ARTÍCULO DE INVESTIGACIÓN ORIGINAL
Vol. 33, Núm. 2, Diciembre, 2012, pp. 87-99

Revista Mexicana de Ingeniería Biomédica
SOMIB

Un algoritmo de Consenso para la Búsqueda Aproximada de Patrones en Cadenas de Proteínas

A. Alba
M. Rubio-Rincón
M. Rodríguez-Kessler
E.R. Arce-Santana
M.O. Méndez

Facultad de Ciencias,
Universidad Autónoma de
San Luis Potosí

RESUMEN

En bioinformática, una de las principales herramientas que permiten la localización de características comunes en cadenas de proteínas o ADN de distintas especies es la búsqueda aproximada de cadenas. Desde el punto de vista computacional, la dificultad de la búsqueda aproximada de cadenas radica en encontrar medidas adecuadas para comparar dos cadenas de manera eficiente, dado que en muchos casos se desea realizar búsquedas en tiempo real, dentro de bases de datos de gran tamaño. En este artículo se propone un método novedoso para la búsqueda aproximada de cadenas basado en una generalización del algoritmo propuesto por Baeza-Yates y Perleberg en 1996 para calcular la distancia de Hamming entre dos secuencias, y una etapa de post-procesamiento que permite reducir de manera significativa el número de falsos positivos reportados por el algoritmo. El método propuesto ha sido evaluado a través de casos sintéticos con secuencias aleatorias, y con casos reales de secuencias de proteínas de plantas. Los resultados muestran que el algoritmo propuesto es altamente eficiente en términos computacionales y en especificidad, en particular al ser comparado con un método publicado anteriormente, basado en la correlación de fase.

Palabras clave: búsqueda aproximada de cadenas, secuencias de proteínas, bioinformática, algoritmos de consenso.

Correspondencia:
Alfonso Alba
Facultad de Ciencias,
UASLP. Av. Salvador Nava
Mtz. S/N, Zona
Universitaria. San Luis
Potosí, SLP, C.P. 78290
Tel: (444) 826-2486 ext.
2906

Fecha de recepción:
29/Octubre/2012
Fecha de aceptación:
10/Diciembre/2012

ABSTRACT

In bioinformatics, one of the main tools which allow scientists to find common characteristics in protein or DNA sequences of different species is the approximate matching of strings. From the computational point of view, the difficulty of approximate string matching lies in finding adequate measures to efficiently compare two strings, since, in many cases, one is interested in performing searches in real time, within large databases. In this paper we propose a novel method for approximate string matching based on a generalization of the algorithm proposed by Baeza-Yates and Perleberg in 1996 for computing the Hamming distance between two sequences. In addition, a post-processing stage which significantly reduces the number of false positives is presented. The proposed method has been evaluated in synthