerce sobre el desempeño del algoritmo. Para ello, en primer lugar, diseñamos herramientas que permiten encontrar una familia de constantes, indexada por parámetros del algoritmo, que acotan el peor desempeño en cada caso. También diseñamos un procedimiento que permite calcular dichas constantes para patrones y alfabetos pequeños. Luego encontramos una familia de constantes análoga a la anterior para acotar el desempeño esperado, bajo perturbación de texto, del algoritmo, y demostramos resultados que aseguran que dicho desempeño perturbado, bajo condiciones razonables, es mejor que el desempeño de peor caso asintóticamente con el tamaño del texto.

Finalmente, mostramos alguna evidencia empírica que sugiere que dentro del espacio de entradas la mayoría lleva a un comportamiento sustancialmente mejor al del peor caso. Las herramientas desarrolladas durante el trabajo y el enfoque aplicado podrían, en un trabajo futuro, justificar la diferencia entre el desempeño teórico y el observado en la práctica para BMH.

AGRADECIMIENTOS

Este trabajo fue realizado con el apoyo financiero de CONICYT via Proyecto Fondecyt 1090227 y el programa Basal-FONDAP Centro de Modelamiento Matemático.

Índice general

Índice de figuras

Índice de notaciones

Lista de símbolos utilizados frecuentemente con su respectiva definición

Σ	Alfabeto de tamaño σ.
T	Texto de tamaño n y caracteres $T_1 \dots T_n$.
P	Patrón de tamaño m y caracteres $P_1 \dots P_m$.
$delta_1(\alpha)$	Shift asociado al caracter $\alpha \in \Sigma$ según la primera regla del algoritmo BM.
$delta_2(\alpha)$	Shift asociado al caracter $\alpha \in \Sigma$ según la segunda regla del algoritmo BM.
K_α	Shift asociado al caracter $\alpha \in \Sigma$ según el algoritmo BMH.
K_m	K_{P_m}.
K_Σ	Variable aleatoria que representa el shift para un caracter α elegido uniformemente en Σ.
$\mathcal{X} = \mathcal{X}(A)$	Espacio de entradas para un algoritmo A.
p	Parámetro de perturbación sobre una entrada, $p \in [0, 1]$.
s	Varianza de una perturbación Gaussiana sobre una entrada, $s \geq 0$.
$\not\alpha$	Cualquier $\beta \in \Sigma \backslash \{\alpha\}$.
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Digrafo de conjunto de nodos $\mathcal{V}$ y arcos $\mathcal{E}$.
$e = (A, B)$	Para $A, B \in \mathcal{V}$, el arco en $\mathcal{E}$ que sale de A y llega hacia B.
$C(e) = C(A, B)$	Costo del arco $e = (A, B) \in \mathcal{E}$
$\ell(e) = \ell(A, B)$	Largo del arco $e = (A, B) \in \mathcal{E}$
τ	Función que asocia a cada arco en $\mathcal{E}$ una expresión regular sobre Σ.

Capítulo 1

Introducción

El problema de búsqueda de patrones en texto (llamado Pattern Matching o String Matching en inglés) consiste en buscar un bloque de texto dentro de un texto más grande, y tiene una serie de aplicaciones desde Ciencias de la Computación hasta Biología. Existen hoy varios algoritmos que resuelven el problema, desarrollados principalmente en la década de 1970, cada uno con sus resultados de desempeño en cuanto a complejidad, tanto de tiempo como de espacio.

Como es usual para el análisis de algoritmos, estos resultados de complejidad están en su mayoría estudiados para el peor caso. Sin embargo, el análisis de peor caso muchas veces no logra describir el comportamiento típico de un algoritmo y suele ser que ambos sean muy diferentes. Más aún, puede ocurrir que el "peor caso" para el desempeño de un algoritmo dado sea un ejemplo desarrollado en el ámbito académico, artificialmente diseñado para derrotar (o dificultar) a dicho algoritmo, y que no sea una buena representación de ninguna instancia típica del problema considerado. En particular, una serie de problemas tienen algoritmos que, a pesar de tener una alta complejidad de peor caso, presentan un muy buen desempeño en la práctica. El algoritmo Simplex, como se verá en secciones posteriores, es el ejemplo por antonomasia de una situación de este tipo. En este contexto surge la motivación para encontrar una medida de complejidad que sea capaz en algún sentido de explicar las diferencias entre el análisis teórico y el desempeño en la práctica.

En el extremo opuesto se puede encontrar el análisis del caso promedio. En éste, se suele considerar una entrada completamente aleatorizada (de acuerdo a una distribución adecuada), midiendo la complejidad del algoritmo de acuerdo al desempeño esperado del algoritmo, medido sobre la aleatoriedad de la entrada. Aunque el análisis del caso promedio puede dar alguna evidencia teórica de que un algoritmo puede funcionar mejor en la práctica, no es usual que explique el comportamiento general del algoritmo en un caso típico: las instancias prácticas de la mayoría de los problemas no son completamente aleatorias y, más aún, incluso si se consigue un buen

resultado para el desempeño promedio (una buena cota sobre la complejidad) bajo una distribución, esto puede no decir nada si se cambia por otra distribución, mucho menos si se considera una instancia típica del problema, ya que aparecen preguntas de difícil respuesta: ¿cuál es la distribución "más adecuada" a considerar para un problema dado? ¿cómo representa dicha distribución una entrada real para el problema?

Por otro lado, es usual que en el proceso de generar la entrada para un algoritmo se cometan errores, cuya procedencia depende de la naturaleza del problema considerado. Una medición puede presentar imprecisiones, una cantidad puede tener un error numérico asociado, se pueden cometer errores en la transcripción de una entrada, etcétera. Esto hace pensar que, "en el mundo real", muchos algoritmos estarán de verdad lidiando con una entrada que presenta una cierta cantidad de errores.

El análisis suavizado intenta ser una especie de medio camino entre el análisis de peor caso y el análisis promedio. En él, se considera una instancia *perturbada* del problema a resolver, donde la perturbación debe ser definida adecuadamente para cada problema, y se aplica a una instancia dada (usualmente una instancia donde el algoritmo a analizar tendría un desempeño pobre). Las peores instancias de un algoritmo pueden ser casos extraordinariamente aislados, y entonces, intuitivamente, una perturbación podría eventualmente convertir una instancia de peor caso en otra donde el algoritmo funcione razonablemente bien. Un algoritmo con buena complejidad suavizada debería tener un buen desempeño en casi todas las instancias "cercanas" a la instancia original. Por supuesto, conceptos como la cercanía de las instancias deben ser adecuadamente definidas para cada problema. Esto es uno de los objetivos de este trabajo, para el caso del problema de búsqueda de patrones.

Más específicamente, intentaremos hacer un análisis enfocado hacia el algoritmo Boyer-Moore-Horspool (BMH) para búsqueda de patrones. Este algoritmo es utilizado en varias aplicaciones prácticas para resolver el problema de búsqueda de patrones, y su simpleza en relación a otros algoritmos es una de las razones que explica nuestro interés inicial en él. A lo largo del trabajo se detallarán los aspectos principales de este algoritmo, particularmente qué lo hace un buen candidato frente al resto de los algoritmos que resuelven este problema, (para este tipo de análisis). El trabajo incluye una serie de resultados sobre el algoritmo, que se espera sean un buen aporte para entender su complejidad suavizada.

Capítulo 2

Formalización del problema

En esta sección primero haremos la revisión bibliográfica adecuada, tanto respecto de algoritmos de búsqueda de patrones como del campo (bastante más nuevo) del análisis suavizado. Nos centraremos en detalles para el algoritmo BMH, introduciendo la notación adecuada y algunos resultados importantes. Luego expondremos nuestro enfoque para el análisis del desempeño de BMH e introduciremos algunos conceptos que serán de gran utilidad en las secciones posteriores.

2.1. Revisión bibliográfica

2.1.1. Búsqueda de patrones

Una *palabra* es una secuencia de caracteres en un alfabeto finito Σ. Llamamos ε a la palabra vacía. Dadas tres palabras $x,\ y,\ z$ decimos que x es un *prefijo* de xy, un *sufijo* de yx y un *factor* de yxz. Si $y \neq \varepsilon$ diremos, cuando queramos remarcar el hecho de que $y \neq \varepsilon$, que x es un *prefijo propio* de xy y un *sufijo propio* de yx. Dadas palabras $T = T_1 \dots T_n$ y $P = P_1 \dots P_m$, que llamamos respectivamente el *texto* y el *patrón*, el problema de búsqueda de patrones consiste en encontrar todas las ocurrencias de P en T, es decir, todas las subpalabras S de T de tamaño m tal que $S_i = P_i$ para $i = 1, \dots, m$, o decidir si no hay ninguna. La mayoría de los algoritmos clásicos que resuelven este problema escanean el texto basados en uno de tres enfoques siguientes: buscando ya sea prefijos, sufijos o factores del patrón, hasta encontrar una ocurrencia completa (respectivamente el prefijo, sufijo o factor más grande posible) [1]. Existen algoritmos con un enfoque diferente pero no os tratamos acá. En todos los casos que consideramos, una ventana llamada *ventana de búsqueda*, del mismo tamaño que el patrón, se desliza por el texto para escanearlo, y se mueve de izquierda a derecha siguiendo reglas propias a cada algoritmo. El escaneo del texto se hace comparando carácter a carácter entre la ventana en el texto y el patrón, según las reglas de cada caso. Cuando un carácter del texto no concuerda con

el carácter del patrón al que se le compara decimos que encontramos un *mismatch*.

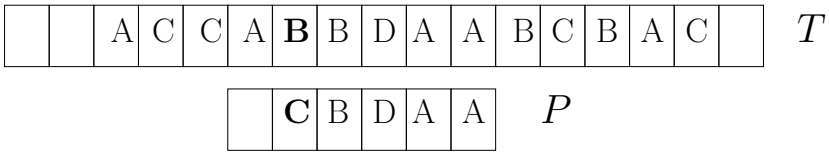

Figura 2.1: Mismatch patrón-texto, en negrita.

En lo que sigue mencionaremos las características principales de cada enfoque, basándonos en descripciones ya hechas principalmente por Navarro y Raffinot [1]. Este no es un resumen completo del estado del arte en cuanto a búsqueda de patrones, sino más bien una explicación general de los algoritmos que comparten ciertas características comunes que están en el enfoque de nuestro trabajo y constituyen un punto de partida para él:

- **Búsqueda por prefijos:** La búsqueda se hace hacia adelante en la ventana, leyendo todos los caracteres uno tras otro. Para cada posición de la ventana, buscamos el prefijo más largo de la ventana que es también prefijo del patrón. El ejemplo más conocido es el algoritmo Knuth-Morris-Pratt (KMP).
- **Búsqueda por sufijos:** La búsqueda se hace de derecha a izquierda en la ventana, leyendo el sufijo más largo de la ventana que también es sufijo del patrón. Con este enfoque es posible dejar de leer algunos caracteres en el texto, con lo que los algoritmos que lo ocupan pueden aspirar a ser sublineales, en el sentido de que no revisan el texto completo antes de terminar. Los ejemplos más conocidos son el algoritmo Boyer-Moore (BM) y, más relevante para este trabajo, la modificación hecha por Horspool a este último algoritmo conocida sencillamente como Boyer-Moore-Horspool (BMH).
- **Búsqueda por factores:** La búsqueda se hace hacia atrás en la ventana, buscando el sufijo más largo de la ventana que es también un factor del patrón. Este tipo de enfoques suele llevar a algoritmos muy eficientes, pero sufre del problema de que es complejo reconocer los factores del patrón, por lo que el desempeño de los algoritmos se ve perjudicado principalmente por el preprocesamiento. El ejemplo más conocido de este tipo de algoritmos es el llamado Backward Dawg Matching (BDM).

Comenzamos ahora a revisar la bibliografía pertinente para los algoritmos, en el orden en que fueron mencionados arriba:

La idea original para el algoritmo KMP fue propuesta por Morris y Pratt en 1970 [2]. Describiremos ahora someramente este algoritmo, e incluiremos más abajo un ejemplo que es recomendable leer junto con la descripción: Sea u un prefijo de P, y sea $b(u)$ el mayor prefijo de u que también es un sufijo de u (notamos que en ese caso $b(u)$ es también un prefijo de P). Tal prefijo $b(u)$ se llama el *borde* de u (Admitimos $b(u) = \varepsilon$). En la etapa de preprocesamiento se calcula $b(u)$ para cada prefijo u del patrón, que usamos para determinar el movimiento a la derecha de la ventana de búsqueda: si la ventana ha comparado patrón y texto hasta encontrar un prefijo u del patrón, seguido de un mismatch, entonces se mueve la ventana de búsqueda a la derecha $|u| - |b(u)|$ caracteres, siempre que la posición en el texto que causó el mismatch (digamos T_i) sea igual a $P_{|b(u)|+1}$. En caso contrario, se verifica que $T_i = P_{|b(b(u))|+1}$ y de ser así, se desplaza la ventana de búsqueda $|u| - |b(b(u))|$ espacios a la derecha, y así sucesivamente, hasta que una de las comparaciones entre T_i y los $P_{|b(...b(u))|+1}$ concuerda o hasta que uno de los bordes no tenga su propio borde (ε no tiene borde), momento en el cual podemos desplazar la ventana hasta sobrepasar T_i.

Consideremos el siguiente ejemplo para ilustrar el enfoque de Morris y Pratt: sea $P = baabacba$ y sea el texto $T = baabaaabbacbccabca$. En primer lugar, se compara patrón y texto desde la primera posición hacia la derecha hasta encontrar un mismatch en $T_6 = a \neq P_6 = c$. El prefijo u es en este caso $baaba$ y su borde es $b(u) = ba$. Como $T_6 = a = P_{|b(u)|+1}$ podemos desplazar la ventana $|u| - |b(u)|$ caracteres a la derecha, de tal forma que sabemos que $P_{1.,3} = T_{4.,6}$. La nueva comparación es entre $P_4 = b$ y $T_7 = a$, lo que produce un mismatch. Después de este mismatch el prefijo u del patrón es $u = baa$, con lo que $b(u) = \varepsilon$ y, entonces, como $T_7 \neq P_{|b(u)|+1} = b$, sabemos que se puede mover la ventana hacia la derecha hasta sobrepasar T_7 por completo, con lo que la nueva comparación es entre P_1 y T_8, que preoduce concordancia. El siguiente mismatch ocurre entre $P_2 = a$ y $T_9 = b$. Ahora $u = a$ y de nuevo $b(u) = \varepsilon$, y como $T_9 = P_1 = b$, movemos la ventana $|u| - 0 = 1$ carácter a la derecha. El nuevo mismatch ocurre entre $P_3 = a \neq T_{11} = c$. El prefijo es $u = ba$, que de nuevo tiene borde ε, y como $P_1 \neq T_{11}$, podríamos mover la ventana de búsqueda más a la derecha del carácter T_{11}. Por supuesto, ya no hay ninguna esperanza de encontrar el patrón en lo que queda de texto, por lo que podemos parar. La siguiente tabla ilustra el preprocesamiento del patrón.

Prefijo u	$b(u)$
b	ε
ba	ε
baa	ε
$baab$	b
$baaba$	ba
$baabac$	ε
$baabacb$	b
$baabacba$	ba

En el trabajo más difundido de 1977 Knuth [13] propuso un cambio (de nuevo incluimos un ejemplo más abajo): si después del prefijo u existe un mismatch entre texto y patrón, digamos de nuevo en la posición i del texto, T_i (que es lo que gatilla el desplazamiento de la ventana hacia la derecha), entonces para que pueda existir una ocurrencia del patrón tal que $T_i = P_{|b(u)|+1}$, necesariamente se debe tener $T_i \neq P_{|u|+1}$, esto es $P_{|b(u)|+1} \neq P_{|u|+1}$. Esa restricción se incluye en el preprocesamiento del algoritmo para cada prefijo propio de P, y esa es la versión actual más conocida del algoritmo KMP, aunque por supuesto existen múltiples casos específicos y variantes.

Utilizamos el mismo ejemplo de antes para ilustrar una ejecución de KMP con los comentarios que hemos mencionado. En primer lugar, la nueva tabla de bordes, agregando la restricción $P_{|b(u)|+1} \neq P_{|u|+1}$, se ve como sigue:

Prefijo u	$b(u)$
b	ε
ba	ε
baa	ε
$baab$	ε
$baaba$	ba
$baabac$	ε
$baabacb$	ε

Con esta tabla se puede ejecutar el algoritmo sobre el mismo texto del caso anterior. Al principio la ejecución es igual al caso MP, incluso el primer borde que se encuentra es $b(u) = ba$, ya que el primer borde es $baaba$, luego el primer desplazamiento es como antes. De hecho, la ejecución es igual hasta el final, ya que en ningún momento se encuentra un borde u que produzca bajo KMP un borde igual a ε que no habría producido el mismo borde bajo MP. Sin embargo, la diferencia en la tabla ilustra la diferencia entre ambos enfoques: mientras más bordes iguales a ε haya, más probable es luego de cada comparación que podamos mover la ventana de búsqueda lo más a la derecha posible, más allá del carácter que produjo el mismatch.

En su trabajo, Knuth, Morris y Pratt prueban que en el peor caso su algoritmo hace $O(n+m)$ comparaciones, y por otro lado el preprocesamiento adecuado puede hacerse en $O(m)$ operaciones. Además, mencionan que las llamadas cadenas de Fibonacci

$$\phi_1 = b, \qquad \phi_2 = a, \qquad \phi_n = \phi_{n-1}\phi_{n-2}, \quad n \geq 3,$$

son casos especialmente patológicos para el desempeño de este algoritmo.

Baeza-Yates [15] deriva varios resultados para el caso aleatorio en KMP. Usando cadenas de Markov, obtiene principalmente que el número esperado (sobre texto aleatorio de tamaño n donde cada carácter es elegido uniforme e independientemente sobre el alfabeto finito Σ) de comparaciones patrón-texto para KMP, denotado $\bar{C}_{KMP}(n)$, cumple

$$\frac{\bar{C}_{KMP}(n)}{n} \leq 2 - \frac{1}{|\Sigma|} + O\left(\frac{1}{n}\right),$$

cuando $|\Sigma| \leq \lceil \log_\phi m \rceil$, y, si llamamos ϕ a la razón de oro $(1+\sqrt{5})/2$

$$\frac{\bar{C}_{KMP}(n)}{n} \leq 1 + \frac{1}{|\Sigma|} + \frac{\lceil \log_\phi m \rceil - 1}{|\Sigma|^3} + O\left(\frac{\log^2(m)}{|\Sigma|^4}\right),$$

en caso contrario. Esto último, en particular, es una mejora sobre un resultado anterior debido a Régnier [16].

Al ver estos resultados se hace aparente una particularidad importante de estos algoritmos: existe una diferencia importante entre el desempeño en el caso promedio y el desempeño de peor caso. Veremos que esto no es un comportamiento único a KMP. Más aún, puede ser mucho más pronunciado en otros casos. Esta es una de las características que hace a este problema bastante atractivo de revisar con herramientas del análisis suavizado.

También en 1977 Boyer y Moore presentaron su propio algoritmo, basado en búsqueda por sufijos [3]. Otra vez, presentaremos un ejemplo más abajo para revisar de mejor manera el funcionamiento de este algoritmo. El enfoque de Boyer y Moore consiste en efectuar un preprocesamiento del patrón para generar dos heurísticas (en la práctica, dos tablas) que indican cuánto debe desplazarse la ventana de búsqueda cada vez que encontremos un mismatch patrón-texto. Inicialmente el patrón se alinea con el texto de manera de comparar el m-ésimo carácter de ambos y, luego, si ellos coinciden, se compara hacia atrás. En su trabajo, Boyer y Moore basan la presentación de su algoritmo en varias observaciones:

- Si el primer carácter del texto que se revisa en la actual ventana de búsqueda (esto es, el que se compara con P_m) al revisar el texto es uno que sabemos no aparece en P, eso descarta cualquier ocurrencia del patrón en la ventana de texto, y luego podemos desplazar la ventana m caracteres a la derecha, sin ningún riesgo de pasar por alto una ocurrencia de P en T.

- Más generalmente, supongamos que el carácter comparado con P_m ocurre en el patrón, y su última ocurrencia (de izquierda a derecha) en P es en la posición j. Esto quiere decir que se puede desplazar la ventana de búsqueda a la derecha $m - j$ posiciones sin riesgo de perder una ocurrencia de P.

- Suponiendo ahora que el carácter más a la derecha en la ventana de búsqueda coincide con P_m, queremos seguir comparando hacia atrás para encontrar una posible ocurrencia de P. O bien dicha ocurrencia existe, o encontramos un mismatch en la ventana de búsqueda, digamos en la posición j del patrón (esto es, comparando P_j con el carácter correspondiente de la ventana de búsqueda se encuentra un mismatch). Sea $\alpha \neq P_j$ el carácter que se lee en el texto. Si la última ocurrencia de α de izquierda a derecha en el patrón es en la posición $i < j$, entonces, aplicando el razonamiento de arriba, se puede desplazar la ventana de búsqueda $j - i$ posiciones a la derecha sin riesgo de pasar por alto una ocurrencia del patrón, llegando incluso a poder desplazar la ventana j posiciones a la derecha si es que α no aparece en el patrón. Si para $i > j$ se tiene $P_i = \alpha$ (esto es, hay una ocurrencia de α en P que está a la derecha del mismatch), este desplazamiento no tiene sentido, pero siempre se puede desplazar la ventana una posición a la derecha.

- Si el mismatch ocurre en la posición j del patrón (y una posición correspondiente en la ventana de búsqueda en el texto), entonces sabemos que los últimos $m - j$ caracteres del patrón y la ventana coinciden y, más aún, sabemos que en el texto T la secuencia $P_{j+1..m}$ que se lee en la ventana de búsqueda está precedida por un carácter que **no** es P_j, digamos $\alpha \neq P_j$. Esto quiere decir que, si sabemos que el patrón no tiene como factor al string $P_{j+1..m}$ precedido de un carácter que no sea P_j, entonces se puede desplazar la ventana de texto m posiciones a la derecha sin riesgo de ignorar una ocurrencia del patrón. Si, más generalmente, dicho factor ocurre en el patrón, de forma tal que para $i < j$ se tiene $P_{i+1..i+m-j} = P_{j+1..m}$ y $P_i \neq P_j$, entonces la ventana se puede desplazar $j - i$ posiciones a la derecha. Esta no es la formulación original del algoritmo, pero es la más utilizada.

Como se ve en la Figura 2.2, el desplazamiento, basado en la última observación, es de tres caracteres, ya que, como se muestra en itálica, existe una ocurrencia del substring AA tres posiciones a la izquierda del final en P, que está precedida por un carácter que no es el que provoca el mismatch (en negrita).

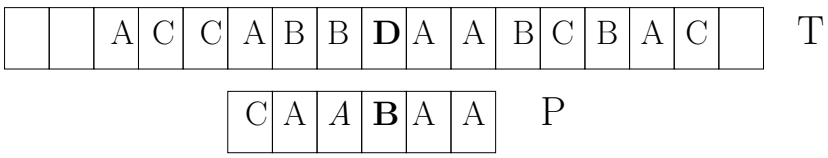

Figura 2.2: Desplazamiento basado en la última observación.

Teniendo en mente esto, el algoritmo BM consiste en preprocesar el patrón de manera de crear las tablas de shifts (llamadas $delta_1$ y $delta_2$), una inspirada en las primeras tres observaciones, mientras que la segunda sale de la última observación. La primera tabla tiene una entrada por cada carácter del alfabeto, y cada entrada se puede escribir, para $\alpha \in \Sigma$,

$$delta_1(\alpha) = \begin{cases} m & \text{si } \alpha \notin P, \\ m - j & \text{si } P_j = \alpha \text{ y } P_i \neq \alpha \text{ para } i > j. \end{cases}$$

Así, en el caso en que la ocurrencia más a la derecha en el patrón del carácter α que produce el mismatch en una ventana de búsqueda sea a la izquierda del mismatch (digamos $P_j = \alpha$ es la ocurrencia más a la derecha de α en P y el mismatch ocurre al comparar un carácter del texto con $P_i,\ i > j$), o bien α no ocurre en el patrón, y entonces basado en las observaciones de arriba, el algoritmo permite mover la ventana de búsqueda $m - i + delta_1(\alpha)$ posiciones a la derecha. Si, en caso contrario, la ocurrencia más a la derecha de α es a la derecha de la comparación que provoca el mismatch, el algoritmo permite moverse un espacio a la derecha (los autores llaman *worthless delta* a una situación como esa).

La segunda tabla tiene tantas entradas como el largo del patrón, de tal forma que, si se hacen i comparaciones antes de encontrar un mismatch, (esto es, el mismatch ocurre entre el carácter P_{m-i+1} y un α en el texto, con $\alpha \neq P_{m-i+1}$), el valor de $delta_2(i)$ especifica la cantidad de espacios que, basado en la última observación de arriba, se puede desplazar la ventana de búsqueda a la derecha. En cada caso, la cantidad de espacios que la ventana se desplaza a la derecha se toma como el máximo entre los valores indicados por las tablas $delta_1$ y $delta_2$.

Volviendo al ejemplo que utilizamos para los algoritmos anteriores, presentamos las tablas $delta_1$ y $delta_2$ para ese caso, de forma de desarrollar el ejemplo para BM. En la segunda columna guardamos los valores de $delta_1$, y en la cuarta los de $delta_2(i)$ para $i = 1, \ldots, m$, que nos indica cuánto se puede desplazar la ventana de búsqueda a la derecha si es que hemos hecho i comparaciones antes de encontrar un mismatch, basado en la observación que da origen a la tabla $delta_2$.

$\alpha \in \Sigma$	$delta_1(\alpha)$	$i \in \{1, \ldots, m\}$	$delta_2(i)$
a	-	1	1
b	1	2	5
c	2	3	3
		4	6
		5	6
		6	6
		7	6
		8	6

Los autores notan que el algoritmo podría correr en tiempo lineal en el peor caso incluso si se prescindiera de la tabla $delta_1$, y el resultado se tendría incluso para alfabetos infinitos, pero mencionan que hacer esto dañaría el desempeño promedio del algoritmo. Knuth [13] muestra que la versión propuesta por Boyer y Moore es lineal en el peor caso.

Ya en 1980 Horspool [4] propuso una simplificación del algoritmo anterior inspirado en dos observaciones relativamente sencillas:

- Si bien la eliminación de la tabla $delta_1$ en el algortimo anterior producía aún un algoritmo lineal en el peor caso, el comentario posterior de Boyer y Moore sobre cómo eso afectaría el desempeño promedio del algoritmo da la idea de que dicha tabla usualmente entrega los mejores desplazamientos a la derecha.
- En la construcción de la tabla $delta_1$ no es necesario usar como referencia el carácter que produjo un mismatch. Cualquier carácter que se haya leído en la comparación en la ventana de búsqueda actual puede servir.

La segunda observación merece algún comentario posterior: Si uno sabe que existe algún carácter $\alpha \in P$ que es muy poco común en el lenguaje en el que están escritos T y P, entonces uno podría buscar en T dicho carácter, sin molestarse en seguir comparando si no lo encuentra. Horspool llama SLFC (Scan for Lower Frequency Character) a dicho enfoque, pero no es su contribución principal.

La primera observación, explica Horspool, resulta del hecho de que la tabla $delta_2$ no suele contribuir demasiado a la velocidad del algoritmo, y sólo se usa en general para poder lidiar de mejor manera con patrones repetitivos (Eg: "XABCYXABC"). Pero, continúa, como este tipo de patrones no es muy común, no vale la pena gastar el tiempo de preprocesamiento para la tabla $delta_2$. El autor apoya su afirmación en evidencia empírica: variando el tamaño de los patrones (seleccionados aleatoriamente como un substring del texto T, con $|T| = 80000$), tomó el promedio entre varias ejecuciones de los algoritmos (principalmente para nuestro interés, el algoritmo BM y la simplificación propuesta) de la cantidad de caracteres escaneados por

segundo.

Así, la simplificación propuesta por el autor es como sigue: olvidar la tabla $delta_2$ por completo y, para decidir la cantidad de espacios que la ventana de búsqueda se mueve a la derecha cada vez, se utiliza una tabla (llamada $delta_{12}$ por Horspool) que tiene tantas entradas como caracteres en el alfabeto y que, análogo a la tabla $delta_1$ de BM, dice cuál es la ocurrencia más a la derecha del carácter en el patrón. Así, luego de encontrar un mismatch, la ventana se desplaza $delta_{12}(\alpha) = m - j$ espacios, donde j es la ocurrencia de más a la derecha del carácter α en el patrón, y el carácter α es **el primero que se lee al empezar a escanear la ventana de búsqueda**, es decir, el carácter que se compara con P_m.

Presentamos en lo que sigue la tabla $delta_{12}$ aplicada al ejemplo $P = baabacba$ para $\Sigma = \{a, b, c\}$. La tabla es sencillamente

$\alpha \in \Sigma$	$delta_{12}(\alpha)$
a	3
b	1
c	2

En este caso la ejecución del algoritmo es como sigue: en primer lugar se compara $P_8 = a$ con $T_8 = b$, obteniendo un mismatch. El patrón se desliza $delta_{12}(b) = 1$ carácter a la derecha para volver a comparar P_8 con $T_9 = b$, donde obtenemos lo mismo que en la primera comparación. Al comparar P_8 con T_{10} obtenemos concordancia, hasta que comparamos P_6 con T_8, donde hay un mismatch. Como $delta_{12}(a) = 3$, la nueva comparación es entre P_8 y T_{13}, lo que produce mismatch y un desplazamiento de dos caracteres. Luego comparamos P_8 con T_{15} donde obtenemos concordancia. Pero la comparación entre P_7 y T_{14} da un mismatch, y luego desplazamos la ventana de búsqueda 3 caracteres a la derecha para comparar P_8 con el último carácter de T. La concordancia es inmediatamente seguida por un mismatch entre P_7 y T_{17}, y el algoritmo se detiene pues no quedan posiciones más a la derecha. Incluimos también un pseudocódigo del algoritmo BMH ya que es nuestro principal objeto de análisis. El pseudocódigo es una adaptación casi directa, salvo por notación e idioma, del ofrecido por Reignier y Baeza-Yates [7].

```
Algoritmo BMH(T, P, n, m, Σ)
int delta12[|Σ|]
para (j = 0, j < |Σ|, j++) delta12[j] = m **Preprocesamiento parte 1
para (j = 1, j < m, j++) delta12[P[j]] = m − j **Preprocesamiento parte 2
para (i = m, i ≤ n, i+ = delta12[T[i]]){ **Búsqueda
int k = i
Para (j = m, j > 0 && T[k] = P[j], j−−) k−−
```

`Si` $(j == 0)$ `Ocurrencia-Patrón-en-posición-`$k + 1$
}

Esta simplificación terminó siendo conocida como el **Algoritmo BMH** o Boyer-Moore-Horspool. Como es el algoritmo más importante para este trabajo nos referiremos a él, a sus propiedades y a los resultados que han sido demostrados para él con cierta profundidad en una sección posterior (particularmente su desempeño en el peor caso y caso promedio, y explicaremos por qué es un buen candidato para análisis suavizado). Si bien existen otros algoritmos (en particular, algoritmos que utilizan el enfoque de búsqueda por factores), ellos no son de importancia para el resto de este trabajo y por lo tanto no los tratamos acá.

2.1.2. Análisis suavizado

Para un algoritmo A, sea $\mathcal{X}(A)$ el espacio de sus entradas (asumimos que $\mathcal{X}(A)$ tiene estructura de espacio vectorial respecto de alguna operación de suma y ponderación por escalares adecuada). Si no hay ambigüedad, notamos sencillamente $\mathcal{X}(A) = \mathcal{X}$, y asumimos que existe una medida del tamaño de las entradas (bajo una codificación razonable) que llamamos $\|\cdot\| : \mathcal{X} \to \mathbb{N}$. Sea $\mathcal{X}_n = \{x \in \mathcal{X} \mid \|x\| = n\}$ el espacio de entradas de tamaño $n \in \mathbb{N}$. Sea $\mathcal{C}_A(x)$ una medida de complejidad del algoritmo en la entrada $x \in \mathcal{X}$. El algoritmo A tiene complejidad de peor caso $f : \mathbb{N} \to \mathbb{N}$ si, para todo $n \in \mathbb{N}$ suficientemente grande, se tiene

$$\max_{x \in \mathcal{X}_n} \mathcal{C}_A(x) \quad = \quad f(n).$$

Anotamos $r \leftarrow \mathcal{X}_n$ para referirnos al proceso de elegir aleatoriamente una entrada en $\mathcal{X}_n$ de acuerdo a una distribución adecuada. Con esto, podemos decir que el algoritmo A tiene complejidad promedio $g : \mathbb{N} \to \mathbb{N}$ si, para todo $n \in \mathbb{N}$ suficientemente grande, se tiene

$$\mathbb{E}_{r \leftarrow \mathcal{X}_n}[\mathcal{C}_A(r)] \quad = \quad g(n).$$

Con esto en mente se puede dar una definición más formal de complejidad suavizada. Sea $x \in \mathcal{X}_n$. Elegimos r de acuerdo a una distribución adecuada en $\mathcal{X}_n$. El algoritmo A tiene complejidad suavizada $h_s : \mathbb{N} \to \mathbb{N}$ si, para todo $n \in \mathbb{N}$ suficientemente grande y un parámetro $0 \leq s < \infty$, se tiene

$$\max_{x \in \mathcal{X}_n} \mathbb{E}_{r \leftarrow \mathcal{X}_n}[\mathcal{C}_A(x + s\,\|x\|\,r)] \quad = \quad h_s(n).$$

El parámetro s permite ajustar la varianza de la perturbación, y se puede entender intuitivamente como la cantidad de ruido que se introduce a la entrada. Si s se hace

0, se vuelve al estudio del peor caso. Si s se hace demasiado grande, se esperaría recuperar los resultados del caso promedio, ya que el ruido aleatorio eclipsa por completo a la entrada original.

La primera noción de análisis suavizado fue introducida por Spielman y Teng [23]. El enfoque principal de dicho trabajo es el de explicar el buen comportamiento en la práctica del algoritmo Simplex para programación lineal. Es un resultado conocido que Simplex tiene complejidad exponencial en el peor caso [6]. Spielman y Teng consideran el problema

$$\begin{aligned} \text{máx} \quad & z^\top x \\ \text{s.a.} \quad & (\bar{A} + G)x \leq \bar{y} + h, \end{aligned}$$

donde $\bar{A}$ es una matriz arbitraria e $\bar{y}$ un vector arbitrario, y G y h son respectivamente una matriz y un vector de variables aleatorias Gaussianas de media 0 y desviación estándar $s \cdot \text{máx}_i \|(y_i, \bar{a}_i)\|$, donde $\bar{a}_i$ son los vectores fila de la matriz $\bar{A}$. Estas perturbaciones aleatorias dependen de los datos y a grandes rasgos pueden ser una buena representación de errores numéricos en los dígitos. En su trabajo, Spielman y Teng muestran que, para el algoritmo de Simplex de dos fases, usando una regla de pivoteo específica, el número esperado de pasos del algoritmo es a lo más $(nd/s)^{O(1)}$, donde d es la dimensión del vector x y n el número de desigualdades que definen el polítopo factible. Esto es, **Simplex presenta una complejidad suavizada polinomial**.

No sólo es este resultado el origen del campo del análisis suavizado, sino que también es uno de sus mayores triunfos. A pesar de que otros algoritmos para programación lineal ya tenían resultados comprobados de peor caso polinomial (lo que resolvía la pregunta respecto de la complejidad de este problema), ninguno de ellos tenía un desempeño práctico que siquiera pudiera rivalizar con Simplex, que siguió siendo el algoritmo preferido en su campo. Durante décadas se buscaron explicaciones teóricas para el buen comportamiento en la práctica de Simplex, sin encontrarse ninguna satisfactoria. Finalmente, con el trabajo de Spielman y Teng, se encontró una explicación razonable: en una codificación razonable del espacio de entradas de Simplex, **las instancias de peor caso para el algoritmo son tan aisladas dentro de su espacio de entradas que una pequeña perturbación puede convertir cualquiera de ellas en una instancia mucho más manejable**. Dicha perturbación tiene toda clase de explicaciones satisfactorias en la formulación del problema: Errores en la medición de datos, errores de redondeo en los números, etc. Sin duda fue este resultado el que al mismo tiempo dio el impulso inicial al análisis suavizado y un nuevo enfoque para el más importante algoritmo de programación lineal.

Este tipo de trabajos inspiró una serie de resultados similares de gran utilidad en análisis numérico y programación lineal [19, 20, 21], donde se ocupa un marco teórico similar al de Spielman y Teng, en particular en lo que respecta al tipo de perturbaciones. Sin embargo, al menos dos observaciones cruciales salen a la vista en este punto:

- En el mismo trabajo [23], los autores proponen que la perturbación aleatoria puede de algún modo destruir propiedades fundamentales de la entrada. En particular, una matriz perderá todas sus entradas nulas luego de ser sometida a una perturbación, reemplazándolas más bien con entradas de pequeño valor. Surge entonces la idea de *perturbaciones preservadoras de propiedades*. Es decir, a pesar de introducir aleatoriedad en la entrada, lo hagan bajo la condición de no perder alguna propiedad fundamental de ésta. En el caso particular de más arriba, se puede considerar una perturbación que sólo afecte a las entradas distintas de cero en la matriz.
- Este modelo de perturbaciones Gaussianas, adecuado para el marco de análisis numérico y otros problemas contínuos en general, no es adecuado para casi ningún problema de naturaleza discreta.

En un trabajo posterior, Spielman y Teng exploran asuntos concernientes a estos dos puntos y hacen un estudio de análisis suavizado para problemas discretos, en particular, para el estudio de ciertos problemas en grafos [18]. Ellos trabajan con nociones tales como las siguientes:

- Sea $\bar{G}$ un grafo y $p \in [0,1]$. La p-perturbación de $\bar{G}$ es el grafo obtenido a partir de $\bar{G}$ convirtiendo cada arco en no-arco con probabilidad p, y cada no-arco en arco con la misma probabilidad (XOR de la matriz de adyacencia). Esta distribución se denota $\mathcal{P}(\bar{G}, p)$.

- Sea f una función definida en el espacio de los grafos. Sea $\bar{G}$ un grafo y $p \in [0,1]$. Una p-perturbación f-preservante de $\bar{G}$ es el grafo aleatorio G con densidad

$$\frac{\mathbb{P}_{G \leftarrow \mathcal{P}(\bar{G},p)}[G \ \wedge \ f(G) = f(\bar{G})]}{\mathbb{P}_{G \leftarrow \mathcal{P}(\bar{G},p)}[f(G) = f(\bar{G})]}.$$

  En particular, si f toma valores en $\{0,1\}$ (podemos asociarla con una propiedad que puede tener o no tener el grafo), esto se refiere a perturbaciones que preservan propiedades.

- Sea A un algoritmo, f una función definida sobre los grafos y $\delta \in [0,1]$. Llamemos $A(G)$ al resultado del algoritmo A en G, donde $A(G) = 0$ significa un resultado incorrecto y $A(G) = 1$ uno correcto. Decimos que A tiene una probabilidad suavizada de error δ bajo p-perturbaciones f-preservantes si

$$\max_{\bar{G}} \mathbb{P}_{G \leftarrow \mathcal{P}(\bar{G},p)}[A(G) = 0 | f(G) = f(\bar{G})] \leq \delta.$$

Con este marco, Spielman y Teng logran ciertos resultados en cuanto a propiedades de grafos como por ejemplo decidir si tienen cliques de cierto tamaño, bisecciones o si son bipartitos [18]. Este trabajo es interesante, en cuanto está en la línea de problemas discretos.

Manthey y Reischuk [22] usan un modelo de análisis suavizado para el estudio de la altura de árboles de búsqueda binaria, obteniendo cotas superiores e inferiores para la altura esperada de un árbol de búsqueda binaria en una secuencia perturbada.

Finalmente, vale la pena mencionar el trabajo de Andoni y Krauthgamer [24], donde obtienen un resultado de complejidad suavizada para un problema íntimamente ligado al de búsqueda de patrones, que es el de cálculo de distancia de edición. En su trabajo, Andoni y Krauthgamer consideran dos palabras $x^*, y^* \in \{0,1\}^d$ y $0 \leq p \leq 1$. Una función $A : [d] \to [d] \cup \perp$ es un *Alineamiento* entre x^* e y^* si es monónotona creciente en $A^{-1}([d])$ y cumple $x^*(i) = y^*(A(i))$ para $i \in A^{-1}([d])$. El *largo* de A es $len(A) = |A^{-1}([d])|$ (el número de posiciones bien alineadas). El modelo de perturbación es el siguiente:

- Fijar un alineamiento de largo máximo A^* entre x^* e y^*.
- Sea $\pi_x \in \{0,1\}^d$ tal que $\pi_x(j) = 1$ con probabilidad $p/2$ y 0 con probabilidad $1 - p/2$, todas las posiciones independientes entre sí.
- Sea $\pi_y \in \{0,1\}^d$ definido análogamente (e independientemente) salvo que las posiciones alineadas entre x^* e y^* se mantienen correlacionadas, esto es, para $i \in (A^*)^{-1}([d])$, se cumple $\pi_y(i) = \pi_x((A^*)^{-1}(i))$.
- Poner $x(i) = x^*(i) + \pi_x(i)$ e $y(i) = y^*(i) + \pi_y(i)$ donde la suma es módulo 2. Esta es la instancia perturbada del problema.

En este caso, la propiedad que se quiere preservar después de la perturbación es el alineamiento: gracias a la correlación de los elementos de π_y y π_x, es seguro que el alineamiento en la instancia original lo seguirá siendo en la instancia perturbada. Esto resulta particularmente útil ya que el algoritmo propuesto por estos autores depende en gran medida del mayor alineamiento entre las palabras a las que se quiere calcular la distancia de edición. Basados en esta perturbación, Andoni y

Krauthgamer producen un algoritmo que dados $\epsilon, p > 0$, $x, y \in \{0,1\}^d$, aproxima la distancia de edición entre x e y hasta un factor $O((1/\epsilon p)\log(1/\epsilon p))$ y, si x e y provienen de una instancia perturbada como se definió arriba, con alta probabilidad (sobre la aleatoriedad de la perturbación) tiene tiempo de ejecución $O(d^{1+\epsilon})$. Vale la pena notar que este enfoque no es el original de trabajos como [23] (y como el que buscamos en este trabajo), ya que en vez de analizar un algoritmo y demostrar que tiene baja complejidad suavizada, los autores proponen su propio algoritmo que funciona bajo un marco similar al del análisis suavizado y demuestran luego que ese algoritmo corre, con alta probabilidad, en un tiempo bajo.

Este trabajo tiene también un interés especial para nosotros pues trabaja explícitamente con perturbaciones definidas para textos, por lo que el modelo de perturbación utilizado sirvió como referencia en nuestro trabajo. Si esperamos analizar el algoritmo BMH en un contexto suavizado, es crucial la definición de la perturbación. De esta forma, e inspirados también en los trabajos de complejidad promedio para BMH y otros algoritmos [7, 8, 16, 17], diseñamos una perturbación para textos de la siguiente forma: cada caracter en el texto es perturbado con probabilidad $p \in [0,1]$ independientemente y, si es perturbado, es reemplazado por un caracter elegido uniformemente en el alfabeto Σ. Así, cuando a lo largo del trabajo hablemos de la distancia entre textos en el espacio de entrada para BMH, nos referiremos a la distancia de Hamming, y no a alguna alternativa como la distancia de edición o distancia en permutaciones, que no son adecuadas en el contexto de nuestra perturbación.

2.2. El algoritmo

En la sección anterior ya describimos el algoritmo BMH como variante de BM. En esta sección vamos a profundizar en algunos aspectos y dar algunas definiciones que se usarán en el resto de este trabajo.

Conocemos la entrada del algoritmo: P, T y Σ. De ahora en adelante denotaremos por σ el tamaño de Σ, ie $|\Sigma| = \sigma$. Sabemos que BMH comienza alineando el patrón con el texto, posicionando una ventana de búsqueda de tamaño m al inicio de T, de forma que los primeros m caracteres de T queden alineados con el patrón, de manera de buscar P en la ventana comparando primero P_m con T_m. Si la ventana de búsqueda en el texto no coincide con el patrón, la desplazamos hacia la derecha según lo que indique la tabla que Horspool llamó $delta_{12}$. Más específicamente, la ventana se mueve $delta_{12}(T_m)$ a la derecha, donde, para $\alpha \in \Sigma$

$$delta_{12}(\alpha) \;=\; \text{mín}\{j \mid j = m \text{ o } (1 \leq j \leq m-1 \text{ y } P_{m-j} = \alpha)\},$$

esto es, la tabla indica, para cada carácter en el alfabeto, el lugar de su ocurrencia más a la derecha dentro del patrón, si es que tal ocurrencia existe, o es igual a m

en caso contrario. De ahora en adelante, para un patrón P fijo, llamamos el *shift* asociado a cada carácter α del alfabeto a la cantidad $delta_{12}(\alpha)$, y la notamos K_α. Siguiendo con el algoritmo, luego del primer mismatch, la ventana se habrá movido de tal forma que su extremo derecho ya no esté en la posición m del texto sino que en $m + K_{T_m}$, y es el carácter en esta posición el que se comparará con P_m. El carácter que encontremos en esa posición determinará el shift subsecuente, y así hasta el final de la ejecución del algoritmo. Es clara, de una observación como ésta, la importancia que adquiere el carácter que se compara con P_m cada vez que se mueve la ventana de búsqueda, pues será el que determina el shift en cada caso. En una ejecución de BMH llamamos *cabezas* a las posiciones en T que contienen caracteres que serán comparados con P_m, donde notamos $\mathcal{H}$ para el conjunto de cabezas ($T_\mathcal{H}$ son los caracteres en el texto que ocupan dichas posiciones), y denotamos por H su tamaño, ie $|\mathcal{H}| = H$.

Supongamos por otro lado que el carácter que está en una cabeza no es fijo, sino que es elegido uniformemente dentro del texto. Está claro que en una situación como esa el shift resultante es una variable aleatoria, y la llamamos K_Σ, donde además es fácil obtener, para $\Sigma = \{\alpha_1, \ldots, \alpha_\sigma\}$,

$$\mathbb{E}[K_\Sigma] \;=\; \frac{1}{\sigma}\sum_{j=1}^{\sigma} K_{\alpha_j}.$$

Obviamente K_Σ depende del patrón: $K_\Sigma = K_\Sigma(P)$, pero, como haremos a lo largo del trabajo cada vez que no haya confusión, no explicitaremos esta dependencia al referirnos a la variable.

Llamamos *comparación local* al acto de comparar de derecha a izquierda en la ventana de búsqueda hasta encontrar un mismatch patrón-texto. Si llamamos C_i a la cantidad de comparaciones patrón-texto que se hace en una comparación local con la posición i como cabeza, entonces $C = \sum_{i\in\mathcal{H}} C_i$ es la cantidad total de comparaciones que se hace en el texto T.

Finalmente, para una palabra W y dos enteros $i \leq j$, llamaremos $W_{i..j}$ a la secuencia $W_i \ldots W_j$. Dado un carácter $\alpha \in \Sigma$, llamaremos $\not\alpha$ a cualquier carácter en $\Sigma\backslash\{\alpha\}$.

2.3. Resultados conocidos

Presentaremos algunos resultados conocidos sobre el algoritmo BMH, especialmente en lo que se refiere al número de comparaciones. En primer lugar, es fácil notar que, para $P = ba^{m-1}$ y $T = a^n$, el algoritmo hace m comparaciones locales en

cada alineación de la ventana de búsqueda, antes de desplazarla una posición a la derecha. Con esto se tiene que $C = (n-m+1)m$, que es claramente el peor resultado posible. Por otro lado, un patrón como $P = a^m$ y $T = b^n$ da una sola comparación local antes de hacer un shift de m posiciones, con lo que $C = \lfloor n/m \rfloor$, lo que es el mejor caso. Esta es una de las características más interesantes del algoritmo BMH: la enorme diferencia existente entre el desempeño en el mejor y peor caso.

Sin embargo, y ya en la presentación original del algoritmo [4] Horspool hacía notar, presentando observaciones experimentales generadas aleatoriamente usando textos grandes, que la velocidad de la versión simplificada del algoritmo, comparada con la original, daba resultados muy parecidos en cuanto a la velocidad (medida en cantidad de caracteres que se avanza en el texto cada segundo), llegando incluso a superarla en varios casos. Esto es, los resultados empíricos sugieren que BMH, a pesar de ser más simple, es tan bueno como el algoritmo original.

Régnier y Baeza-Yates hacen un análisis del caso promedio del algoritmo reduciéndolo en primer lugar a un proceso estacionario [7]. El modelo probabilístico usado en su trabajo implica que los caracteres aleatorios se eligen uniformemente y de manera independiente unos de otros en el alfabeto. Este modelo de fuente sin memoria es predominante en casi todos los estudios en cuanto a comportamiento del caso promedio para algoritmos de pattern matching. Los autores introdujeron originalmente el concepto de cabeza que usamos más arriba. En primer lugar, ellos derivan un resultado asintótico para el número de cabezas: para un patrón fijo y texto aleatorio, la probabilidad de que cada carácter sea una cabeza converge a $1/\mathbb{E}[K_\Sigma]$. Luego obtienen para un patrón dado P, con $m \geq 4$, el número esperado de comparaciones patrón-texto en un texto aleatorio de tamaño n, denotado $\overline{C}_{BMH}(n, P)$:

$$\frac{\overline{C}_{BMH}(n,P)}{n} = \frac{1}{\mathbb{E}[K_\Sigma]}\left(\frac{\sigma}{\sigma-1} + \mathbb{E}\left[\frac{1}{\sigma^{K_\Sigma}}\right] + O\left(\frac{1}{\sigma^3}\right)\right).$$

Más aún, obtienen que, para un patrón aleatorio (elegido de la misma forma que T), se cumple, si ahora llamamos $\overline{C}_{BMH}(n)$ a la esperanza de comparaciones para este nuevo caso,

$$\frac{\overline{C}_{BMH}(n)}{n} = Ph(\sigma)\left(1 + \frac{1}{\sigma} + \frac{1}{\sigma^2} + O\left(\frac{1}{\sigma^3}\right)\right),$$

para patrones grandes, donde

$$\frac{1}{\sigma} \leq Ph(\sigma) \leq \frac{2}{\sigma+1}.$$

Observemos que todos estos resultados sugieren que BMH se comporta mejor con alfabetos grandes.

En un trabajo posterior, Mahmoud, Smythe y Régnier [17] extienden los resultados anteriores usando funciones generadoras de momentos para extraer una teoría de distribuciones para el problema. En particular, obtienen resultados de normalidad para el número de cabezas y el número de comparaciones para un patrón fijo. Esto es, si llamamos $H_n = H_n(P)$ a la variable aleatoria que cuenta el número de cabezas en texto aleatorio de tamaño n para un patrón fijo P, y análogamente $C_{BMH}(n)$ el número de comparaciones, entonces, para constantes μ_P, s_P, c_P y S_P que dependen del patrón, se tiene

$$\begin{aligned} \frac{H_n - \mu_P\ n}{\sqrt{n}} &\xrightarrow{Dist} \mathcal{N}(0, s_P^2), \\ \frac{C_{BMH}(n) - c_P\ n}{S_P\sqrt{n}} &\xrightarrow{Dist} \mathcal{N}(0, 1), \end{aligned}$$

donde las constantes son obtenidas principalmente a partir de la teoría de momentos. También encuentran un comportamiento mixto en el caso de patrón aleatorio.

Ya hemos discutido varios resultados concernientes al desempeño de BMH, tanto en el peor y mejor caso como en el promedio, destacando las diferencias existentes. Sin embargo, se puede ir un poco más lejos: ya hemos comentado que, con todo lo que implica el análisis para los casos peor, mejor y promedio, en rigor puede ser que ninguno de ellos sea una imagen fiel del desemepeño del algoritmo en la práctica. Ya Horspool, al presentar el algoritmo, hizo una especial mención (apoyado en evidencia empírica, como hicimos notar arriba) a la mayor velocidad, medida tanto en tiempo de ejecución como en número de comparaciones, que su variante tenía frente al algoritmo original, a pesar de su mayor simpleza. Un algoritmo que presente una gran diferencia en desempeño entre el mejor y peor caso es bastante común, e incluso hay varios algoritmos con notables diferencias entre el peor caso y el caso promedio, pero lo que hace especial a BMH es el hecho de que, a pesar de tener un peor caso muy malo (mucho peor que otros algoritmos de búsqueda de patrones), el caso promedio y el mejor caso son muy buenos, mejores que para otros algoritmos y, más aún, el caso promedio se parece mucho más al mejor caso que al peor. Esto, unido al buen comportamiento del algoritmo en la práctica, sugiere fuertemente que las instancias donde BMH tiene un desempeño como el del peor caso o similar son extraordinariamente aisladas, y no son en ningún caso una representación fiel del espacio de entradas para el algoritmo. Son precisamente estas razones las que hacen a este algoritmo un candidato atractivo a revisar bajo el enfoque de análisis suavizado.

2.4. Enfoque

Nos concentramos en la cantidad de operaciones que realiza el algoritmo al buscar el patrón P en el texto T. Las preguntas principales que buscaremos responder son

las siguientes:

- Dado el patrón P, ¿Cuál es el texto T_{Hmin} que minimiza el número de cabezas en la ejecución del algoritmo? ¿Cuál es el texto T_{Cmin} que minimiza el número de comparaciones?
- Dado el patrón P, ¿Cuál es el texto T_{Hmax} que maximiza el número de cabezas? ¿Cuál es el texto T_{Cmax} que maximiza el número de comparaciones?
- ¿Qué tan sensibles a perturbaciones son los textos de las preguntas anteriores? En particular: ¿Cuántas comparaciones se hacen en un texto perturbado, y cómo se compara esto con los textos T_{Cmin}, T_{Cmax}?

Como se deduce de las preguntas anteriores, el enfoque principal es buscar, dado el patrón P, los diversos textos que minimizan o maximizan, según corresponda, el número de comparaciones. Sin embargo, otro parámetro importante es el tamaño σ del alfabeto, ya que afecta los posibles textos que se pueden generar, y por lo tanto influye notoriamente en el desarrollo de nuestro análisis y la respuesta a las preguntas planteadas arriba.

Las preguntas del primer ítem resultan relativamente fáciles de atacar y dedicamos el resto de esta sección a hacerlo. Las preguntas de los demás items requieren un análisis mucho más delicado y a ellas nos dedicaremos a lo largo de varias secciones futuras.

Observación: De ahora en adelante, por simplicidad, notaremos $K_{P_m} = K_m$, el shift asociado al último caracter del patrón.

2.4.1. Encontrando T_{Hmin} y T_{Cmin}

Podemos separar el análisis en casos. Primero consideramos $P = P_1 \dots P_m$ y supongamos que P es tal que existe $\alpha \in \Sigma$ tal que $\alpha \notin P$. Es muy fácil notar que en el texto $T = \alpha^n$ se hace una sola comparación por cabeza antes de avanzar m caracteres gracias al shift. Esto implica que $H = C = \lfloor n/m \rfloor$ que, como mencionamos en una sección anterior, es el mejor caso posible para las comparaciones, sencillamente porque si el algoritmo deja de revisar una ventana de tamaño mayor a m, en esa ventana puede haber una ocurrencia de el patrón que el algoritmo dejaría de detectar. Luego no es posible hacer menos comparaciones. Esto determina fácilmente T_{Hmin}, que es de hecho igual a T_{Cmin}.

El anterior es un caso particularmente optimista. Supongamos por ejemplo que no hay ningún carácter en Σ que no esté en P, pero el patrón es tal que el carácter

P_m no aparece más de una vez, esto es, $P_i \neq P_m$ para $i < m$. En ese caso podemos tomar $T = (P_m)^n$ de tal forma de hacer exactamente dos comparaciones locales en cada cabeza, antes de avanzar m caracteres a la derecha por el shift. Así se tiene que en este texto

$$\begin{aligned} H &= \left\lfloor \frac{n}{m} \right\rfloor, \\ C &= 2 \left\lfloor \frac{n}{m} \right\rfloor . \end{aligned}$$

Sin embargo, es posible, dependiendo del patrón, que este no sea el mejor caso para las comparaciones. Estamos asumiendo que todo el alfabeto está en el patrón. Sea $\alpha \in P_1 \dots P_{m-1}$ tal que K_α es el mayor entre todos los caracteres de $P_1 \dots P_{m-1}$. Si consideramos ahora el texto $T = \alpha^n$, se hace una sola comparación local, y luego

$$\begin{aligned} H &= \left\lfloor \frac{n}{K_\alpha} \right\rfloor, \\ C &= \left\lfloor \frac{n}{K_\alpha} \right\rfloor . \end{aligned}$$

De esta forma se tiene que, si $K_\alpha > m/2$, entonces $T_{Hmin} = (P_m)^n$, mientras que $T_{Cmin} = \alpha^n$, ya que en este último texto se hacen el mínimo de comparaciones locales (una), antes de desplazarse a la derecha la mayor cantidad posible sin tener a P_m como cabeza (lo que daría lugar a dos comparaciones locales lo que según la hipótesis $K_\alpha > m/2$ da lugar a un texto con más comparaciones). En caso contrario se tiene que $T_{Hmin} = T_{Cmin} = (P_m)^n$.

Finalmente, consideramos el caso en que todo el alfabeto está en P y el carácter P_m aparece al menos dos veces en el patrón. Supongamos primero que K_m es el mayor de todos los shifts. Entonces en el texto $T = (P_m)^n$ se hacen de nuevo 2 comparaciones locales, antes de moverse K_m caracteres por el shift. Así

$$\begin{aligned} H &= \left\lfloor \frac{n}{K_m} \right\rfloor, \\ C &= 2 \left\lfloor \frac{n}{K_m} \right\rfloor . \end{aligned}$$

Si, por el contrario, K_m no es el mayor de los shifts, entonces tomamos el carácter con mayor shift α y $T = \alpha^n$. Así se tiene

$$\begin{aligned} H &= \left\lfloor \frac{n}{K_\alpha} \right\rfloor \\ C &= \left\lfloor \frac{n}{K_\alpha} \right\rfloor, \end{aligned}$$

lo que determina, como arriba y por las mismas razones dependiendo de los valores de K_α respecto de K_m, el mejor texto para la cantidad de cabezas y la cantidad de comparaciones respectivamente.

Ya hemos obtenido algunos resultados en cuanto a los mejores resultados posibles en diversos casos. En las secciones siguientes, vamos a hacer el trabajo necesario para responder las preguntas referentes a la cantidad máxima de cabezas y comparaciones, e intentar vincularlas al análisis suavizado del algoritmo BMH.

Capítulo 3

Digrafo de desempeño

En esta sección nos interesa analizar el desempeño de BMH bajo diversas condiciones y para cada conjunto de parámetros para el algoritmo. Nuestro enfoque, como anticipamos en la sección anterior, es el siguiente: dados P y Σ, queremos saber cuál es el peor texto en el sentido de que maximiza el número de comparaciones para BMH. Para responder esta pregunta usaremos una herramienta que podemos construir para cada par (P, Σ) que procederemos a describir: un digrafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ con diversas propiedades que ayudarán a ilustrar y en cierta forma medir aspectos relacionados con la ejecución de BMH. Más aún, como veremos en esta sección y la siguiente, el digrafo será de utilidad para analizar la complejidad de BMH tanto en referencia a la pregunta de cuál es el peor texto, como en otras relacionadas con otros aspectos de complejidad.

Comenzaremos con algunos resultados que son la base de nuestro enfoque y el sostén de casi todo el trabajo realizado. Continuaremos con la descripción de los elementos que compondrán el digrafo a partir de los parámetros P y Σ. A partir de esta descripción se hará relativamente evidente la existencia de un algoritmo que, usando dichos parámetros como entrada, puede construir el digrafo de desempeño, y, finalmente, procederemos a utilizar dicho digrafo para nuestros propósitos.

3.1. Resultados de subaditividad

La presente sección tiene como objetivo principal describir el comportamiento asíntótico de BMH en base a algunos resultados. En particular, nos interesa encontrar límites para $C(T)/n$ cuando $|T| = n \to \infty$, donde recordamos que $C(T)$ es el número de comparaciones patrón-texto que se hacen en una ejecución de BMH en el texto T. Nos será muy útil el Teorema de Fekete, que enunciamos a continuación:

Teorema 3.1.1. ***Fekete [9].*** *Si $\{x_n\}_{n\geq 1}$ es una secuencia de números reales tal que para cada par de enteros n, ℓ se cumple la condición subaditiva*

$$x_{n+\ell} \leq x_n + x_\ell \ ,$$

entonces se tiene que

$$\lim_{n\to\infty} \frac{x_n}{n} = \inf_{n\geq 1} \frac{x_n}{n}.$$

Es fácil extender este teorema a secuencias un poco más generales. Si reemplazamos la condición subaditiva por una levemente más débil: $x_{n+\ell} \leq x_n + x_\ell + q$, donde q es una constante, entonces se puede ver que la secuencia $y_n = x_n + q$ cumple la condición subaditiva tal como está enunciada en el teorema y el resultado aplica para y_n, de donde es fácil heredarlo hacia x_n.

Corolario 3.1.2. *Si $\{x_n\}_{n\geq 1}$ es una secuencia de números reales tal que para cada par de enteros n, ℓ se cumple, para q constante, que $x_{n+\ell} \leq x_n + x_\ell + q$, entonces se tiene el mismo resultado que en el Teorema 3.1.1.*

3.1.1. El enfoque subaditivo en KMP

Szpankowski ya exploró los comportamientos subaditivos para el algoritmo KMP para búsqueda de patrones [8]. Para W un texto cualquiera, llamaremos $C_{KMP}(W)$ a la cantidad de comparaciones realizadas por KMP en W.

Szpankowski utilizó el Teorema Subaditivo Ergódico [10, 11, 12], una generalización del Teorema de Fekete para secuencias de variables aleatorias. Gracias a ello, logró demostrar que la esperanza del número de comparaciones de KMP para un texto aleatorio es asintóticamente igual a una constante (que depende del patrón), incluyendo resultados en cuanto a cotas de concentración. Además, obtuvo un resultado análogo para el comportamiento casi seguro del algoritmo en cuanto a número de comparaciones. También logró resultados similares para el caso determinista (texto dado). El trabajo de Szpankowski se basa en demostrar la siguiente desigualdad: Para todo patrón P y texto T con $|T| = n$, $|P| = m$, y para todo $1 \leq r \leq n$, se tiene

$$C_{KMP}(T_{1..n}) \leq C_{KMP}(T_{1..r}) + C_{KMP}(T_{r..n}) + 2m^2.$$

3.1.2. El enfoque subaditivo para BMH

Nos interesa encontrar resultados análogos a los presentados para KMP descritos en la sección anterior, pero para el algoritmo BMH. Esto sería de gran utilidad en el análisis suavizado del algoritmo. Sin embargo, como veremos de inmediato, no es posible replicar el análisis anterior, al menos para la secuencia presentada. Esto ocurre porque BMH no exhibe la misma conducta subaditiva que sabemos presenta KMP.

Contraejemplo

Para ver que la secuencia $C(T)$ no es subaditiva para el algoritmo BMH, consideremos el patrón $P = AAABAB$ y un texto como

$$\begin{aligned} T_{1..\ell-1} &= A^{\ell-1} \\ T_{\ell..n} &= (AAAABC)^{(n-\ell+1)/m}, \end{aligned}$$

donde tomamos n y ℓ de forma que $(n-\ell+1)/m$ sea entero. Esto quiere decir que a partir del ℓ-ésimo carácter de T, el texto consiste en repeticiones del bloque $AAAABC$. Vale la pena observar que, al escanear un texto $T_{r..s}$, la primera cabeza se ubica en la posición $r+m-1$ (en particular, en $T_{1..n}$ la primera cabeza se ubica en T_m). Ahora bien, es fácil ver que si tomamos $r = \ell$ entonces

$$C(T_{\ell..n}) = \frac{n-\ell+1}{m},$$

ya que la primera cabeza se ubica en $T_{\ell+m-1} = C$, y entonces el primer texto encontrado es el bloque $AAAABC$, en el que se hace exactamente una comparación antes de encontrar un mismatch con el patrón y avanzar m caracteres hacia la derecha dado que C no aparece en el patrón.

Sin embargo, si tomamos, por ejemplo, $r = \ell + 1$, ocurre que al hacer las comparaciones en el texto $T_{r..n}$ la primera cabeza que encontraremos será $T_{\ell+m} = A$, donde haremos una comparación local antes de avanzar un carácter a la derecha, donde de nuevo encontraremos una A. Esto seguirá hasta encontrar la cabeza B, donde, luego de 3 comparaciones locales, avanzaremos 2 caracteres a la derecha, por sobre el carácter C y encontraremos A como cabeza, repitiendo el proceso. Así, en cada bloque haremos 7 comparaciones y, escaneando el texto, veremos el bloque $(n-\ell)/m$ veces. Así, para $r = \ell + 1$

$$C(T_{\ell+1..n}) = 7\frac{n-\ell}{m},$$

y notamos así que la diferencia entre $C(T_{\ell..n})$ y $C(T_{\ell+1..n})$ no puede estar acotada por ninguna función independiente de n. Ahora bien, las primeras $\ell - 1$ posiciones del texto están escogidas de forma tal que, al escanearlo, todas las posiciones desde m hasta ℓ serán cabeza, ya que el carácter A produce un shift igual a 1, y luego en particular la posición ℓ será una cabeza. Luego, 7 comparaciones más adelante (las correspondientes al primer bloque $AAAABC$), la posición $\ell + m$ también será una cabeza. En ese momento, el análisis es el mismo que el que se hizo para el texto $T_{\ell+1..n}$. Sabemos, más aún, que cada una de las cabezas A implica exactamente una comparación local antes del shift. Con eso es fácil obtener que, para $\ell > m$,

$$C(T_{1..n}) = C(T_{1..\ell-1}) + 7 + C(T_{\ell+1..n}).$$

Por otro lado, se obtiene fácilmente que $C(T_{1..\ell}) = \ell - m + 1$ y $C(T_{1..\ell-1}) = \ell - m$. Con esto, una cota del estilo $C(T_{1..n}) \leq C(T_{1..r}) + C(T_{r..n}) + f(m)$ para $r = \ell$, sería equivalente a obtener que, para una función $f(m)$, se tiene

$$C(T_{\ell+1..n}) - C(T_{\ell..n}) \quad \leq \quad f(m) - 6.$$

Pero ya hemos mostrado que no se puede acotar la diferencia entre $C(T_{\ell+1..n})$ y $C(T_{\ell..n})$ por ninguna función independiente de n. Así, no se tiene la subaditividad como en KMP.

3.1.3. La secuencia $\max_{|T|=n} C(T)$

Dado $n \geq 1$, sabemos que debe existir algún texto (eventualmente más de uno) de tamaño n que maximice el número de comparaciones al buscar el patrón $P \in \Sigma^m$. Llamaremos un *peor texto* de tamaño n a $T \in \Sigma^n$ que alcance dicho máximo. Sea $x_n = \max_{|T|=n} C(T)$.

Veamos que x_n sí cumple la propiedad subaditiva.

Lema 1. Para todo patrón P, y para cada $m \leq r \leq n - m$ se tiene:

$$\max_{|T|=n} C(T) \quad \leq \quad \max_{|T|=r} C(T) + \max_{|T|=n-r+1} C(T) + 2m^2$$

Demostración: Probaremos que en los peores textos de tamaño n, r y $n-r+1$ se cumple la desigualdad deseada. Sean

- $A = A(P)$ un peor texto de tamaño n.
- $A' = A'(P)$ un peor texto de tamaño r.
- $A'' = A''(P) = A''_{r..n}$ un peor texto de tamaño $n - r + 1$.

Donde enumeramos el último texto desde r hasta n por comodidad.

Sea r_1 la posición más a la derecha en el texto A, menor o igual que $m-r+1$, que es una cabeza. Sea r_2 la posición más a la izquierda en A, mayor o igual a $m+r-1$, que es cabeza. Análogamente, sea r' la cabeza más a la derecha al revisar el texto A' (eventualmente se puede tener $r = r'$). La Figura 3.1 ilustra la situación descrita.

Ahora bien, por definición, sabemos que no existen posiciones entre r_1 y $r-m+1$ ni entre r_2 y $r+m-1$ que sean cabeza. A lo más, entonces, hay $2(m-2)+1$ posiciones entre r_1 y r_2 que pueden ser cabeza al leer el texto A. Por cada una de esas cabezas se pueden hacer a lo más m comparaciones locales. Luego:

$$C(A) \quad \leq \quad C(A_{1..r_1}) + C(A_{r_2-m+1..n}) + 2m^2.$$

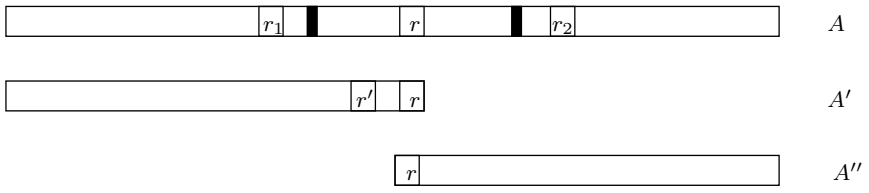

Figura 3.1: Textos A, A' y A'', con la posición relativa de r, r_1, r_2 y r'. En negro están marcadas las posiciones $r - m + 1$ y $r + m - 1$.

Como A' y A'' son respectivamente peores textos de tamaño r y $n-r+1$, es claro que $C(A_{1..r_1}) \leq C(A')$ y $C(A_{r_2-m+1..n}) \leq C(A'')$, de donde se obtiene la conclusión deseada. □

Por supuesto, el número de comparaciones hechas por BMH está acotado inferiormente por el mejor caso. Esto en particular implica que el ínfimo en el Teorema de Fekete es positivo. Con este resultado hemos demostrado el siguiente teorema:

Teorema 3.1.3. *Para cada alfabeto Σ y patrón P de tamaño m, existe $\gamma_0 = \gamma_0(P, \Sigma) \geq 1/m$ tal que*

$$\lim_{n\to\infty} \max_{|T|=n} \frac{C(T)}{n} = \inf_{n\geq 1} \max_{|T|=n} \frac{C(T)}{n} = \gamma_0$$

3.2. Motivación para la construcción del digrafo de desempeño

En una ejecución de BMH cada carácter en el texto puede ser comparado con el patrón varias veces, una o ninguna, dependiendo de diversos factores. Lo que sí sabemos es que por cada cabeza vamos a comparar hasta encontrar un mismatch patrón - texto (o bien hasta encontrar el patrón completo). Esto quiere decir que cada comparación local (donde no encontremos el patrón) se verá de la forma

$$\cancel{P_{m-\ell}} P_{m-\ell+1} \dots P_{m-1} P_m.$$

Ahora bien, algunos de los caracteres que son comparados con el patrón en la comparación local descrita pueden ya haber sido inspeccionados (es decir, el carácter en la posición correspondiente ya fue comparado con el patrón) en alguna comparación anterior, ya sea como cabeza o no. Supongamos que además de los caracteres agregamos una marca a las posiciones que son la cabeza actual o fueron una cabeza anterior. La comparación se podría ver de la forma

$$\cancel{P_{m-\ell}}\ \dot{P}_{m-\ell+1}\ \dots P_{m-1}\ \dot{P}_m,$$

donde los caracteres marcados con un punto fueron cabezas en alguna comparación local anterior o lo son en la comparación local actual. Por supuesto, cada una de las cabezas anteriores desencadena comparaciones locales que se ven básicamente de la misma forma, y vale notar que en cada comparación local el carácter de más a la derecha (que corresponde a la cabeza de la comparación local actual) **siempre** es una cabeza, es decir siempre tiene la marca correspondiente. Pongamos como ejemplo P =abcab. Vamos a ver algunos bloques de texto sobre el alfabeto {a,b,c} como objetos a través de los cuales podemos movernos. Por ejemplo, los bloques

cbcab
bacca
caabc

son bloques que pueden ser parte del texto T. Sin embargo hay dos de ellos, los dos últimos, que no pueden ser una comparación local en una ejecución del algoritmo como se vio más arriba, ya que en ellos luego de ver el carácter de más a la derecha (a y c respectivamente) se tendrá inmediatamente un mismatch con el carácter de más a la derecha del patrón y se abandonará la comparación local. Esto quiere decir que BMH no puede ver este tipo de bloques de texto (al menos no completos).

Tsung-Hsi Tsai ya exploró la idea de asociar algoritmos de búsqueda de patrones (BM y BMH) con un digrafo cuyos nodos son bloques de texto [25]. Nuestro enfoque es un poco más profundo: buscamos emular a través de un digrafo una ejecución de BMH. Esto es, dado un alfabeto y un patrón, nos interesa construir el texto que ve BMH o, más bien, todos los textos posibles que podría ver BMH, de manera de escoger aquel que sirva para nuestro análisis. Con este objetivo en mente comenzaremos a describir los nodos del digrafo.

3.3. Nodos

En primer lugar vamos a describir el conjunto $\mathcal{V}$ de nodos de nuestro digrafo de desempeño. La idea principal es hacer nodos que correspondan a bloques del patrón, leído desde el final hacia atrás, de forma de imitar lo que ve el algoritmo en una ejecución. Además de texto, los nodos tienen cabezas en posiciones adecuadas, de forma que un nodo corresponda a un bloque de texto tal como es visto por el algoritmo. Podemos decir que un nodo A toma la forma:

$$A \;=\; a_{m-r+1} \dots a_m,$$

donde $a_i \in \Sigma \cup \dot{\Sigma}$, con $\dot{\Sigma} = \{\dot{a} \mid a \in \Sigma\}$, y entonces $a_i \in \dot{\Sigma}$ indica la presencia de una cabeza en la posición i del nodo, para $i = m-r+1, \dots, m$. El número r es el *tamaño* del nodo A, y lo anotamos $|A|$. El tamaño de un nodo corresponde exactamente a

la cantidad de comparaciones que el algoritmo hace, desde a_m hacia la izquierda, en el bloque de texto $a_{m-r+1} \dots a_m$. Dividimos el conjunto de nodos en nodos de una cabeza y nodos multicabeza.

3.3.1. Nodos de una cabeza

Los nodos de una cabeza corresponden a bloques de texto que el algoritmo escanea una sola vez. Como sabemos de la descripción de BMH, en cada comparación local se empieza escaneando de derecha a izquierda y por lo tanto esta única cabeza está en la posición de más a la derecha: $a_m \in \dot{\Sigma}$, $a_i \in \Sigma$ para todo $i < m$.

Los primeros nodos de una cabeza son los de tamaño 1. En ellos, el algoritmo sólo hace una comparación antes de encontrar un mismatch con el patrón. Luego existen $\sigma - 1$ nodos de tamaño 1, uno para cada $\alpha \neq P_m$. Los nodos de una cabeza son como muestra la Figura 3.2 (obsérvese en la figura la marca sobre el carácter que indica la presencia de una cabeza en esa posición).

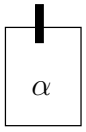

Figura 3.2: Nodos de una cabeza y tamaño 1.

De la misma manera podemos definir nodos de una cabeza de tamaño $r > 1$. Para $\alpha \neq P_{m-r+1}$, definimos $t = m - r + 2$ y consideramos los nodos de la forma que se ve en la Figura 3.3. El tamaño máximo de uno de estos nodos es la máxima distancia

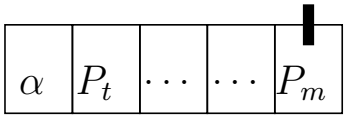

Figura 3.3: Nodos de una cabeza y tamaño r.

posible entre dos cabezas, es decir, el mayor de los K_α para $\alpha \in \Sigma$. Esto debido a que la distancia máxima entre dos cabezas es precisamente el mayor de los K_α. Luego, si el nodo fuera de mayor tamaño, necesariamente debe haber una cabeza más. Cuando dicho tamaño máximo es m, podemos agregar como nodo de una cabeza al bloque de texto que corresponde al patrón completo, lo que representa una situación donde se escanea dicho bloque de texto una vez y se encuentra el patrón (ver Figura 3.4)

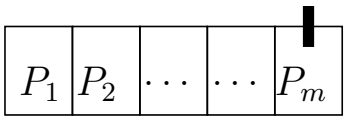

Figura 3.4: El patrón como bloque de texto de una cabeza.

3.3.2. Nodos multicabeza

Los nodos multicabeza se configuran de una manera parecida a los nodos de una cabeza, con la salvedad de que si $A = a_{m-r+1} \ldots a_m$, entonces añadimos la condición de que para cada $i < m$ con $a_i \in \dot{\Sigma}$ (una posición donde hay una cabeza) se debe tener $a_{i+j} \in \Sigma$ para $j = 1, \ldots, k-1$ y $a_{i+k} \in \dot{\Sigma}$, donde $k = K_{a_i}$. En la Figura 3.5 se ilustra la situación descrita.

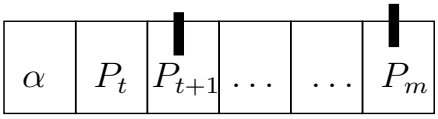

Figura 3.5: Un nodo multicabezas.

Llamamos *cabeza principal* de un nodo a la cabeza de más a la derecha, es decir, a la que corresponde a la posición m. Llamamos *cabezas secundarias* a las demás.

3.4. Arcos

Los arcos del digrafo conectan entre sí a bloques de texto que se pueden ver en una ejecución del algoritmo. Podemos distinguir dos casos: arcos sin traslape entre los bloques de texto y arcos con traslape entre los bloques de texto.

3.4.1. Arcos sin traslape

En este caso los textos de los dos nodos conectados son totalmente independientes entre sí, como muestra la Figura 3.6.

Para que la situación descrita suceda, es necesario que el shift dado por la última cabeza del primer nodo sea lo "suficientemente" grande. Esto es, el shift debe abarcar el suficiente espacio como para que en ninguna de las comparaciones subsecuentes se vuelva a comparar el patrón con el carácter de más a la derecha del primer nodo. Formalmente, si $A = a_{m-r+1} \ldots a_m$, $B = b_{m-s+1} \ldots b_m$ (observamos que en este caso $|A| = r$ y $|B| = s$), entonces existe el arco (A, B) sin traslape si y sólo si $K_{a_m} \geq j$, donde $m - s + j$ es el menor índice tal que $b_{m-s+j} \in \dot{\Sigma}$. En particular, para $s = 1$ (un nodo de la forma $\dot{b}_m$), se tiene $j = 1$, de donde sigue que todos los nodos inciden

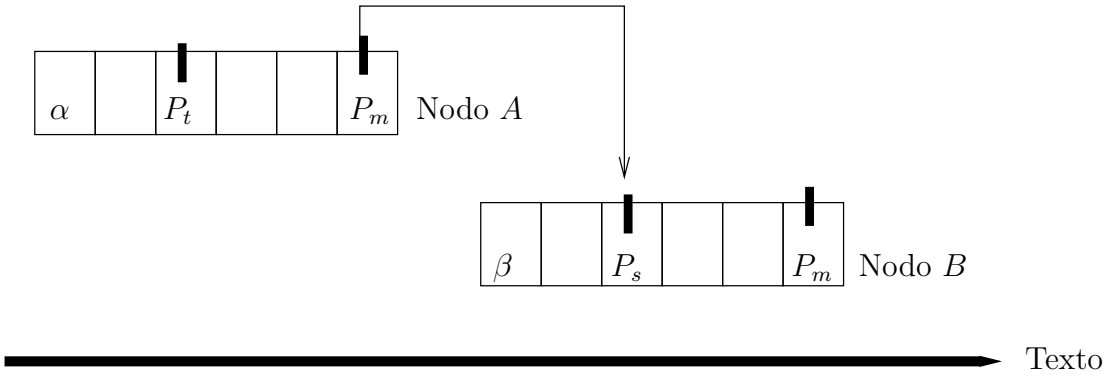

Figura 3.6: Dos nodos se conectan sin traslape entre los bloques.

sobre los nodos de tamaño 1.

Cuando un arco no tiene traslape diremos que existe un *gap* g, donde $0 \leq g \leq m-1$. El gap en cada arco es una propiedad que se relaciona íntimamente con el largo del arco, que formalizaremos y explicaremos con más detalle en la sección 3.5.

3.4.2. Arcos con traslape

Desde el punto de vista del texto, un arco con traslape ocurre cuando la cabeza principal de un nodo es una cabeza secundaria del nodo siguiente. Esto sólo puede ocurrir cuando el texto del nodo final es un sufijo del patrón. Así, el traslape entre los bloques de texto implica que en un arco de este tipo un sufijo del nodo inicial coincide tanto en texto como en cabezas con un prefijo del nodo final. Formalmente, esto quiere decir que dados $A = a_{m-r+1} \dots a_m$ y $B = b_{m-s+1} \dots b_m$, entonces una condición para que exista el arco (A, B) con traslape es

$$\exists\, t \in \{m-s+1, \dots, m-1\} \text{ tal que } b_t \in \dot{\Sigma} \text{ y } \forall\ j = m-s+1, \dots, t,\ a_{m-t+j} = b_j. \tag{3.1}$$

Notamos que en ese caso el t-ésimo carácter del nodo B es el más a la derecha que está en el traslape, y luego son en total $t-m+s$ los caracteres que están traslapados y $m-t$ los que no lo están. Llamaremos a esta situación un *t-traslape*.

Agregamos una condición extra que es más delicada y requiere algo más de notación y contexto, por lo que la justificaremos posteriormente. Queremos que, para que exista el arco (A, B) con traslape, el traslape *sea maximal*. Esto es, (A, B) es un arco con t-traslape si y sólo si se cumple la condición (3.1) para t y

$$s > t \ \Rightarrow\ \text{para ningún } C \in \mathcal{V} \text{ se cumple (3.1) para } s \tag{3.2}$$

La idea principal al crear el digrafo con todas las características que hemos visto es representar el texto mediante transiciones en $\mathcal{G}$. Ya sabemos que los nodos son

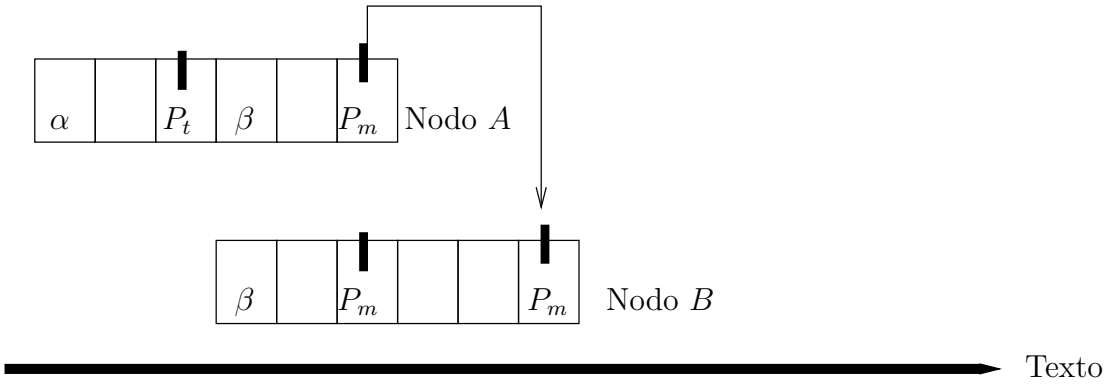

Figura 3.7: Traslape entre dos bloques de texto.

bloques de texto que pueden ser comparaciones locales de BMH, dado el patrón y el alfabeto. Consideremos un nodo cualquiera $A = a_{m-r+1} \dots a_m \in \mathcal{V}$. Queremos establecer una relación entre el texto que compone a los nodos y una porción de texto que puede revisar el algoritmo BMH. Más aún, nos interesan propiedades de esa porción de texto, tales como el número de comparaciones que BMH hace en ella, cuántas cabezas tiene, o cuánto se habrá desplazado a la derecha la ventana de búsqueda después de haber leído la porción de texto completa. Supongamos que A se conecta a un nodo $B = b_{m-s+1} \dots b_m$ mediante el arco (A, B) sin traslape. Según lo que sabemos de los arcos sin traslape, esto quiere decir que ninguna cabeza secundaria del nodo B es la cabeza principal del nodo A. Identificamos una porción de texto de tamaño $|B|$ dada exactamente por el texto que compone al nodo B. Si, por otro lado, A se conecta con B mediante un arco (A, B) con t-traslape, podemos identificar una porción de texto de tamaño $m - t$ dada exactamente por la porción de texto en B que no está en el traslape. Las Figuras 3.6 y 3.7 ilustran ambas situaciones y cómo ellas se relacionan con el texto: como se ve en ellas, al usar un arco (y por ende pasar de un nodo a otro) encontramos cada vez un bloque de texto **que no se había visto antes**, en el sentido de que recorriendo el texto de izquierda a derecha, asumiendo que ya tenemos acceso a los caracteres dados por el nodo A, encontramos un bloque de texto (de tamaño $|B|$ o $m - t$ respectivamente si no hay traslape o si hay un t-traslape) a los que no se tenía acceso anteriormente. Nos interesa asociar los arcos y trozos de texto mediante una correspondencia que llamamos τ, de forma que al recorrer cada arco en $\mathcal{E}$ se vaya escribiendo un texto que tenga que ver con el texto encontrado en los nodos comunicados por cada arco. Veremos un ejemplo para motivar y luego formalizaremos la correspondencia τ.

Consideramos el patrón $P = baa$ sobre $\Sigma = \{a, b\}$. Sabemos que debe haber un nodo $A = b\dot{a}\dot{a}$, un nodo $B = \dot{b}$, un nodo $C = \dot{a}\dot{a}\dot{a}$, entre otros. No es difícil deducir que existen los arcos (A, B) sin traslape y (A, C) con un 2-traslape. Podemos identificar el arco (A, B) con el texto b que es el que se ve en el nodo B (si ignoramos la cabeza),

e identificamos el arco (A, C) con el texto a que, descontando el traslape e ignorando las cabezas, corresponde al texto del nodo C. De esta forma, si consideramos una correspondencia de la forma

$$\begin{aligned} \tau(A, B) &= b, \\ \tau(A, C) &= a, \end{aligned}$$

entonces, cada vez que se recorre alguno de esos dos arcos, sabemos que se puede ir construyendo un texto dado por los caracteres que indica la correspondencia en cada caso. Vale la pena considerar otro caso: usando la misma notación, sabemos que existe el bucle (B, B) en un arco sin traslape. Ocupando la misma idea, la correspondencia sería $\tau(B, B) = b$. Sin embargo, como queremos usar el digrafo para representar secciones de un texto que se ven durante ejecuciones de BMH, debemos hacer una salvedad: sabemos que $K_b = 2$, luego al estar en el primer nodo B y leer la cabeza $\dot{b}$ vamos a avanzar dos posiciones hacia la derecha. Esto determina una posición exactamente del texto, y deja una sin determinar, como ilustra la Figura 3.8. Aunque esto no determine por completo el texto, determina una familia de textos que tienen en común la manera en la que se ven cuando son escaneadas por el algoritmo BMH, ya que sabemos que la posición marcada con Σ en la Figura 3.8 no será vista por BMH (no será comparada con ningún elemento del patrón). El uso del símbolo Σ en la Figura 3.8 se explica como sigue: como el algoritmo no comparará la posición marcada con Σ con el patrón, esto es equivalente a decir que el algoritmo "no verá" esta posición, y entonces ahí puede haber cualquier carácter en Σ. Esto de hecho determina una familia de textos que son iguales en todas las posiciones que no están marcadas con Σ, y que son equivalentes en el sentido de que BMH los escaneará de la misma forma (en particular hará el mismo número de comparaciones). En general, si llamamos α al carácter en la cabeza principal del nodo de salida, observemos que esto está hecho de tal forma que entre la cabeza de más a la derecha del nodo de salida y la cabeza de más a la izquierda del nodo de llegada (en este caso particular el nodo de salida y el de llegada son el mismo) hay exactamente $K_\alpha - 1$ posiciones.

Ahora podemos definir formalmente la correspondencia entre arcos y texto: dado un arco (A, B), donde $A = a_{m-r+1} \dots a_m$ y $B = b_{m-s+1} \dots b_m$, con un t-traslape, la correspondencia $\tau(A, B)$ asocia al arco el texto consistente en los $m - t$ caracteres de más a la derecha de B (ignorando cabezas). Análogamente, para un arco sin traslape, la correspondencia asocia al arco un texto de tamaño $|B|$, que corresponde exactamente al texto, sin contar cabezas, que compone al nodo B. En ambos casos, se hace de tal forma que haya exactamente K_{a_m} posiciones entre el carácter correspondiente a la cabeza principal de A (que es a_m) y la cabeza de más a la izquierda de B que no sea parte del traslape (si existe un traslape).

Sea $\phi : \Sigma \cup \dot{\Sigma} \to \Sigma$ una función tal que $\phi|_\Sigma = \mathrm{id}$ y $\phi(\dot{\alpha}) = \alpha$ para $\dot{\alpha} \in \dot{\Sigma}$ (esto es,

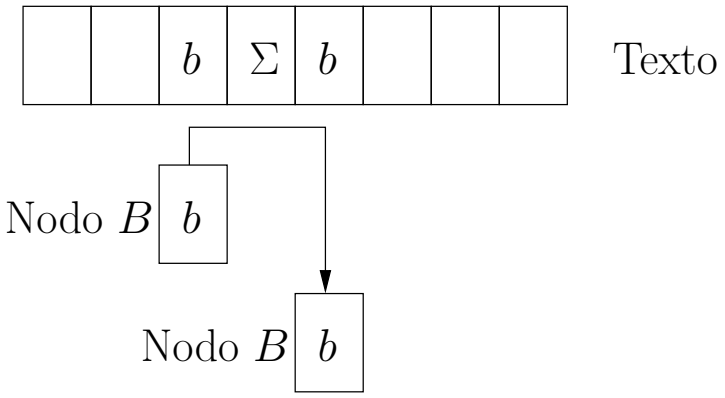

Figura 3.8: La correspondencia arcos-texto para el bucle (B, B).

una función que toma componentes de los nodos, ya sean cabezas o no, y les asocia el carácter correspondiente en Σ). Consideremos un nodo $B = b_{m-r+1} \dots b_m$. Se puede ver τ como una función que asocia arcos con expresiones regulares de la siguiente forma: si (A, B) es un arco con t-traslape entonces $\tau(A, B) = \phi(b_{m-t+1}) \dots \phi(b_m)$. Si, por otro lado, (A, B) es un arco sin traslape con un gap g, entonces $\tau(A, B) = \Sigma^g \phi(b_{m-r+1}) \dots \phi(b_m)$. Si tenemos un camino $\mathcal{R}$ que ocupa los arcos $e_1, \dots, e_j$, entonces extendemos esta correspondencia de la forma natural:

$$\tau(\mathcal{R}) \quad = \quad \tau(e_1)|| \dots ||\tau(e_j).$$

Ya hemos definido los nodos y arcos que componen el digrafo de desempeño que podemos construir en cada caso. Así, dado el patrón P y el alfabeto Σ, llamamos $\mathcal{G}(P, \Sigma) = (\mathcal{V}, \mathcal{E})(P, \Sigma)$, o sencillamente $\mathcal{G}$ si no hay confusión, al digrafo compuesto por el conjunto de nodos y arcos que acabamos de describir.

3.5. Costo y largo de los arcos

En esta sección definiremos dos funciones, $C : \mathcal{E} \to \mathbb{N}$ y $\ell : \mathcal{E} \to \mathbb{N}$, que llamaremos *costo* y *largo* respectivamente, que vincularán el digrafo $\mathcal{G}$ con el desempeño de BMH en cada caso. En lo que sigue especificaremos la forma en la que se calculan estas funciones.

Cada nodo, al ser un bloque de texto con ciertas posiciones que son cabeza, tiene asociado un número de comparaciones ejecutadas por el algoritmo. Dicha cantidad de comparaciones es el costo del nodo.

En los nodos de una cabeza esto es sencillamente el tamaño del nodo (es decir, el número de comparaciones que se hacen desde la cabeza principal hacia atrás),

pero en los nodos multicabeza la situación en más delicada. Tenemos que sumar a las comparaciones que se hacen desde la cabeza principal las comparaciones que se hacen en cabezas secundarias. Sin embargo, es fácil darse cuenta de que en un arco con traslape se cumple que la cabeza principal del nodo de salida corresponde a una cabeza secundaria en el nodo incidido. Luego es necesario tomar medidas para evitar problemas de doble conteo. Para ello, se analiza cada cabeza secundaria en cada nodo para ver si ella puede ser la cabeza principal de otro nodo. Así, para evitar el doble conteo, basta asegurarse de que el bloque de texto que va desde el inicio del nodo hasta la cabeza secundaria **no sea** un sufijo del patrón P.

En lo que sigue notaremos $\mathbb{1}_X(\cdot)$ a la función indicatriz clásica de un conjunto X. Para j tal que $a_j \in \dot{\Sigma}$, llamemos $\bar{j}$ al menor índice tal que $a_{\bar{j}+1} \dots a_m$ es un sufijo del patrón. Obsérvese que, de esta forma, las comparaciones locales que se hacen en la cabeza a_j son precisamente $j - \bar{j} + 1$. Sea $A = a_{m-r+1} \dots a_m \in \mathcal{V}$, y sea $B \in \mathcal{V}$ tal que $(A, B) \in \mathcal{E}$, y sea $m - r + 1 \leq k < m$ el mayor índice tal que $a_k \in \dot{\Sigma}$ y $a_{m-r+1} \dots a_k$ es un sufijo del patrón. Podemos definir formalmente el costo del arco (A, B) como

$$C(A, B) \;=\; r + \sum_{j=k+1}^{m-1} (j - \bar{j} + 1)\mathbb{1}_{\dot{\Sigma}}(a_j).$$

Está claro de esta definición que el costo es en rigor una propiedad de los nodos y no de los arcos. Por ello, ponemos el costo en los arcos salientes de cada nodo. Esto por supuesto implica que todos los arcos salientes del mismo nodo tienen igual costo.

Teniendo una noción del costo de los arcos, tiene sentido preguntarse por los valores máximos o mínimos que puede tomar dicho costo. Como se vio más arriba, dado que el costo de un arco debe representar de alguna forma el número de comparaciones que se hacen en un bloque de texto dado por el nodo de donde sale el arco, una cota inferior directa para el costo es el tamaño de dicho nodo. Es decir que, dado $A \in \mathcal{V}$, para todo $B \in \mathcal{V}$ tal que $(A, B) \in \mathcal{E}$, se tiene $C(A, B) \geq |A|$. Por otro lado, en el peor de los casos todos los caracteres del nodo $A = a_{m-r+1}, \dots, a_m$ son cabezas ($a_i \in \dot{\Sigma}$ para cada $i = m - r + 1, \dots, m$). En ese caso, el peor costo que se puede pagar debido al nodo A es de $|A|^2$, lo que da una primera cota superior para el costo de cada arco $(A, B) \in \mathcal{E}$.

El largo es una propiedad de los arcos, ya que depende de los dos nodos que se conectan. El largo corresponde a la cantidad de caracteres que avanza hacia la derecha el algoritmo dentro del texto cuando pasa por los nodos que conecta cada arco. Para $A = a_{m-r+1} \dots a_m$ y $B = b_{m-s+1} \dots b_m$, el largo del arco (A, B) es

$$\ell(A, B) \;=\; K_{a_m} + \sum_{j=t+1}^{m} K_{b_j} \mathbb{1}_{\dot{\Sigma}}(b_j),$$

donde $t \in \{m-s+1, \ldots, m-1\}$. Como en la definición de traslape, t es el carácter más a la derecha en el nodo B que está en el traslape ($t = 0$ si no hay traslape). En el caso sin traslape notamos que ahora se puede definir formalmente el gap g: para el arco (A, B) sin traslape, se tiene $g = \ell(A, B) - |B|$.

3.6. Resultados preliminares

Si aspiramos a usar el digrafo que hemos construido para analizar el algoritmo BMH, será necesario antes probar algunos resultados. La mayoría de ellos se deducen a partir de las definiciones, ya sea de nodos o de arcos, y están orientados a corregir posibles problemas de ambigüedad o de doble conteo de caracteres, cuando un paseo en el digrafo sea utilizado para simular una ejecución del algoritmo.

En primer lugar, notamos que es posible que dos nodos A y B cumplan características tanto de arco con traslape como sin, o bien que haya dos traslapes posibles. Esto último quiere decir que, en la definición de arcos con traslape, existen $t_1 < t_2$ ambos cumpliendo (3.1) en cada caso.

Por otro lado, consideremos una situación como la que se ilustra en la Figura 3.9. Si sólo consideramos la condición (3.1) (y no (3.2)), según lo que se ve en la figura, deberían existir tanto los arcos (Z, A) como (Z, B) y (A, B), ya que todos ellos cumplen las condiciones necesarias para formar un traslape. Es fácil notar que el camino ZAB y el arco (Z, B) representan la misma situación en cuanto a texto según la correspondencia τ que definimos en la sección de arcos con traslape, pero queremos evitar esta ambigüedad.

Es necesario evitar el tipo de situaciones descrita en el párrafo anterior para usar correctamente el digrafo como herramienta. Notemos que los problemas mencionados se pueden evitar si exigimos, cuando existe un arco con traslape entre dos nodos, que dicho traslape sea maximal, en el sentido de que debe ocurrir en la cabeza secundaria de más a la derecha del nodo final tal que el texto de dicho nodo, desde esa cabeza hasta atrás, sea un sufijo del patrón, sin ningún error. Esta es la razón por la cual agregamos la condición (3.2) a la condición (3.1) como exigencia para que exista un arco con traslape. Esto elimina la ambigüedad. Más generalmente, esto elimina siempre la ambigüedad, ya que imponemos que todos los arcos que llegan con traslape a un cierto nodo lo hagan siempre con el mismo traslape. Es decir, la condición (3.2) se traduce en que, dado $B \in \mathcal{V}$ tal que un nodo A incide sobre él con un t-traslape, entonces todos los nodos que inciden sobre él deben tener el mismo t-traslape. De lo contrario encontraremos un problema al momento de contar las operaciones usando la función de costo. Hemos probado lo siguiente:

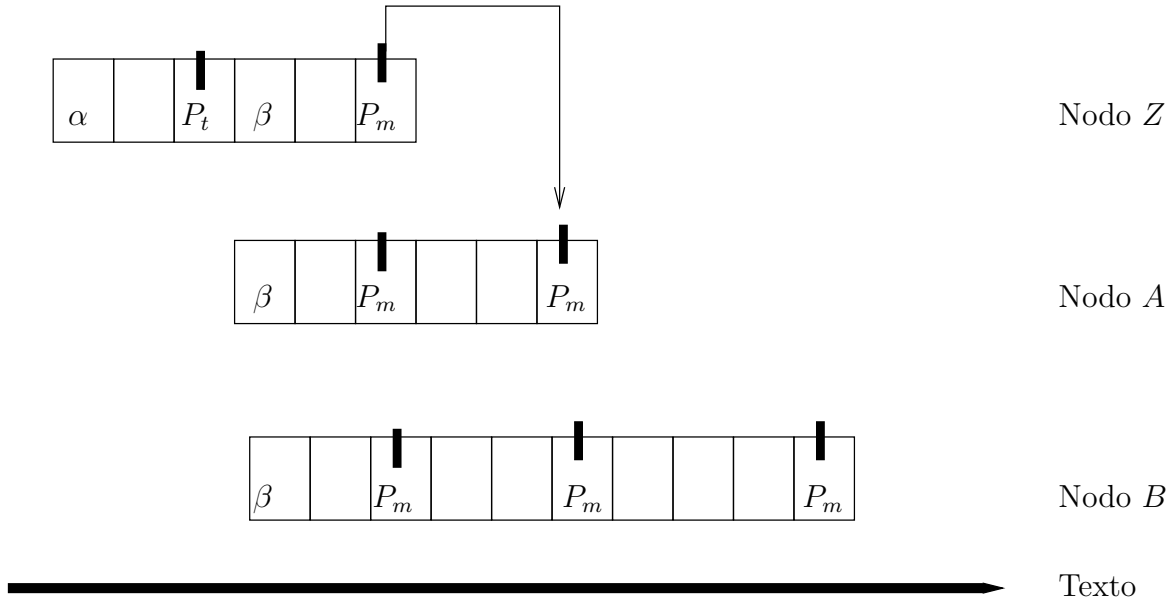

Figura 3.9: Ambigüedad en cuanto a traslapes y la relación con el texto.

Proposición 1. Sea $B \in \mathcal{V}$ tal que los nodos que inciden sobre él se conectan con traslape. Entonces el traslape es siempre el mismo, definido por (3.1)-(3.2). En particular, todos los arcos $(A, B) \in \mathcal{E}$ tienen el mismo largo.

Puede surgir la preocupación, si imponemos la condición (3.2) para todos los arcos con traslape, de cómo ello afecta a la correspondencia τ en cuanto a la construcción de texto que las transiciones en el digrafo implican. En particular, nos interesa asegurar que imponiendo esta condición se puede todavía, mediante τ, generar los mismos textos que se podría si es que no lo impusiéramos. El siguiente lema nos asegura que así es.

Lema 2. Sean A, B y C nodos tales que existe un t_1-traslape entre A y B y un t_2-traslape entre C y B, con $t_1 < t_2$. Entonces existe un nodo A' tal que $(A', B) \in \mathcal{E}$ con un t_2-traslape y tal que $\tau(A', B) = \tau(A, B)$.

Demostración: Basta con extender el nodo A hacia la derecha: por la hipótesis sobre el nodo C existe $t_2 > t_1$ tal que $b_{t_2} \in \dot{\Sigma}$ y de allí hacia atrás el texto en B es un sufijo del patrón. Tomamos entonces $X = b_{t_1+1} \dots b_{t_2}$ y $A' = A||X$, que cumple (3.1), tiene un t_2-traslape con B, y como la secuencia de cabezas que lo componen es factible (ya que lo es en A y en B), se tiene que $A' = A||X \in \mathcal{V}$. Como los caracteres que se agregan a la derecha para formar A' son todos caracteres que forman parte de B, sigue que $\tau(A, B) = \tau(A', B)$. □

El siguiente resultado asegura la utilidad del digrafo para representar ejecuciones de BMH. Esto es, mostraremos que los caminos en $\mathcal{G}$ generan textos que tienen largo

y costo (en cantidad de comparaciones patrón texto) equivalente al de los arcos que se transitan en dicho camino. Más aún, la posición de las cabezas está también dada por los nodos que se recorren en cada camino.

Lema 3. Sea T un texto y sea P un patrón, ambos sobre el alfabeto Σ. Entonces la ejecución de BMH en T buscando P, salvo a lo más por las primeras $m-1$ cabezas, se puede representar de manera única como un camino en $\mathcal{G}$. Conversamente, un camino en $\mathcal{G}$ representa una ejecución de BMH buscando P en una familia de textos.

Demostración: Para $h \in \mathcal{H}$, el conjunto de cabezas, llamamos $\overline{h}$ a la posición más a la izquierda en el texto que se compara con el patrón, asociado a la cabeza h (esto por supuesto implica que $T_{\overline{h}+1..h}$ es un sufijo de P).

Sea $h_1 \in \mathcal{H} \cap \{m, \ldots, 2m-1\}$ tal que $\overline{h}_1$ es mínimo y $\overline{h}_1 \geq m$. Si hay más de un elemento en la intersección que cumpla esas condiciones, tomamos la cabeza de más a la derecha como h_1. Definimos recursivamente una secuencia de cabezas. Dado h_j, escogemos h_{j+1} como

$$h_{j+1} \quad = \quad \text{máx}\{\arg\min_{h>h_j} \overline{h}\},$$

decidiendo igual que como en h_1 cuando haya ambigüedad. Esto define una secuencia creciente de cabezas en T, digamos $\{h_j\}_{j=1}^{t}$ que recorren completamente el texto T en el sentido de que toda la porción de T que es escaneada en la ejecución BMH lo es también cuando consideramos solo las cabezas en nuestra secuencia.

Podemos considerar entonces los trozos de texto $T_{\overline{h}_j..h_j}$, con cabezas entre $\overline{h}_j$ y $h_j - 1$ según corresponda para la ejecución considerada (y una cabeza principal en h_j). Con esto, definimos la secuencia de nodos $\{A_j\}_{j=1}^{t}$ determinadas por los trozos de texto y las cabezas recién mencionados, que existen por construcción.

Observación: Como sabemos que ninguna de las $m-1$ posiciones más a la izquierda en el texto puede ser una cabeza, puede ocurrir que las primeras comparaciones locales lleven a textos $T_{\overline{h}_j..h_j}$ que no tienen un nodo correspondiente cuando consideramos las cabezas en el texto. Esto ocurre a lo más para las primeras $m-1$ cabezas, de ahí que el resultado se puede asegurar para la porción de ejecución del algoritmo que sobrepasa las primeras $m-1$ posiciones.

Así, lo único que queda por probar es que en cada caso el arco (A_j, A_{j+1}) existe. En efecto, si $\overline{h}_{j+1} > h_j$ es directo que el arco (A_j, A_{j+1}) existe (sin traslape), ya que no hay ninguna comparación asociada a una cabeza posterior a h_j que llegue más atrás que la posición $h_j + 1$. En caso contrario, por construcción los nodos A_j y A_{j+1} cumplen las condiciones para formar un arco con traslape, y por definición de $\overline{h}_j$, h_j,

$\overline{h}_{j+1}$ y h_{j+1} dicho traslape es maximal, según la definición dada más arriba. Sigue el resultado deseado.

Para la conversa, escogemos cualquier nodo $A^1 \in \mathcal{V}$, y definimos T_m como A^1_i, donde i es la posición más a la izquierda en A^1 que es una cabeza. A partir de esta posición definimos $|A^1|$ caracteres de T: $T_{m-i+1..m+|A^1|-i}$. Llamemos $h_1 = m+|A^1|-i$. Dado A^j y h_j, tomamos A^{j+1} como un vecino de A^j y $h_{j+1} = h_j + \ell(A^j, A^{j+1})$. Definimos entonces, usando la correspondencia τ, $T_{h_j..h_{j+1}} = \tau(A^j, A^{j+1})$, con lo que vamos formando el texto que se escanea en la ejecución de BMH. Esto no determina el texto completo: dependiendo de los h_j y los valores de $|A^j|$ puede ocurrir que posiciones del texto no sean definidas por este proceso (un ejemplo de ello se vio anteriormente en la Figura 3.8). Esto quiere decir que el texto T definido de esta forma no es único, pero sí es único en el sentido de que una ejecución de BMH va a escanear solo el trozo de texto que está únicamente determinado, independientemente de los demás caracteres. □

3.7. Cantidad de nodos

En esta sección daremos cotas para la cantidad de nodos en el digrafo en función de ciertos parámetros de P. En la siguiente tabla se muestra la cantidad máxima de nodos en el digrafo para distintos valores de m y distintos alfabetos. Para obtenerla, sencillamente se cicla sobre todos los patrones posibles de cada tamaño para cada alfabeto y se construye el digrafo según las reglas descritas hasta ahora. Se muestra también un patrón $P^{\max}$ que en cada caso es el que maximiza el número de nodos. Dicho patrón $P^{\max}$ no es necesariamente único, pero se incluye como referencia, en particular porque a medida que aumenta m se puede apreciar una tendencia en el tipo de patrones que maximizan el número de nodos: como se ve, la tendencia es tal que se puede maximizar el número de nodos en $\mathcal{G}$ con patrones que cumplen $P_m = P_{m-1}$ y luego, de derecha a izquierda, tienen exactamente una aparición de cada uno de los caracteres del alfabeto hasta donde es posible (cuando el alfabeto se acaba), repitiendo después el carácter α con el mayor K_α hasta completar los m caracteres.

m	Σ	$P^{\max}$	Nodos
4	$\{a,\ b,\ c\}$	abcc	19
4	$\{a,\ b,\ c,\ d\}$	abcc	28
4	$\{a,\ b,\ c,\ d,\ e\}$	abcc	35
4	$\{a,\ b,\ c,\ d,\ e,\ f\}$	abcc	42
5	$\{a,\ b,\ c\}$	aabcc	30
5	$\{a,\ b,\ c,\ d\}$	abcdd	40
5	$\{a,\ b,\ c,\ d,\ e\}$	abcdd	55
5	$\{a,\ b,\ c,\ d,\ e,\ f\}$	abcdd	66
6	$\{a,\ b,\ c\}$	aaabcc	46
6	$\{a,\ b,\ c,\ d\}$	aabcdd	61
6	$\{a,\ b,\ c,\ d,\ e\}$	abcdee	78
6	$\{a,\ b,\ c,\ d,\ e,\ f\}$	abcdee	96
7	$\{a,\ b,\ c\}$	aaaabc	74
7	$\{a,\ b,\ c,\ d\}$	aaabcdd	87
7	$\{a,\ b,\ c,\ d,\ e\}$	aabcdee	106
7	$\{a,\ b,\ c,\ d,\ e,\ f\}$	abcdeff	127

El siguiente resultado da una cota explícita, en términos del tamaño del patrón y el alfabeto, para el tamaño del digrafo.

Proposición 2. Para un alfabeto Σ de tamaño σ y $P \in \Sigma^m$, se tiene $|\mathcal{V}(P,\Sigma)| \leq m\sigma + \sigma(m-1)m/2$

Demostración: Contemos primero los nodos de una cabeza. Llamemos K^* al mayor shift que da el alfabeto Σ para el patrón P. Si $K^* < m$ entonces para $r = 1, \ldots, K^*$ hay exactamente $\sigma - 1$ nodos de tamaño r y una cabeza, ellos corresponden a sufijos del patrón con un error en la posición de más a la izquierda, dicho error es el que implica las $\sigma - 1$ posibilidades para los nodos en cada caso. Si $K^* = m$, por otro lado, entonces lo anterior se mantiene pero hay σ nodos de tamaño m (ya que se debe agregar el patrón como nodo de tamaño m).

Contemos ahora los nodos multicabezas. Es importante notar que en un nodo la cabeza de más a la izquierda determina todas las subsiguientes, esto es, si $A = a_{m-r+1} \ldots a_m$, entonces el menor índice j con $a_j \in \dot{\Sigma}$ determina todas las cabezas de más a la derecha. Esto quiere decir que, para $j = m - r + 1, \ldots, m - 1$, si la cabeza de más a la izquierda ocurre en a_j entonces si $r < m$ hay a lo más $\sigma - 1$ nodos distintos de tamaño r con dicha cabeza de más a la izquierda (correspondientes a los posibles caracteres $\not{P_{m-r+1}}$) y si $r = m$ hay a lo más σ nodos de tamaño r tal que el índice j represente a su cabeza de más a la izquierda.

Sumando ambos casos (nodos de una cabeza y nodos con su cabeza más a la izquierda en la posición $j < m$) se tiene

$$\begin{aligned} |\mathcal{V}| &\leq \sum_{r=1}^{m} \sigma + \sum_{j=1}^{m-1} \sum_{r=m-j+1}^{m} \sigma \\ &= m\sigma + \sigma \sum_{j=1}^{m-1} j \\ &= m\sigma + \sigma(m-1)m/2, \end{aligned}$$

que es el resultado buscado. □

3.8. Ciclos

Gracias al Lema 3 sabemos que la correspondencia τ asocia un texto (o familia de textos) a cada camino en $\mathcal{G}$. Sabemos además que una ejecución de BMH se puede representar de manera única como un camino en el digrafo que recorre la secuencia de nodos $A^j,\ j = 1, \ldots, t$ definidos en el lema. Así, un texto más largo debiera implicar más transiciones. Es decir, un camino más largo. En particular, para $n \to \infty$, vamos a hacer un número arbitrariamente grande de transiciones. Así, es inevitable encontrar ciclos. Cada ciclo tiene un costo y un largo asociado, y corresponde únicamente a una familia de textos.

Nos referiremos ahora con más profundidad a los ciclos. Sea $\mathcal{C}$ un ciclo en $\mathcal{G}$, donde llamamos $\mathcal{E}(\mathcal{C})$ a los arcos que el ciclo recorre. Notamos $C(\mathcal{C})$ el costo del ciclo y $\ell(\mathcal{C})$ su largo. Esto es, $C(\mathcal{C}) = \sum_{e \in \mathcal{E}(\mathcal{C})} C(e)$ y $\ell(\mathcal{C}) = \sum_{e \in \mathcal{E}(\mathcal{C})} \ell(e)$. Hablaremos de *costo amortizado* cuando nos refiramos al costo de un camino dividido por su largo.

Para investigar el valor de la constante γ_0, nos centraremos en investigar los ciclos de costo promedio máximo (CCPM). Es decir, aquellos ciclos $\mathcal{C}^*$ tales que $C(\mathcal{C}^*)/\ell(\mathcal{C}^*) \geq C(\mathcal{C})/\ell(\mathcal{C})$ para todos los ciclos $\mathcal{C}$ en $(\mathcal{V}, \mathcal{E})$. Esto se explica en los dos siguientes resultados. En primer lugar, nos interesa determinar si un CCPM puede ser simple, es decir, un ciclo que no pasa por un nodo más de una vez, salvo por el nodo inicial.

Lema 4. Para cada patrón P y alfabeto Σ, existe un CCPM simple en $\mathcal{G}(P, \Sigma)$.

Demostración: Sea $\mathcal{C}$ un ciclo no simple de largo mínimo, y sea A uno de los nodos de $\mathcal{C}$ al que se llega dos o más veces recorriendo $\mathcal{C}$. Podemos dividir $\mathcal{C}$ en dos ciclos

distintos $\mathcal{C}_1$ y $\mathcal{C}_2$, donde al menos uno de ellos pasa por A exactamente una vez. Se tiene

$$\begin{aligned} C(\mathcal{C}) &= C(\mathcal{C}_1) + C(\mathcal{C}_2), \\ \ell(\mathcal{C}) &= \ell(\mathcal{C}_1) + \ell(\mathcal{C}_2). \end{aligned}$$

Supongamos sin pérdida de generalidad que $C(\mathcal{C}_1)/\ell(\mathcal{C}_1) \geq C(\mathcal{C}_2)/\ell(\mathcal{C}_2)$. Así, se tiene que

$$\ell(\mathcal{C}_1)C(\mathcal{C}_2) \leq C(\mathcal{C}_1)\ell(\mathcal{C}_2).$$

Sumando $\ell(\mathcal{C}_1)C(\mathcal{C}_1)$ a ambos lados y factorizando, sigue que

$$\ell(\mathcal{C}_1)(C(\mathcal{C}_2) + C(\mathcal{C}_2)) \leq C(\mathcal{C}_1)(\ell(\mathcal{C}_1) + \ell(\mathcal{C}_2)).$$

Luego $C(\mathcal{C}_1)/\ell(\mathcal{C}_1) \geq C(\mathcal{C})/\ell(\mathcal{C})$. Pero $C(\mathcal{C})/\ell(\mathcal{C}) \geq C(\mathcal{C}_1)/\ell(\mathcal{C}_1)$ pues $\mathcal{C}$ es CCPM. Sigue que $\mathcal{C}_1$ es CCPM de largo menor a $\ell(\mathcal{C})$, contradiciendo la minimalidad. Concluimos que $\mathcal{C}$ debe ser simple. □

Los siguientes resultados capturan la propiedad más importante de los CCPM y su relación con la constante γ_0.

Lema 5. Sea T un texto con $|T| = n$ que recorre una cantidad cualquiera de ciclos distintos. Existe un texto $\overline{T}$, $|\overline{T}| = n$, que recorre un único ciclo y que cumple

$$C(\overline{T}) \geq C(T) - Q,$$

donde $Q > 0$ no depende de n.

Demostración: Sea $\mathcal{R}$ el ciclo de mayor costo promedio que se recorre al leer T. Es importante notar que tanto $C(\mathcal{R})$ como $\ell(\mathcal{R})$ no dependen de n. Sean $\mathcal{R}_1, \ldots, \mathcal{R}_t$ los ciclos (eventualmente enumerados con repetición) distintos a $\mathcal{R}$ que se recorren al leer T. Sean además $L_0, \ldots, L_{t-1}, L_t$ los largos de los caminos entre dos ciclos distintos en T (L_0 y L_t son los largos antes del primer y después del último ciclo respectivamente).

Sea $\overline{T}$ el texto que resulta de eliminar todos los ciclos de T, reemplazándolos por recorrer más veces el ciclo $\mathcal{R}$. La cantidad de veces extra que se recorre el ciclo $\mathcal{R}$ será r, donde

$$r = \left\lfloor \frac{\ell(\mathcal{R}_1) + \ldots + \ell(\mathcal{R}_t)}{\ell(\mathcal{R})} \right\rfloor.$$

Intentaremos calcular $C(\overline{T})$, el costo del nuevo texto $\overline{T}$. Este texto no tiene ninguno de los ciclos distintos de $\mathcal{R}$, por lo que hay que restar $C(\mathcal{R}_1) + \ldots + C(\mathcal{R}_t)$

a su costo total. Por otro lado, se agregan r repeticiones de $\mathcal{R}$, es decir que $C(\overline{T}) = C(T) + r \cdot C(\mathcal{R}) - C(\mathcal{R}_1) - \ldots - C(\mathcal{R}_t)$. Por definición de r, se cumple

$$r \leq \frac{\ell(\mathcal{R}_1) + \ldots + \ell(\mathcal{R}_t)}{\ell(\mathcal{R})} < r + 1,$$

luego se tiene que

$$\begin{aligned}(r+1)C(\mathcal{R}) &> \frac{\ell(\mathcal{R}_1) + \ldots + \ell(\mathcal{R}_t)}{\ell(\mathcal{R})}\, C(\mathcal{R}) \\ &= \left(\frac{\ell(\mathcal{R}_1)}{\ell(\mathcal{R})} + \ldots + \frac{\ell(\mathcal{R}_t)}{\ell(\mathcal{R})}\right) C(\mathcal{R}) \\ &\geq C(\mathcal{R}_1) + \ldots + C(\mathcal{R}_t),\end{aligned}$$

lo último gracias a que $C(\mathcal{R})/\ell(\mathcal{R})$ es el mayor costo promedio entre todos los ciclos que se recorren en T. Se tiene entonces que

$$C(\overline{T}) \geq C(T) - C(\mathcal{R}),$$

donde $C(\mathcal{R})$ es una constante positiva que no depende de n. □

Este argumento vale para cualquier texto de tamaño n. Llamemos T^{n*} a un texto tal que $C(T^{n*}) = \text{máx}_{|T|=n} C(T)$. Por el Lema 5, podemos construir $\overline{T}^n$ que consta de un camino antes de empezar con repeticiones de un solo ciclo $\mathcal{R}$ de costo $C(\mathcal{R})$, y finaliza con otro camino. Este texto es tal que $C(\overline{T}^n) \geq C(T^{n*}) - C(\mathcal{R})$, donde, como ya vimos, $C(\mathcal{R})$ no depende de n y, en particular, es despreciable comparada con n. Es decir

$$\frac{C(\overline{T}^n)}{n} \geq \frac{C(T^{n*})}{n} - \frac{O(1)}{n}$$

Tomando límite sobre n, y llamando $C(\mathcal{C}^*)$ y $\ell(\mathcal{C}^*)$ al costo y largo respectivamente de un CCPM, hemos probado el siguiente resultado:

Teorema 3.8.1. *Sea $\mathcal{C}^*$ un CCPM en $\mathcal{G}(P, \Sigma)$. Entonces se tiene*

$$\gamma_0(\Sigma, P) = \frac{C(\mathcal{C}^*)}{\ell(\mathcal{C}^*)}$$

3.9. El algoritmo de Karp

Ya sabemos que podemos encontrar el valor de la constante $\gamma_0(P, \Sigma)$ a partir de un CCPM una vez que hayamos construido $\mathcal{G}(P, \Sigma)$. Buscamos entonces una forma algorítmicamente lo más eficiente posible para encontrar dicho ciclo.

En primer lugar consideramos la siguiente transformación: Supongamos que $(A, B) \in \mathcal{E}$ cumple $\ell(A, B) = 2$. Si incluimos un nodo ficticio entre A y B de tal forma que haya exactamente dos arcos entre los nodos, entonces el largo del arco se traduce efectivamente en la cantidad de arcos que separan los dos nodos (o bien, la cantidad de pasos en que se llega de un nodo a otro). La Figura 3.10 ilustra la situación.

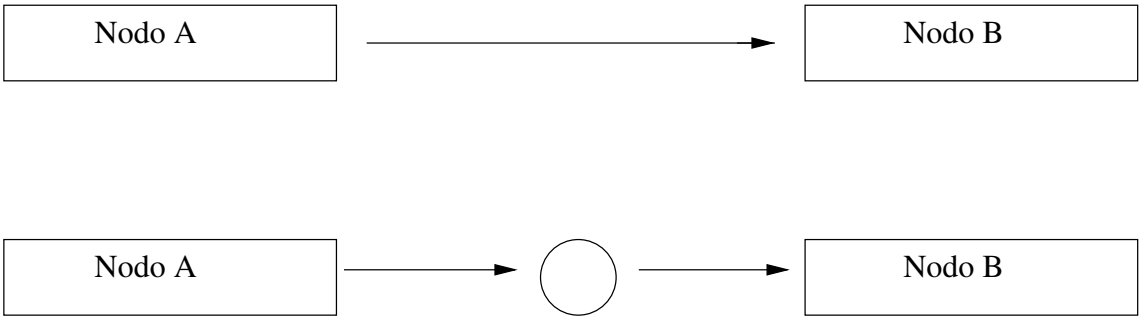

Figura 3.10: Insertando un nodo ficticio.

En general, para cualquier valor de $\ell(A, B)$ podemos hacer el mismo procedimiento insertando $\ell(A, B) - 1$ nodos ficticios entre A y B para lograr el mismo resultado. Además, agregamos la regla de que los arcos que inciden sobre nodos ficticios tienen costo nulo, de forma tal que el costo del arco original queda sobre el arco que incide sobre el nodo real B. Así, obtenemos un grafo $\mathcal{G}'(P, \Sigma) = (\mathcal{V}', \mathcal{E}')(P, \Sigma)$ en donde podemos encontrar el CCPM usando un enfoque clásico como el Algoritmo de Karp [26], que pasamos a describir en los párrafos siguientes y se basa en el teorema que enunciaremos a continuación. Primero, escogemos un $S \in \mathcal{V}$ arbitrario que llamamos *fuente* y definimos recursivamente, para $B \in \mathcal{V}$ y $1 \leq k \leq |\mathcal{V}| - 1$

$$D_k(B) \;=\; \max_{(A,B)\in\mathcal{E}} \left\{ D_{k-1}(A) + C((A, B)) \right\}, \tag{3.3}$$

con condiciones iniciales $D_0(S) = 0$ y $D_0(B) = -\infty$ para $B \neq S$. Así, $D_k(A)$ es el costo máximo de un camino de largo k desde la fuente hasta cada nodo A. Con esto podemos enunciar el Teorema de Karp.

Teorema 3.9.1 (Karp [26])**.** *El valor de un ciclo de costo promedio máximo λ^* de $(\mathcal{V}, \mathcal{E})$ está dado por*

$$\lambda^* \;=\; \max_{A\in\mathcal{V}} \min_{0\leq k\leq|\mathcal{V}|-1} \frac{D_{|\mathcal{V}|}(A) - D_k(A)}{|\mathcal{V}| - k} \tag{3.4}$$

Para implementar el algoritmo utilizamos una sencilla rutina de tres partes propuesta por Dasdan y Gupta [27].

1. Inicializamos el valor de D como $-\infty$ excepto para la fuente. Construimos además un arreglo Π que guarda, para $k = 0, \ldots, |\mathcal{V}| - 1$ y $B \in \mathcal{V}$ en $\Pi(k, B)$,

el nodo predecesor. Es decir, el nodo A tal que A logra el máximo en el lado derecho de (3.3).

2. Construimos, usando la ecuación (3.3), la lista completa de los valores de las funciones D y la lista de predecesores.

3. Calculamos el cuociente en (3.4) y tomamos el mínimo según k en cada caso, guardando el resultado en una lista de tantos valores como nodos, además del k particular que alcanza el mínimo en cada caso. Finalmente, con toda esta información, usamos la lista Π para construir el CCPM.

Observación: El algoritmo de Karp es polinomial en el tamaño del digrafo. De hecho, la complejidad de este paso es $\Theta(|\mathcal{V}'|^2 + |\mathcal{V}'||\mathcal{E}'|)$ [27].

3.10. Ejemplo: El peor caso

Vamos a ilustrar la creación del digrafo de desempeño con un ejemplo clásico: el peor caso para BMH con $\Sigma = \{a, b\}$ y $P = ba^{m-1}$. Para $m = 3$, construiremos $\mathcal{G}$ y encontraremos el CCPM. El patrón es $P = baa$.

En primer lugar, construimos los nodos de una cabeza: el único nodo de tamaño 1 es $A = \dot{b}$, ya que no hay más caracteres $\cancel{P_m}$. El otro nodo de una cabeza es $B = b\dot{a}$. No hay más nodos de una cabeza, dado que los únicos shifts disponibles para este alfabeto son $K_a = 1$ y $K_b = 2$. Luego no puede haber nodos de tamaño 3 con una cabeza. Los nodos de dos cabezas son $C = a\dot{a}\dot{a}$, $D = b\dot{a}\dot{a}$ y $E = \dot{b}a\dot{a}$, y vemos que no puede haber más ya que sólo hay dos caracteres en el alfabeto y hay sólo dos posiciones (aparte de la cabeza principal) donde poner la segunda cabeza. Llegamos así al único nodo de tres cabezas, $F = \dot{a}\dot{a}\dot{a}$.

Analizamos ahora los arcos: por motivos de espacio no vamos a ver todas las combinaciones, pero sí algunos arcos más interesantes. No es difícil ver que los nodos E y C cumplen (3.1)-(3.2), las condiciones para un 2-traslape. Por otro lado, el nodo F cumple las condiciones para un bucle con 2-traslape consigo mismo. Los nodos C y D cumplen también las condiciones para un 2-traslape con F, donde vemos cómo se cumple la Proposición 1. Ellos son todos los arcos con traslape de este digrafo. Usando ahora la definición de la función de largo ℓ, presentamos una tabla con todos los arcos, indicados por su largo, para el digrafo.

	A	B	C	D	E	F
A	2	2	-	3	4	-
B	1	-	-	-	3	-
C	1	-	-	-	3	1
D	1	-	-	-	3	1
E	1	-	1	-	3	-
F	1	-	-	-	3	1

Como los costos están en los arcos salientes de cada nodo, basta con que describamos el costo de cada nodo usando la función de costos C, y luego se heredan dichos costos a los arcos salientes. Como ya hemos mencionado, el primer paso para calcular el costo es determinar la posición j de más a la derecha en el texto de cada nodo que es un sufijo del patrón. Luego contamos la cantidad de comparaciones que, desde $j+1$ en adelante, hace BMH desde cada cabeza hacia atrás, y sumamos. Con esto:

Nodo	Costo
A	1
B	2
C	3
D	5
E	4
F	3

Así, ya tenemos completamente determinado el digrafo de desempeño. En este caso particular, no es necesario ejecutar el algoritmo de Karp para determinar cuál es el CCPM, dado que sabemos de una observación anterior que existe el bucle (F, F), con largo 1 y costo 3, por lo que este bucle (que es, por supuesto, un ciclo) tiene costo amortizado $3 = m$. Si existiese un ciclo de costo amortizado mayor, entonces se tendría $\gamma_0 > m$ lo que es claramente imposible. Luego encontramos que $\gamma_0(P, \Sigma) = m$ (lo que ya sabíamos), y más aún sabemos que el ciclo que maximiza el costo amortizado es $(\dot{a}\dot{a}\dot{a}, \dot{a}\dot{a}\dot{a})$, lo que de acuerdo a la correspondencia $\tau(\dot{a}\dot{a}\dot{a}, \dot{a}\dot{a}\dot{a}) = a$ implica un texto como $T = a^n$, que es precisamente el peor texto, como sabíamos de antes, para este caso (y para cualquier caso del algoritmo). El digrafo está ilustrado en la figura 3.11.

Ya hemos descrito la construcción y uso del digrafo y el algoritmo de Karp para encontrar los CCPM, y con ellos las constantes γ_0 de cada caso. Incluimos entonces en lo que sigue una tabla con algunos valores de $\gamma_0(P, \Sigma)$ para distintos P y Σ.

En la tabla, la columna "Texto" es la correspondiente al CCPM. Esto es, si el ciclo es $\mathcal{C}^*$, entonces lo que se lee en dicha columna es $\tau(\mathcal{C}^*)$. Son repeticiones de

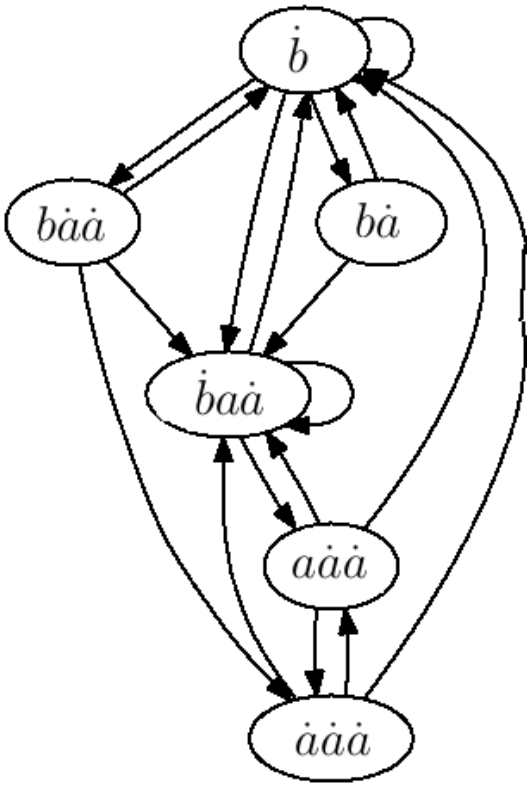

Figura 3.11: El digrafo de peor caso, $m = 3$.

este tipo de textos los que llevan al máximo de comparaciones totales en cada caso. Es muy interesante notar que en cada uno de estos casos (y en todos los estudiados hasta el momento), el CCPM es de hecho un bucle. Conjeturamos que el CCPM siempre es un bucle (el de mayor costo amortizado).

P	Σ	γ_0	Texto
$baba$	$\{a, b, c\}$	2	ba
$abba$	$\{a, b\}$	4/3	bba
$bacca$	$\{a, b, c\}$	5/3	cca
$abcca$	$\{a, b, c, d\}$	5/4	$bcca$
$babaa$	$\{a, b\}$	3	a
$bacba$	$\{a, b, c\}$	5/3	cba
$bccca$	$\{a, b, c\}$	1	c
aba	$\{a, b\}$	3/2	ba
bba	$\{a, b\}$	1	b
$cbacba$	$\{a, b, c\}$	2	cba

3.11. Estadísticas para patrones más largos

Veremos a continuación un histograma de frecuencias para patrones de tamaño $m = 4$. Fijando el carácter $P_m = a$ sobre el alfabeto $\Sigma = \{a, b, c\}$ (de manera de no considerar permutaciones sobre el alfabeto), ciclamos sobre los patrones de tamaño 4. Con la restricción $P_m = a$, esto da lugar a 27 patrones distintos cuyo valor de $\gamma_0(P, \Sigma)$ se calcula para cada caso usando el algoritmo descrito en este capítulo. La Figura 3.12 muestra los resultados, en donde mostramos la frecuencia, entre los 27 patrones considerados, de los valores encontrados para γ_0/m en una partición de $[0, 1]$ en intervalos de largo 0.05.

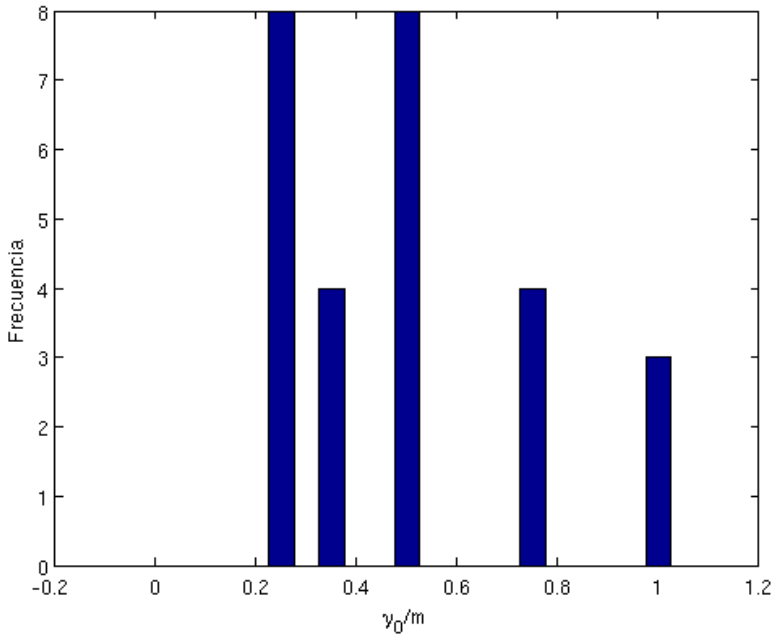

Figura 3.12: Frecuencias de γ_0/m para 27 patrones $m = 4,\ \Sigma = \{a, b, c\},\ P_m = a$.

Es interesante observar, de la Figura 3.12, que de hecho existe un gap no despreciable entre los peores valores de γ_0/m (que sabemos es igual a 1 en el peor caso) y los valores inmediatamente más pequeños. Esto es más evidencia que apunta a que las instancias de peor caso de este problema son pocas en comparación con las demás instancias del espacio de entradas del algoritmo. La figura 3.13 muestra la misma información pero de otro modo: dividiendo el intervalo $[0, 1]$ en 100 partes iguales, se cuenta el número de patrones P tales que $\gamma_0(P, \Sigma)/m \leq X$ para X en cada uno de los subintervalos considerados. Este tipo de gráficos sirve para ver el espacio de patrones y la complejidad que cada uno de esos patrones implica para el algoritmo.

De la misma forma incluimos resultados para $m = 5$ y el mismo alfabeto. En este caso los patrones a analizar son $3^4 = 81$, y se incluyen para este caso las mismas

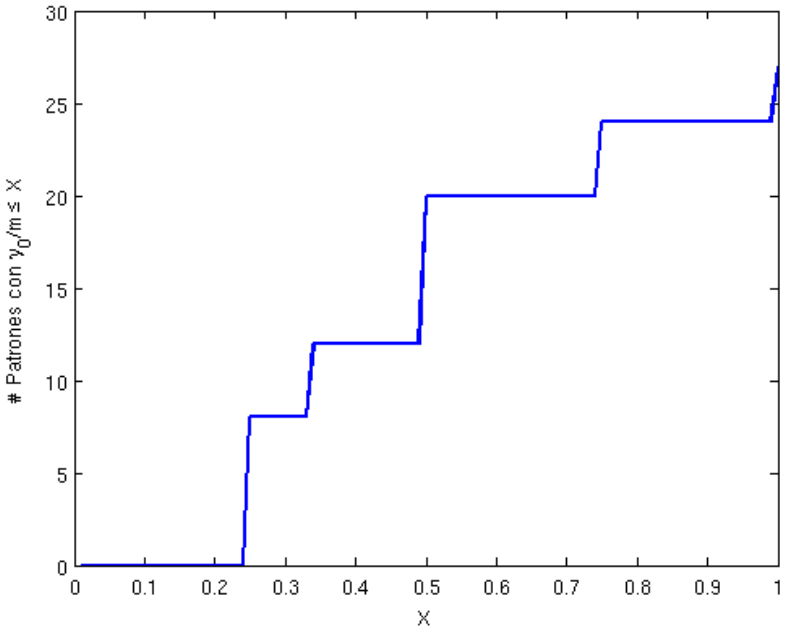

Figura 3.13: Número de patrones con $\gamma_0/m \leq X$, $X \in [0,1]$, $m = 4$.

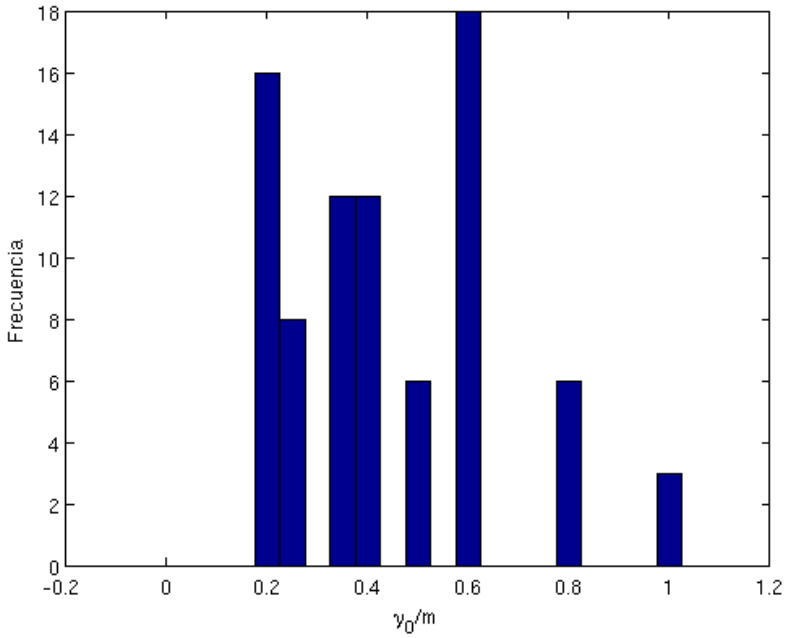

Figura 3.14: Frecuencias de γ_0/m para 81 patrones, $m = 5,\ \Sigma = \{a, b, c\},\ P_m = a$.

figuras que en el anterior. Las figuras 3.14 y 3.15 ilustran los resultados. Vale la pena recalcar que en ambos casos los patrones que alcanzan el peor valor de γ_0 (que es

m) son exactamente 3, y entre ellos y los valores inmediatamente menores existe una brecha no desepreciable. Más aún, la gran mayoría de los patrones, como se ve en los histogramas, quedan mucho más cercanos al extremo izquierdo, es decir, tienen valores de γ_0/m mucho más cercanos a $1/m$ que a 1. Aunque no se cubre en este trabajo, esto sugiere la pregunta de qué ocurre si el patrón es sometido a perturbación, particularmente qué pasaría con la constante γ_0 asociada.

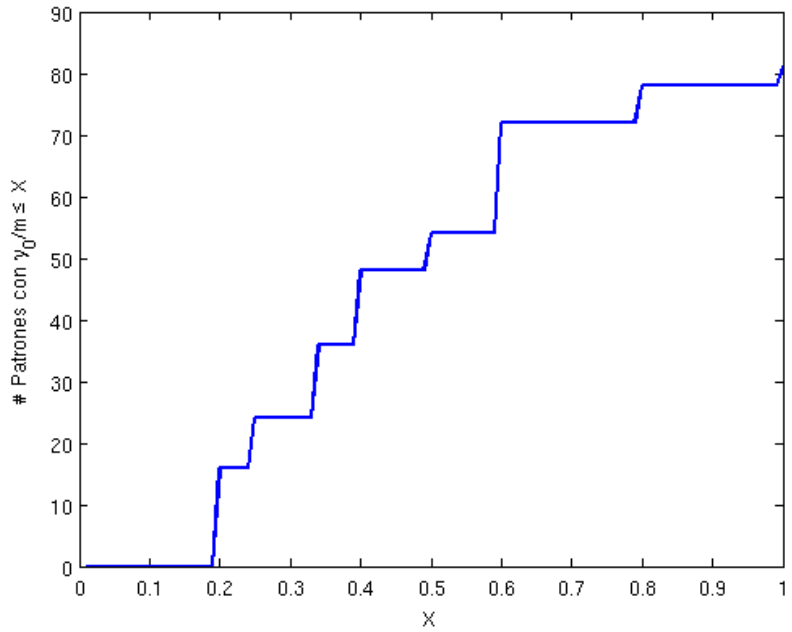

Figura 3.15: Número de patrones con $\gamma_0/m \leq X$, $X \in [0,1]$, $m = 5$.

Terminamos este capítulo con un resultado un poco más general. ¿Cuántas veces puede ser visto cada carácter del texto en una ejecución del algoritmo? Una cota superior trivial para esta cantidad es m, pero podemos acotar mejor. Si consideramos una posición arbitraria i en el texto y una ventana de tamaño $m-1$ consistente en los caracteres inmediatamente posteriores a i, sabemos que el carácter en i se verá una vez si es que $i \in \mathcal{H}$, más a lo sumo una vez por cada carácter P_m que haya en la ventana considerada. Más aún, cada uno de esos caracteres P_m deben estar en una posición que fue cabeza para que sea factible leer el carácter i-ésimo cada vez. Consideremos la cabeza más a la derecha que compara el patrón con el i-ésimo carácter, que llamaremos j, y sea A el primer nodo, en el camino que se hace en $\mathcal{G}$ al recorrer el texto, en el que se ve la posición j del texto (es fácil notar que de hecho la posición j es la cabeza principal del nodo A).

La Figura 3.16 da una idea de esta situación. Como A es un nodo, el texto que lo compone debe ser un sufijo del patrón (eventualmente con un error en su posición más a la izquierda), con las cabezas adecuadamente posicionadas. Llamemos t_P al

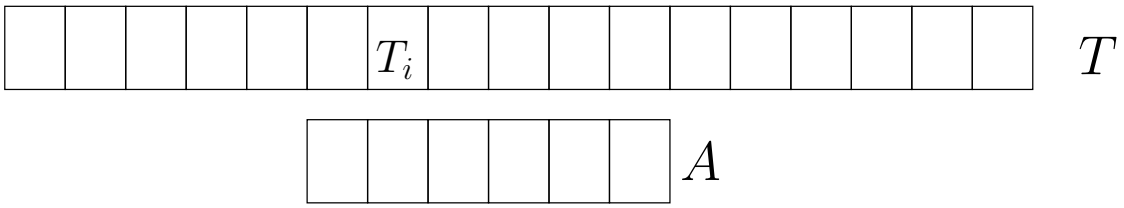

Figura 3.16: Cantidad máxima de comparaciones por posición.

número de caracteres en P que son iguales a P_m. Como el nodo A incluye a la posición i y todas las cabezas que ven la posición i, sigue que la cantidad de veces que se puede ver el carácter i-ésimo está acotada superiormente por las ocurrencias de P_m en P, eventualmente sumando 1 si la posición i fue una cabeza. Esto es, $\gamma_0 \leq t_P + 1$.

Más aún, si llamamos t'_P al número máximo de cabezas iguales a P_m que puede tener un nodo en $\mathcal{G}$, dado que todas las ocurrencias de P_m mencionadas en el párrafo anterior están incluidas en el nodo A, y necesariamente todas ellas fueron cabezas para haber comparado el patrón con el i-ésimo carácter, se tiene la proposición siguiente:

Proposición 3. Para todo patrón P y alfabeto Σ, se tiene

$$\gamma_0(P, \Sigma) \quad \leq \quad t'_P + 1.$$

Este resultado podría ser usado para conseguir cotas sobre $\mathbb{E}_{P \in_R \Sigma^m}[\gamma_0(P, \Sigma)]$, por ejemplo, o bien como una cota preliminar cada vez que se quiera calcular γ_0 sin necesidad de crear el grafo completo (basta con crear todos los nodos y encontrar aquel con mayor cantidad de cabezas P_m). El resultado aún puede ser mejorado, pero no es nuestra motivación y no exploramos más este tipo de resultados en este trabajo.

Capítulo 4

Modelo de perturbación

Este capítulo está dedicado a presentar resultados para el comportamiento general de BMH, principalmente con la ayuda del digrafo de desempeño presentado en el capítulo anterior. En lo inmediato, nos abocaremos a seguir los pasos que nos permitan hacer un análisis teniendo en mente el contexto suavizado. En particular, definiremos una perturbación para el texto y probaremos algunas propiedades generales de dicha perturbación, antes de analizar el desempeño del algoritmo bajo los efectos de ella.

4.1. Perturbación

Inspirados en la mayoría de los trabajos anteriores sobre el análisis para el caso promedio en algoritmos del problema de búsqueda de patrones, donde el modelo probabilístico se basa en una fuente sin memoria, y especialmente en trabajos anteriores que han propuesto modelos de perturbación sobre textos [3, 13, 7, 8, 25], trabajaremos en lo que sigue con un texto perturbado de la siguiente manera:

- Sea $\widetilde{T}$ un texto cualquiera, con $|\widetilde{T}| = n$.
- Cada posición de $\widetilde{T}$ es perturbada, independientemente, con probabilidad p, donde $p \in [0, 1]$ es el *parámetro de perturbación*.
- Las posiciones perturbadas son reemplazadas por caracteres elegidos uniformemente en Σ, de nuevo de forma independiente.

Como se puede ver, recuperamos el concepto de fuente probabilística sin memoria como las que hemos visto hasta ahora, en el sentido de que la perturbación en una posición no afecta posiciones futuras, y cada posición perturbada es reemplazada según variables independientes e idénticamente distribuidas. Un valor de $p = 0$ quiere decir que estamos en el estudio del peor caso, ya que el texto original y el

texto perturbado son, obviamente, iguales entre sí, mientras que $p = 1$ significa que volvemos al estudio del caso promedio, ya que todas las posiciones son perturbadas, lo que es equivalente a que todas las posiciones sean elegidas de manera uniforme en Σ. Si lo que nos motiva es hacer un estudio de la complejidad suavizada del algoritmo, es claro que el parámetro de perturbación representa el ruido al cual es sometida la entrada (solamente el texto, en nuestro análisis - no consideramos ruido sobre el patrón ni sobre el alfabeto). Llamamos T al texto perturbado resultante. La Figura 4.1 ilustra cómo funciona la perturbación.

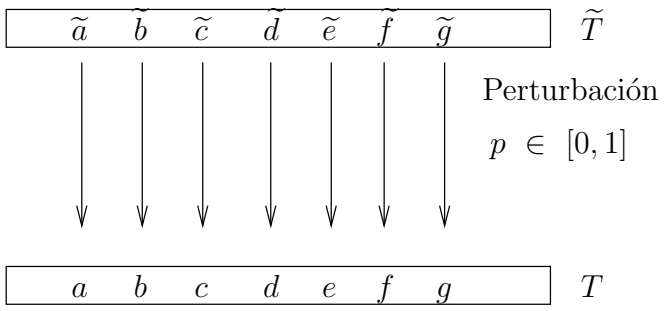

Figura 4.1: La perturbación en el texto.

No descartamos a priori la posibilidad de que un carácter perturbado sea reemplazado por sí mismo (lo que en efecto no produce ningún cambio), ni somos capaces, al ver T, de discernir cuáles caracteres provienen de una perturbación y cuáles no.

Es fácil notar que

- La probabilidad de que un carácter sea *alterado* (esto es, perturbado y reemplazado por un carácter distinto del original) es $p' = p(1 - 1/\sigma)$.
- La probabilidad de que un carácter sea perturbado y reemplazado por un $\alpha \in \Sigma$ específico es p/σ.

Notamos $T \leftarrow \widetilde{T}$ para referirnos al proceso de obtener T perturbando $\widetilde{T}$.

4.2. Subaditividad

En la sección anterior probamos un comportamiento subaditivo para el peor caso del algoritmo en textos de tamaño n. En lo que sigue veremos que dicho comportamiento se extiende al caso perturbado, considerando una secuencia adecuada que se ajuste a la aleatoriedad que hemos introducido.

Consideramos, para p fijo, la secuencia $\max_{|\widetilde{T}|=n} \mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)]$. Notemos que para $p=0$ esto es exactamente la secuencia $\max_{|T|=n} C(T)$ que consideramos en el capítulo anterior, mientras que para $p=1$ la secuencia es sencillamente $\mathbb{E}_{T\in_R\Sigma^n}[C(T)]$, ya que la perturbación elimina completamente cualquier dependencia que el texto orginal pueda tener en el texto resultante. Lo que sigue es probar el comportamiento subaditivo que nos interesa. El siguiente lema nos entrega ese resultado.

Lema 6. Para todo patrón P, y para todo $1 \leq r \leq n$, se cumple

$$\max_{|\widetilde{T}|=n} \mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)] \;\leq\; \max_{|\widetilde{T}|=r} \mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)] + \max_{|\widetilde{T}|=n-r+1} \mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)] + 2m^2$$

Demostración: La demostración es similar a la del Lema 1 para el caso $p=0$. Sean $\widetilde{A}$, $\widetilde{A}'$ y $\widetilde{A}''$ textos que maximizan la esperanza de comparaciones entre todos los textos posibles de tamaño n, r y $n-r+1$ respectivamente, donde, como antes, enumeramos $\widetilde{A}'' = \widetilde{A}''_{r..n}$ por comodidad. Llamaremos a los textos perturbados $A \leftarrow \widetilde{A}$, $A' \leftarrow \widetilde{A}'$ y $A'' \leftarrow \widetilde{A}''$ respectivamente.

Consideramos las cabezas r_1, r_2 y r' tal como en la demostración del Lema 1, sólo que en este caso todas ellas son variables aleatorias. Esto es:

- r_1 es la cabeza con mayor índice al leer el texto perturbado $A \leftarrow \widetilde{A}$, con $1 \leq r_1 \leq r-m+1$,
- r_2 es la cabeza con menor índice al leer el texto perturbado $A \leftarrow \widetilde{A}$, con $r+m-1 \leq r_2 \leq n$ y
- r' es la cabeza con mayor índice al leer el texto perturbado $A' \leftarrow \widetilde{A}'$ (eventualmente $r' = r$).

Notemos que, independientemente de cuál sea el texto A obtenido luego de la perturbación, se tiene que

$$C(A) \;\leq\; C(A_{1..r_1}) + C(A_{r_2-m+1..n}) + 2m^2,$$

de donde es fácil obtener que

$$\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A)] \;\leq\; \mathbb{E}_{r_1}[\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{1..r_1})|r_1]] + \mathbb{E}_{r_2}[\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{r_2-m+1..n})|r_2]] + 2m^2.$$

Por otro lado, dado que A' maximiza la esperanza de comparaciones entre textos de tamaño r, se tiene

$$\mathbb{E}_{r_1}[\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{1..r_1})|r_1]] \leq \mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{1..r})] \leq \mathbb{E}_{A'\leftarrow\widetilde{A}'}[C(A')].$$

De la misma forma podemos concluir que

$$\mathbb{E}_{r_2}[\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{r_2-m+1..n})|r_2]] \leq \mathbb{E}_{A\leftarrow\widetilde{A}}[C(A_{r..n})] \leq \mathbb{E}_{A''\leftarrow\widetilde{A}''}[C(A'')].$$

Así, obtenemos

$$\mathbb{E}_{A\leftarrow\widetilde{A}}[C(A)] \;\leq\; \mathbb{E}_{A'\leftarrow\widetilde{A}'}[C(A')] + \mathbb{E}_{A''\leftarrow\widetilde{A}''}[C(A'')] + 2m^2,$$

que es precisamente el resultado buscado. □

Usando el mismo razonamiento que para el Teorema 3.1.3 y ayudados por el lema recién demostrado, hemos probado el siguiente teorema:

Teorema 4.2.1. *Para cada alfabeto Σ y patrón P de tamaño m, y para cada $p \in [0,1]$ parámetro de perturbación, existe $\gamma_p = \gamma_p(P,\Sigma) \geq 1/m$ tal que*

$$\lim_{n\to\infty} \max_{|\widetilde{T}|=n} \frac{\mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)]}{n} \;=\; \inf_{n\geq 1} \max_{|\widetilde{T}|=n} \frac{\mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)]}{n} \;=\; \gamma_p$$

Con esto, ya sabemos que existen constantes que acotan el peor desempeño esperado del algoritmo. Nos gustaría obtener explícitamente el valor de dichas constantes. Esto, como veremos a continuación, tiene la complicación de que en la mayoría de los casos seremos incapaces de decir con certeza cuáles son los textos que maximizan la esperanza de comparaciones para un patrón y alfabeto dado.

Ya sabemos, de la sección anterior, construir un texto que da el peor desempeño asintótico para un patrón y alfabeto fijo, y por eso la primera intuición es la de considerar los peores textos como los textos de peor esperanza. Sin embargo, conjeturamos que los textos de peor desempeño **no son necesariamente** los textos que maximizan la esperanza de comparaciones en cada caso.

Lo que buscamos en el caso de texto perturbado no es necesariamente el texto que tendrá el peor desempeño, sino aquel que tendrá un mal desempeño que **será robusto**, en algún sentido, frente a la perturbación propuesta. A partir de este tipo de observaciones podemos notar que el problema de determinar las constantes γ_p y, si fuera posible, los textos que en cada caso maximizan la esperanza de comparaciones, están ligados de alguna forma al problema de determinar la complejidad suavizada del algoritmo. En lo que sigue nos interesa principalmente obtener la mayor cantidad posible de información sobre las constantes γ_p para cada patrón y alfabeto.

4.3. Resultados

Ya sabemos de la existencia de constantes que acotan el peor desempeño del algoritmo en el caso perturbado. Lo que ahora queda es determinar cuánto se gana con la pertubación. Más específicamente, qué tanto mejora el valor de las constantes

a medida que aumenta el valor de la perturbación, si es que de hecho lo hace.

Por un lado, sabemos del trabajo sobre el caso aleatorio de BMH (texto aleatorio para patrón fijo [7]) que para $p = 1$, el caso completamente perturbado, la cantidad esperada de comparaciones cumple

$$\frac{\mathbb{E}[C(T)]}{n} = \frac{1}{\mathbb{E}[K_\Sigma]}\left(\frac{\sigma}{\sigma-1} + \mathbb{E}\left[\frac{1}{\sigma^{K_\Sigma}}\right] + O\left(\frac{1}{\sigma^3}\right)\right),$$

donde al perturbar el texto por completo se elimina la dependencia en el texto original $\widetilde{T}$, como explicamos arriba. De esta forma, esperaríamos que a medida que el parámetro de perturbación se acerca a 1, el valor de la constante γ_p se acerca al valor del caso completamente aleatorio.

Aún con esto en mente, nuestro interés principal no recae en las perturbaciones grandes, sino en valores cercanos a 0 que puedan representar, como discutimos en un principio, ligeros errores en la entrada o en la lectura de ella (en este caso el texto que está sujeto a perturbación).

Así, nos concentramos en resultados que, o bien muestren que en casos prácticos la perturbación implica que la constante de peor caso mejora considerablemente, o bien resultados teóricos que garanticen una mejora de las constantes γ_p con respecto a γ_0, a medida que aumenta el valor del parámetro p.

Sabemos del análisis de peor caso que el más grande valor que puede tomar la constante $\gamma_0(P, \Sigma)$ es $\gamma_0 = m$. Luego, ciertamente nos interesará también encontrar condiciones para el patrón y el alfabeto que garanticen para ciertos valores del parámetro de perturbación un resultado del estilo $\gamma_p(P, \Sigma) = o(m)$. Este es el tipo de resultados que en lo inmediato vamos a intentar demostrar. Llamaremos $K_{\Sigma'}$ a la variable aleatoria que indica el shift de una cabeza perturbada que no es P_m. Así, $\mathbb{E}[K_{\Sigma'}] = \sum_{\alpha \neq P_m} K_\alpha/(\sigma - 1)$.

Teorema 4.3.1. *Sea un patrón P tal que existe un carácter $\not{P_m}$ con shift mayor o igual a 2. Entonces existe una constante $\rho = \rho(P, p, \Sigma) > 0$ tal que*

$$\gamma_p \leq \gamma_0 - \rho.$$

Demostración: Consideramos un texto perturbado arbitrario, que llamaremos T_p. Sea a el carácter $\not{P_m}$ con mayor shift, es decir tal que K_a es máximo entre todos los caracteres distintos de P_m. Nos interesa ver cuántas veces, en la representación en el digrafo del texto perturbado de tamaño n (dada por el Lema 3), se puede recorrer el bucle que une al nodo $\dot{a}$ consigo mismo. Para hacer esto necesitamos que el texto

tenga al carácter a en dos cabezas consecutivas y que el texto inmediatamente a la derecha de la segunda de estas cabezas, en una ventana de tamaño $m-1$, sea tal que ninguna comparación local llegue a volver a escanear ninguna de las dos cabezas $\dot{a}$ mencionadas anteriormente (en particular, no deben ver la de más a la derecha). Notemos que el largo del bucle es exactamente K_a. Sea ℓ_0 el largo que se recorre en el texto antes de usar el bucle $(\dot{a}, \dot{a})$ por primera vez. Luego, sea ℓ_1 el largo que se recorre hasta la segunda vez que en el camino que representa al texto perturbado, se recorre el bucle, y así hasta ℓ_e, donde e es tal que $\sum_{i=0}^{e} \ell_i + e \cdot K_a = n + o(n)$.

Es importante notar que los largos $\ell_1, \ldots, \ell_{e-1}$ se recorren entre nodos $\dot{a}$, luego son ciclos. El camino de largo ℓ_0 que se recorre inicialmente y el camino de largo ℓ_e del final no son necesariamente ciclos, y llamaremos a su costo respectivamente C_0 y C_e. Por otro lado el bucle tiene costo 1 y se recorre e veces. Con esto el costo $C(T_p)$ de este texto cumple

$$\begin{aligned} C(T_p) &\leq C_0 + \gamma_0(\sum_{i=1}^{e-1} \ell_i) + C_e + e \\ &\leq C_0 + \gamma_0(\sum_{i=0}^{e} \ell_i + e \cdot K_a) + C_e - e(\gamma_0 \cdot K_a - 1). \end{aligned}$$

Ahora bien, veamos la probabilidad en cada cabeza de recorrer el bucle $\dot{a} - \dot{a}$. Para que esto ocurra, como mencionamos arriba, es necesario encontrar dos cabezas $\dot{a}$ consecutivas seguidas de una ventana de texto tal que ninguna comparación local vuelva a escanear ninguna de dichas cabezas $\dot{a}$. La probabilidad de encontrar dos cabezas $\dot{a}$ consecutivas es al menos $(p/\sigma)^2$. Por otro lado, después de la segunda cabeza $\dot{a}$ avanzamos K_a, luego, para no volver a escanear la cabeza $\dot{a}$ de más a la derecha en la comparación local inmediatamente posterior a la segunda cabeza $\dot{a}$, es necesario que estos K_a caracteres no sean un sufijo del patrón. La probabilidad de que esto ocurra es al menos $1 - (1-p')^{K_a}$. Luego de dicha comparación local, supongamos que encontramos solamente cabezas perturbadas con un carácter $\not{P}_m$, ello implica que los shifts en cada una de estas cabezas son entre sí variables aleatorias independientes e idénticamente distribuidas. Para $t > 0$, llamemos α_i, $i = 1, \ldots, t$ a los caracteres que se encuentran en las t primeras de estas cabezas perturbadas (todos esos caracteres son resultado de una perturbación). Se tiene que $\mathbb{E}[\sum_{i=1}^{t} K_{\alpha_i}] = t\mathbb{E}[K_{\alpha_i}] = t\mathbb{E}[K_{\Sigma'}]$. Esto todavía se tiene si escogemos t como la variable aleatoria que representa al índice más pequeño tal que $\sum_{i=1}^{t} K_{\alpha_i} \geq m - K_a$. En efecto,

$$\mathbb{E}\left[\sum_{i=1}^{t} K_{\alpha_i}\right] = \mathbb{E}_t\left[\mathbb{E}\left[\sum_{i=1}^{t} K_{\alpha_i} \mid t\right]\right] = \mathbb{E}_t[t\ \mathbb{E}\left[K_{\Sigma'}\right]|t] = \mathbb{E}[t]\mathbb{E}\left[K_{\Sigma'}\right].$$

Cada una de esas cabezas origina una sola comparación local (en particular no se seguirá comparando hacia la izquierda, luego es imposible que se vuelva a escanear

hasta la posición en la que está la cabeza $\dot{a}$ de más a la derecha). De la definición de t es directo que $\sum_{i=1}^{t-1} K_{\alpha_i} < m - K_a$, y luego, tomando esperanza, sigue

$$\mathbb{E}[t] \;<\; \left\lfloor \frac{m - K_a}{\mathbb{E}[K_{\Sigma'}]} \right\rfloor + 1.$$

Llamemos $\nu = 2\,\lfloor (m - K_a)/\mathbb{E}[K_{\Sigma'}] \rfloor + 2$, por la desigualdad de Markov se tiene

$$\mathbb{P}[t \geq \nu] \leq \frac{1}{2},$$

Consideremos la cantidad $(p/\sigma)^2(1-(1-p')^{K_a})\mathbb{E}_t[(p')^t]$. De nuestro análisis esta es una cota inferior a la probabilidad de que se encuentren dos cabezas consecutivas $\dot{a}$ y luego una ventana de texto tal que el algoritmo no vuelva a comparar el patrón con ninguna de las dos cabezas $\dot{a}$, esto es, que se recorra el bucle $(\dot{a}, \dot{a})$.

Para manejar $\mathbb{E}_t[(p')^t]$, separamos y condicionamos en los eventos $\{t \geq \nu\}$ y $\{t < \nu\}$. Obtenemos

$$\mathbb{E}_t[(p')^t] = \mathbb{E}[(p')^t | t \geq \nu]\mathbb{P}[t \geq \nu] + \mathbb{E}[(p')^t | t < \nu]\mathbb{P}[t < \nu] \geq 0 + (p')^\nu \mathbb{P}[t < \nu] \geq 1/2\,(p')^\nu,$$

de donde

$$\pi \;=\; \frac{1}{2}\left(\frac{p}{\sigma}\right)^2 (1-(1-p')^{K_a})(p')^\nu$$

es una cota inferior a la probabilidad, en cada cabeza del texto T_p, de recorrer el bucle $(\dot{a}, \dot{a})$, y, dado que hay al menos n/m cabezas en cualquier texto, sigue que

$$e \;\geq\; \frac{n}{m}\pi.$$

Con todo esto, podemos obtener

$$\frac{C(T_p)}{n} \;\leq\; \frac{C_0 + C_e}{n} + \gamma_0 \frac{(n + o(n))}{n} - \frac{\pi}{m}(\gamma_0 \cdot K_a - 1).$$

Notemos que $C_0 + C_e = O(1)$, pues está claro que el largo de un camino que no pasa por el bucle $(\dot{a}, \dot{a})$ no depende del largo del texto, sino de propiedades del digrafo, que a su vez no depende de n. Con esto, haciendo $n \to \infty$ se obtiene el resultado, para $\rho = \pi(K_a - 1)\gamma_0/m$. $\square$

Vale la pena hacer una observación; consideremos de nuevo el peor caso posible, esto es $P = ba^{m-1},\ \Sigma = \{a, b\}$. Como ya sabemos, el peor ciclo en este caso en $\mathcal{G}(P, \Sigma)$ es un bucle de largo 1 y costo m. El carácter b tiene asociado un shift igual a $m-1$, con lo que $\mathbb{E}[K_{\Sigma'}] = m-1$, $\gamma_0 = m$ y $\pi = 1/2\ (p/2)^2 \cdot (1-(1-p/2)^{m-1})(p/2)^2$.

Podemos asegurar en virtud del teorema anterior que $\gamma_p(P,\Sigma) \leq m - (p/2)^4 \cdot (1 - (1-p/2)^{m-1})(m-2)$.

Para $p = 1$ la cota superior es $m - (1/2)^4 \cdot (1 - (1/2)^{m+1})(m-2)$, lo que aún está lejos del resultado del caso promedio para patrón fijo asegurado por Régnier y Baeza-Yates para $\mathbb{E}[K_\Sigma] = m/2$ y $\sigma = 2$, lo que indica que sin duda existen maneras de mejorar este resultado. Aún así, el resultado ofrece una noción preliminar razonable de cómo disminuye la dificultad del problema una vez expuesta la entrada a la perturbación.

Por otro lado, la condición que pide este teorema, a saber, que exista un carácter $\not{P}_m$ con shift mayor o igual a 2, se cumple en casi todos los casos. En particular, si $\sigma \geq 3$, necesariamente debe haber un carácter $\not{P}_m$ con shift mayor o igual a 2. Incluso si $\sigma = 2$, cualquier patrón en el que $P_{m-1} = P_m$ cumple la misma condición. El único caso (la única familia de casos, en rigor) en el que esta condición no se cumple es en patrones del estilo $P = \Sigma^* ba$ para $\Sigma = \{a, b\}$.

4.3.1. Cantidad de cabezas perturbadas

Vamos a considerar ahora un resultado de naturaleza diferente. En primer lugar, vamos a discutir algunos resultados relativos a la cantidad de cabezas en un texto perturbado. Llamaremos $\widetilde{\mathcal{H}} = \{\widetilde{i}_1, \ldots, \widetilde{i}_{|\widetilde{\mathcal{H}}|}\}$ al conjunto de cabezas original, es decir, antes de la perturbación. Es fácil notar que $1 \leq \widetilde{i}_{j+1} - \widetilde{i}_j \leq m$ para $j = 1, \ldots, |\widetilde{\mathcal{H}}| - 1$. Más aún, se tiene que $\widetilde{i}_{j+1} - \widetilde{i}_j = K_{\alpha_{\widetilde{r}}}$, donde $\alpha_{\widetilde{r}}$ es el carácter de Σ que se encuentra en la posición $\widetilde{i}_j$.

Luego de la perturbación, el conjunto de cabezas cambia según hayan sido perturbadas las cabezas, pero la relación anterior se mantiene: Si el conjunto de cabezas ahora es $\mathcal{H} = \{i_1, \ldots, i_{|\mathcal{H}|}\}$, se tiene que $i_{j+1} - i_j = K_{\alpha_r}$, donde α_r es el carácter (eventualmente perturbado) que se encuentra en la posición i_j-ésima del texto.

Lema 7. Consideremos una ejecución de BMH en texto perturbado. Para $k > 0$ sea $\mathcal{H}_k$ el conjunto de las primeras k cabezas, y sea $\mathcal{F}_k$ el conjunto de cabezas, dentro de las primeras k, que son perturbadas. Entonces $|\mathcal{F}_k| \sim \text{Binomial}(k, p)$

Demostración: Definamos el conjunto $\mathbb{H}$ de *Cabezas factibles* para el patrón P, como la colección de todas las tuplas de posiciones en el texto que pueden ser el el conjunto de cabezas. Formalmente, llamando $K_{\max}$ al mayor shift posible entre todos los caracteres en Σ:

$$\mathbb{H} = \{(i_1, \ldots, i_h) | i_{j+1} - i_j = K_\alpha, \text{ algún } \alpha \in \Sigma, j = 1, \ldots, h-1; n - K_{\max} \leq i_h \leq n\}.$$

Sea $\mathcal{H}$ el conjunto de cabezas luego de la perturbación. Llamaremos $F_k = |\mathcal{F}_k|$, que es la variable aleatoria a la que queremos encontrar una distribución. Sea $\mathcal{H}' \in \mathbb{H}$, calcularemos primero $\mathbb{P}[F_k = j \mid \mathcal{H}_k = \mathcal{H}'_k]$, donde $\mathcal{H}'_k$ son los primeros k elementos de $\mathcal{H}'$. Ahora bien, dado que, en el evento considerado conocemos exactamente el conjunto de cabezas luego de la perturbación, encontrar el número de cabezas perturbadas corresponde a calcular la suma de los eventos en que cada una de las k cabezas fue perturbada, lo que ocurre con probabilidad p e independientemente entre cada posición. Con esto, condicional al evento considerado, se tiene que F_k es una binomial de parámetros p y k, es decir

$$\mathbb{P}[F_k = j \mid \mathcal{H}_k = \mathcal{H}'_k] \;=\; \binom{k}{j} p^j (1-p)^{k-j}.$$

Luego, si llamamos $\mathbb{H}_k$ al conjunto de tuplas de tamaño k que se obtienen de los primeros k elementos de cada una de las tuplas en $\mathbb{H}$, se tiene

$$\begin{aligned}
\mathbb{P}[F_k = j] &= \sum_{\mathcal{H}'_k \in \mathbb{H}_k} \mathbb{P}[F_k = j \mid \mathcal{H}_k = \mathcal{H}'_k]\mathbb{P}[\mathcal{H}_k = \mathcal{H}'_k] \\
&= \sum_{\mathcal{H}'_k \in \mathbb{H}_k} \binom{k}{j} p^j (1-p)^{k-j} \mathbb{P}[\mathcal{H}_k = \mathcal{H}'_k] \\
&= \binom{k}{j} p^j (1-p)^{k-j} \sum_{\mathcal{H}'_k \in \mathbb{H}_k} \mathbb{P}[\mathcal{H}_k = \mathcal{H}'_k] \\
&= \binom{k}{j} p^j (1-p)^{k-j}.
\end{aligned}$$

Tenemos entonces que F_k se distribuye como una binomial de parámetros p y k, como queríamos probar. □

Esto ya da una idea de cuántas cabezas perturbadas deberíamos encontrar en una ejecución de BMH en texto perturbado. Notemos que, dado que no sabemos exactamente cómo se componen los textos que maximizan la esperanza de comparaciones en cada caso, no tenemos control sobre cuáles son los caracteres que existirán en el texto perturbado al ejecutar BMH. En particular, no sabemos qué carácter encontraremos en cada cabeza, y mucho menos cómo se comportará la secuencia de shifts. Así, la utilidad de un resultado como el Lema 7 es el hecho de que nos otorga algún control sobre los shifts, ya que al menos sabemos que, en una cabeza perturbada, el shift es la variable aletoria K_Σ, y sabemos que la cantidad de cabezas que tienen un shift dado por esa variable aleatoria se distribuye como una binomial. Este resultado nos ayudará más adelante a probar un resultado general sobre la cantidad esperada de cabezas que se pueden ver en un texto perturbado.

4.3.2. Análisis de la cantidad de cabezas

Como ya sabemos, en el peor caso pueden haber hasta $n - m$ cabezas en el texto. Luego esperaríamos, en primer lugar, que la perturbación tuviera algún efecto mitigante en la cantidad de cabezas y, en segundo lugar, un efecto en la cantidad de comparaciones locales que se hace en cada cabeza. En lo que sigue analizaremos, usando el resultado reciente sobre la distribución de la cantidad de cabezas perturbadas, lo que ocurre con la cantidad total de cabezas en el texto perturbado bajo diversas condiciones sobre el patrón, alfabeto y parámetro de perturbación.

En el desarrollo usaremos la siguiente cota del tipo Chernoff-Hoeffding vista en [14]:

Lema 8. [Lema 9, [14]] Sean $X_1, \ldots, X_\nu$ variables aleatorias independientes e idénticamente distribuidas con soporte en $\{0, \ldots, l\}$ de media μ. Si $X = X_1 + \ldots + X_\nu$, entonces para $a < 0$ se tiene:

$$\mathbb{P}[X - \mathbb{E}X \leq a] \quad \leq \quad \exp\left(-\frac{a^2}{2l\nu\mu}\right).$$

Recordemos que llamamos $H = |\mathcal{H}|$ y sea $k > 0$ a elegir adecuadamente. Se tiene

$$\mathbb{E}[H] = \mathbb{E}[H\mathbb{1}_{\{H \leq k\}}] + \mathbb{E}[H\mathbb{1}_{\{H > k\}}] \leq k + (n - m)\mathbb{E}[\mathbb{1}_{\{H > k\}}]. \tag{4.1}$$

Y ahora buscaremos una cota adecuada para $\mathbb{E}[\mathbb{1}_{\{H>k\}}] = \mathbb{P}[H > k]$. Para que haya más de k cabezas es necesario que la suma de los primeros k shifts sea menor o igual a $n - m$. Esto es

$$\sum_{i \in \mathcal{H}_k} K_{T_i} \quad \leq \quad n - m.$$

Con esto, analizaremos la probabilidad del evento $\sum_{i \in \mathcal{H}_k} K_{T_i} \leq n - m$. Como no sabemos qué carácter está en una posición determinada del texto no perturbado, sólo tenemos cierto control sobre lo que pasa en las posiciones perturbadas, ya que al menos en ese caso tenemos una distribución de probabilidad para los caracteres que pueden ocupar dichas posiciones. Más específicamente, nos preocupan las cabezas que son perturbadas. Así separamos y condicionamos según los eventos $\{F_k \leq (1 - \xi_1)pk\}$ y $\{F_k > (1 - \xi_1)pk\}$, para $\xi_1 > 0$ adecuadamente escogido. Se tiene:

$$\begin{aligned}\mathbb{P}\left[\sum_{i \in \mathcal{H}_k} K_{T_i} \leq n - m\right] &= \mathbb{P}\left[\sum_{i \in \mathcal{H}_k} K_{T_i} \leq n - m \mid F_k \leq (1 - \xi_1)pk\right] \mathbb{P}\left[F_k \leq (1 - \xi_1)pk\right] \\ &\quad + \mathbb{P}\left[\sum_{i \in \mathcal{H}_k} K_{T_i} \leq n - m, F_k > (1 - \xi_1)pk\right]. \end{aligned} \tag{4.2}$$

Usando que F_k se distribuye como una binomial de parámetros p y k, sigue la clásica cota para desviación de binomiales [28]:

$$\mathbb{P}[F_k \leq (1-\xi_1)pk] \quad \leq \quad \exp\left(-\frac{\xi_1^2 pk}{2}\right).$$

Llamemos $\epsilon_1 = \exp(-\xi_1^2 pk/2)$. Por (4.2) sigue que

$$\mathbb{P}\left[\sum_{i\in\mathcal{H}_k} K_{T_i} \leq n-m\right] \quad \leq \quad \epsilon_1 + \mathbb{P}\left[\sum_{i\in\mathcal{H}_k} K_{T_i} \leq n-m, F_k > (1-\xi_1)pk\right].$$

Nos concentraremos en lo que sigue en el evento

$$\{\sum_{i\in\mathcal{H}_k} K_{T_i} \leq n-m\} \quad \bigcap \quad \{F_k > (1-\xi_1)pk\}.$$

Lo único que sabemos en este caso es que al menos $(1-\xi_1)pk$ cabezas fueron perturbadas. En el peor de los casos, cada vez que nos encontremos con una cabeza que no fue perturbada, el shift será igual a uno. Por otro lado, cada vez que nos encontremos con una cabeza perturbada, el shift es una variable aleatoria de esperanza $\mathbb{E}[K_\Sigma]$. Luego de las k primeras cabezas hay $k - F_k$ que asumimos generan un shift igual a uno. De esta forma, denotando $X = \sum_{i\in\mathcal{F}_k}(K_{T_i}-1)$:

$$\begin{aligned}
\mathbb{P}\left[\sum_{i\in\mathcal{H}_k} K_{T_i} \leq n-m, F_k > (1-\xi_1)pk\right] \quad &\leq \quad \mathbb{P}\left[k - F_k + \sum_{i\in\mathcal{F}_k} K_{T_i} \leq n-m, F_k > (1-\xi_1)pk\right] \\
&= \quad \mathbb{P}\left[\sum_{i\in\mathcal{F}_k} (K_{T_i}-1) \leq n-m-k, F_k > (1-\xi_1)pk\right] \\
&= \quad \mathbb{P}\left[\sum_{i\in\mathcal{F}_k} (K_{T_i}-1) - \mathbb{E}X \leq n-m-k-\mathbb{E}X, F_k > (1-\xi_1)pk\right].
\end{aligned}$$

Ahora bien, sabemos que

$$\mathbb{E}[\sum_{i\in\mathcal{F}_k}(K_{T_i}-1) \mid F_k] \quad = \quad F_k(\mathbb{E}[K_\sigma]-1).$$

Donde F_k es una variable aleatoria cuya distribución conocemos. Más aún, en el evento que estamos considerando se tiene $F_k > (1-\xi_1)pk$, y entonces $\mathbb{E}[X] = \mathbb{E}_{F_k}[\mathbb{E}[X \mid F_k]] > (1-\xi_1)pk(\mathbb{E}[K_\Sigma - 1])$. Teniendo esto en cuenta, si llamamos $a = n-m-k-(1-\xi_1)pk(\mathbb{E}[K_\sigma]-1)$, resulta

$$\mathbb{P}[X - \mathbb{E}\,X \leq n-m-k-\mathbb{E}\,X, F_k > (1-\xi_1)pk] \leq \mathbb{P}[X - \mathbb{E}\,X \leq a, F_k > (1-\xi_1)pk]. \tag{4.3}$$

Usando el Lema 8, sigue que, si hacemos k tal que $a < 0$, entonces podemos acotar el término de la derecha en (4.3) por

$$\exp\left(-\frac{[n-m-k-(1-\xi_1)pk(\mathbb{E}[K_\Sigma]-1)]^2}{2(m-1)(1-\xi_1)pk(\mathbb{E}[K_\Sigma]-1)}\right).$$

Para que $a < 0$ basta con tomar

$$k > \frac{n-m}{1+(\mathbb{E}[K_\Sigma]-1)(1-\xi_1)p}. \tag{4.4}$$

Digamos más bien, k tal que para $0 < \xi_2 < 1$ adecuadamente escogido, se cumpla

$$k(1-\xi_2) = \frac{n-m}{1+(\mathbb{E}[K_\Sigma]-1)(1-\xi_1)p}.$$

Con todo este análisis, si llamamos

$$\epsilon_2 = \exp\left(-\frac{[n-m-k-(1-\xi_1)pk(\mathbb{E}[K_\Sigma]-1)]^2}{2(m-1)(1-\xi_1)pk(\mathbb{E}[K_\Sigma]-1)}\right),$$

se tiene, añadiendo a lo que teníamos antes en (4.1),

$$\mathbb{E}[H] \leq k + (n-m)(\epsilon_1+\epsilon_2).$$

Despreciando los términos exponenciales, lo único que sobrevive es k, y luego una cota asintótica para la esperanza de las comparaciones es k, escogido según (4.5). Hemos probado el siguiente resultado:

Teorema 4.3.2. *Para cada m, asintóticamente en n, la cantidad esperada de cabezas en una ejecución de BMH en T, P, Σ, con $T \leftarrow \widetilde{T}$, cumple, para $\xi_1, \xi_2 > 0$ elegidos adecuadamente,*

$$\mathbb{E}[H] \leq \frac{1}{1-\xi_2} \cdot \frac{n-m}{1+(\mathbb{E}[K_\Sigma]-1)(1-\xi_1)p}.$$

Con esto ya tenemos una cota asintótica para la cantidad de cabezas. Incluimos además las Figuras 4.2 y 4.3 que muestran valores asintóticos de ϵ_1 y ϵ_2, para justificar con algunos datos empíricos el hecho de que los despreciamos, y la cantidad de cabezas, comparada con la cota para la cantidad esperada, para diversas perturbaciones.

Todo esto se hace en el marco del peor caso: $P = ba^{m-1}$, $\widetilde{T} = a^n$, de tal forma que la suposición que hacemos al asumir que las cabezas que no son perturbadas dan el shift mínimo posible resulta correcta. En rigor, para la cantidad de cabezas, esto se podría haber hecho con cualquier patrón en un texto original que consista sólamente del carácter P_{m-1} repetido, pero la ventaja del peor caso es que además conocemos exactamente tanto el peor texto como el texto que maximiza la esperanza de comparaciones bajo perturbación (y resultan ser el mismo).

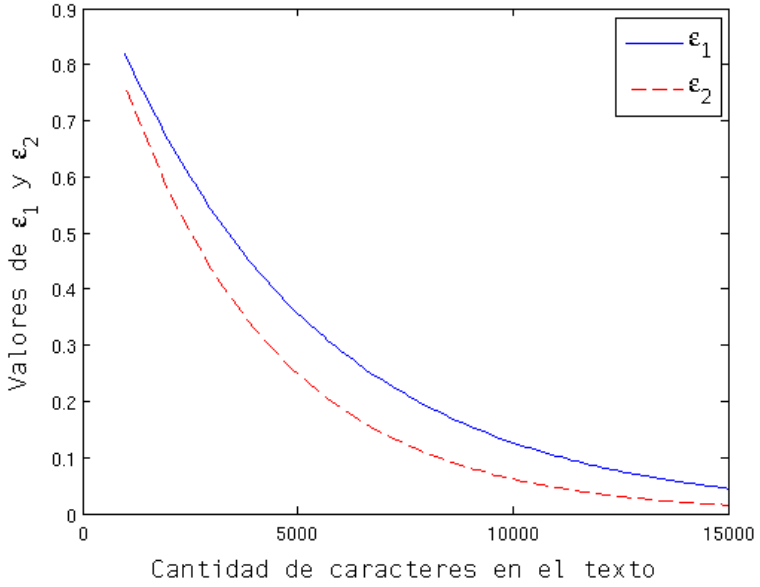

Figura 4.2: ϵ_1 y ϵ_2 para $p = 0{,}1$, $\xi_1 = 0{,}1$, $\xi_2 = 0{,}3$, $m = 40$

La Figura 4.4 muestra una comparación entre el número esperado y obtenido de cabezas. Como la comparación se hace obteniendo el número de cabezas para el peor caso Patrón-Texto (lo que produce el máximo número posible de cabezas), esperaríamos que la razón entre ambos valores sea cercano a 1, y es precisamente lo que se obtiene, para los parámetros indicados en la figura.

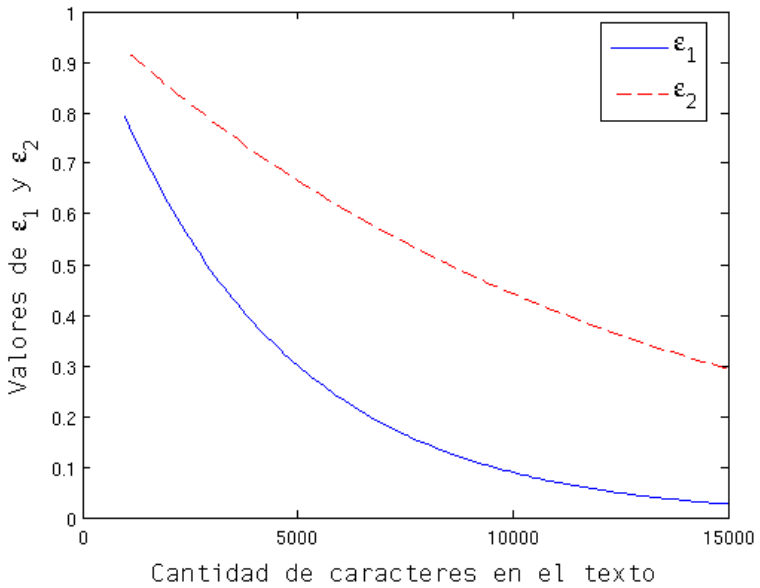

Figura 4.3: ϵ_1 y ϵ_2 para $p = 0{,}2$, $\xi_1 = 0{,}1$, $\xi_2 = 0{,}3$, $m = 40$

El siguiente resultado está basado en el anterior y nos da una cota asintótica en términos de m para la constante γ_p.

Teorema 4.3.3. *Sea P un patrón con $\mathbb{E}[K_\Sigma] = O(m^{1-\varepsilon})$. Entonces para $p = m^{2\varepsilon}/m$ se tiene*

$$\gamma_p(P, \Sigma) \leq m^{1-\varepsilon}.$$

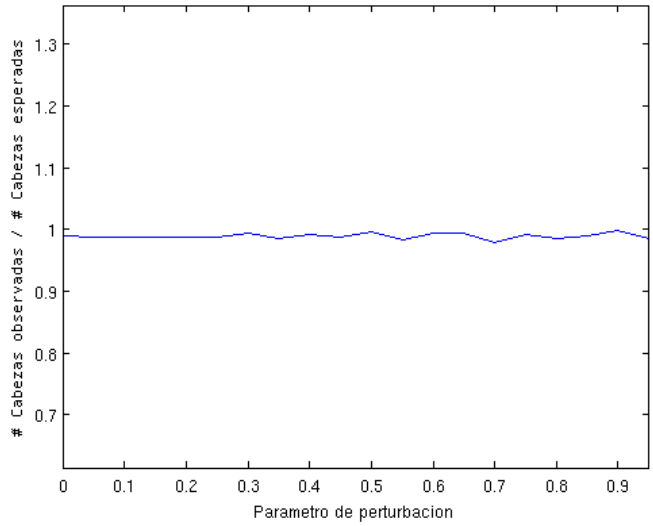

Figura 4.4: Comparación entre la cantidad observada y esperada de cabezas según el parámetro p, $n = 158158$, $m = 20$, $\xi_1 = \xi_2 = 0{,}1$.

Demostración: Del resultado anterior, salvo las constantes ξ_1 y ξ_2 sale que

$$\begin{aligned} \mathbb{E}[H] &\leq O\left(\frac{n-m}{1+(\mathbb{E}[K_\Sigma]-1)p}\right) \\ &\leq O\left(\frac{n-m}{p(\mathbb{E}[K_\Sigma]-1)}\right). \end{aligned}$$

Para $\mathbb{E}[K_\Sigma] = O(m^{1-\varepsilon})$ y $p = m^{2\varepsilon-1}$, esto dice que

$$E[H] \leq O\left(\frac{n-m}{m^\varepsilon}\right).$$

Incluso asumiendo el máximo de comparaciones locales, esto dice que para cualquier $\widetilde{T}$ se tiene $\mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)] \leq m\mathbb{E}_{T\leftarrow\widetilde{T}}[H]$, y entonces

$$\mathbb{E}_{T\leftarrow\widetilde{T}}[C(T)] \leq O(m^{1-\varepsilon}(n-m)),$$

de donde se concluye el resultado deseado para γ_p. □

De este último teorema, en particular, obtenemos una condición suficiente para asegurar que $\gamma_p(P, \Sigma) = o(m)$.

Capítulo 5

Conclusiones y trabajo futuro

Si bien el objetivo primordial de un trabajo de este estilo es demostrar que el espacio de entradas del algoritmo es tal que los picos de complejidad son suficientemente pronunciados (en el sentido de que un leve desplazamiento en el espacio de entradas provoca un gran descenso en la complejidad del algoritmo) y están bastante apartados, nuestro aporte básicamente consiste en demostrar que la familia de constantes que acotan el peor desempeño perturbado del algoritmo son razonablemente menores que las constantes para el peor caso.

En ese sentido, nuestros resultados principales en cuanto al desempeño perturbado son demostrar que la esperanza de comparaciones está asintóticamente acotada por $\gamma_p \cdot n$, donde existe un gap entre los valores de γ_p y γ_0 que sólo depende de características del patrón y del alfabeto, además del parámetro de perturbación. Como esperábamos, para valores de este último parámetro cercanos a 0 se recupera el estudio del peor caso (ya que el gap entre γ_p y γ_0 se anula). Pero, como mencionamos anteriormente, al hacer $p = 1$ no necesariamente se recupera el estudio del caso promedio, lo que indica que ciertamente el análisis provisto es susceptible a ser mejorado. Si bien no se recuperan resultados como los del caso promedio, tenemos una indicación de que el desempeño en texto perturbado es sustancialmente mejor que en el peor texto. Es decir que la perturbación de hecho afecta positivamente al algoritmo en maneras que podemos cuantificar. Otro de nuestros resultados importantes da una condición suficiente sobre P, p y Σ para asegurar que $\gamma_p(P, \Sigma) = o(m)$. Esto merece un comentario posterior: hemos probado que asintóticamente la cantidad esperada de comparaciones en el caso perturbado está acotada por una constante que es despreciable con respecto a m cuando $m \to \infty$, incluso para valores pequeños de perturbación (ya que $p = m^{2\varepsilon}/m \to 0$ cuando $m \to \infty$).

Ahora bien, aunque no hemos encontrado un resultado de complejidad suavizada en ninguno de los sentidos clásicos para este campo, vale la pena hacer notar que sí encontramos y desarrollamos una serie de herramientas (principalmente el digrafo

de desempeño) que permiten demostrar que existe una familia de constantes que acotan el desempeño de peor caso y luego encontramos y demostramos cotas preliminares para dichas constantes, siendo capaces, para valores de m y Σ razonables, de calcularlas explícitamente.

Quedan varias preguntas que valdría la pena responder en el futuro. Algunas de las más importantes pueden ser:

- ¿Qué tanto se puede mejorar el análisis hecho de manera de obtener mejores cotas para γ_p, especialmente comparados con γ_0? El análisis no es trivial y, por ello, en muchas partes nos vimos obligados a asumir condiciones de peor caso (como m comparaciones locales o shifts iguales a 1 repetidamente) que dan lugar a resultados tal vez menos óptimos que los que se puede encontrar. En particular, ya hemos conjeturado que el peor ciclo en el digrafo de desempeño es un bucle. Si se lograra probar un resultado de ese estilo, el análisis de los peores ciclos se podría hacer más sencillo y dar lugar a mejores resultados.

- Los teoremas más importantes sobre el desempeño en el caso perturbado tienen algunas excepciones sobre patrones y alfabetos. Vale la pena preguntarse qué tan restrictivas son estas excepciones, y si generan una familia de patrones y/o alfabetos que pueda ser parametrizada de alguna manera para hacer un análisis separado.

- ¿Hasta qué punto se puede heredar o rehacer un análisis como este para algoritmos distintos? El algoritmo BMH fue escogido por su simplicidad en cuanto a preprocesamiento y buen desempeño en la práctica, pero si se pudieran adaptar argumentos como estos para otros algoritmos de búsqueda de patrones se podría tener un punto de comparación nuevo entre todos estos algoritmos. Particularmente, podría ofrecer nueva evidencia de que el algoritmo BMH, a pesar de ser más simple, tiene mejor desempeño en la gran mayoría de los casos. Por otro lado, nos podemos hacer la pregunta ¿Existe una versión o variante de BMH cuyo análisis suavizado sea más simple o tenga complejidad suavizada demostrablemente mejor a lo que hemos encontrado en este trabajo?

- Consideremos un patrón $\widetilde{P}$, que luego sometemos a una perturbación como la que vimos para el texto, generando un patrón P (esto es, elegimos $P \leftarrow \widetilde{P}$). ¿Qué podemos decir de $\gamma_p(P, \Sigma)$ con respecto de $\gamma_0(\widetilde{P}, \Sigma)$? Las distribuciones de los valores de γ_0 en las figuras del capítulo 4 sugieren que existe un gap entre los peores valores de $\gamma_0(P, \Sigma)$ y los demás. Es decir, que hay solo unos pocos patrones que presentan el peor comportamiento posible, y la gran mayoría de los patrones provoca un mucho mejor desempeño. Un resultado en este marco estaría mucho más cerca de los objetivos usuales del análisis suavizado y daría evidencia más fuerte de que BMH tiene baja complejidad suavizada.

Bibliografía

[1] G. Navarro, M. Raffinot. Flexible pattern matching in strings. *Cambridge University Press*, 2002.

[2] J. H. Morris Jr., V. R. Pratt. A linear pattern matching algorithm. Report 40, University of California, Berkeley, 1970.

[3] R. S. Boyer, J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, vol 20-10, pp 762-772, 1977.

[4] R. N. Horspool. Practical fast searching in strings. *Software - Practice and experience*, vol 10, pp 501-506, 1980.

[5] R. Cole. Tight bounds on the complexity of the Boyer-Moore string matching algorithm. *Proceedings of the second annual ACM-SIAM Symposium on Discrete Algorithms*, pp 224-233, 1991.

[6] V. Klee, G. J. Minty. How good is the simplex algorithm?. *O. Shisha (ed.) Inequalities III, Academic Press, New York* , pp 159-179, 1972.

[7] R. Baeza-Yates, M. Régnier. Average running time of the Boyer-Moore-Horspool algorithm. *Theoretical Computer Science*, vol 92, pp 19-31, 1992.

[8] W. Szpankowski. Average case analysis of algorithms on sequences. *Wiley Series in Discrete Mathematics and Optimization*, 2001.

[9] M. Fekete. Uber die Verteilung der Wurzeln bei gewissen algebraischen Gleichungen mit ganzzahligen Koeffizienten. *Mathematische Zeitschrift*, vol 17, pp 228-249, 1923.

[10] J. F. C. Kingman. Subadditive processes. *Springer-Verlag*, 1976.

[11] T. Ligget. Interacting particle systems. *Springer-Verlag*, 1985.

[12] Y. Derriennic. Un théorème ergodique presque sous additif. *Ann. Probab.*, vol 11, pp 669-677, 1983.

[13] D.E. Knuth, J. H. Morris Jr, V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, vol 6, pp 323-350, 1977.

[14] M. Kiwi, M. Loebl. Largest planar matching in random bipartite graphs. *Random Structures & Algorithms*, vol 21-2, pp 162-181, 2002.

[15] R. Baeza-Yates. String search algorithms revisited. *Lecture Notes In Computer Science*, vol 382, pp 75-96, 1989.

[16] M. Régnier. Knuth-Morris-Pratt algorithm: An analysis. *Proceedings on the 14th Symposium on Mathematical Foundations of Computer Science*, vol 379, Lecture Notes in Computer Science, pp 431-444, Springer-Verlag, 1989.

[17] H. M. Mahmoud, R. T. Smythe, M. Régnier. Analysis of Boyer-Moore-Horspool string-matching heuristic. *Random structures and algorithms*, vol 10, issue 1-2, pp 169-186, 1997.

[18] D. Spielman, S-H Teng. Smoothed analysis: Motivation and discrete models. *Lecture Notes In Computer Science*, vol 2748, pp 256-270, 2003.

[19] P. Bürgisser, F. Cucker, M. Lotz. General formulas for the smoothed analysis of condition numbers. *Comptes Rendus Mathematique*, vol 343, issue 2, pp 145-150, 2006.

[20] H. Röglin, B. Vöcking. Smoothed analysis of integer programming. *Mathematical programming*, vol 110, n 1, pp 21-56, 2007.

[21] A. Sankar. Smoothed analysis of Gaussian elimination. *Tesis PhD*, Massachusets Institute of Technology, 2004.

[22] B. Manthey, R. Reischuk. Smoothed analysis of the height of binary trees. *Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAACS)*, pp 483-492, 2005.

[23] D. Spielman, S-H Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, vol 51, pp 385-463, 2004.

[24] A. Andoni, R. Krauthgamer. The smoothed complexity of edit distance. *Lecture Notes In Computer Science*, vol 5125, pp 357-369, 2008.

[25] T-H Tsai. Average case analysis of the Boyer-Moore algorithm. *Random structures and algorithms*, vol 28, issue 4, pp 481-498, 2006.

[26] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, vol 23, pp 309-311, 1978.

[27] A. Dasdan, R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions*, vol 17, issue 10, pp 889-899, 1998.

[28] T. Cormen, C. Leiserson, R. Rivest. Introduction to Algorithms. *The MIT Press*, 1990.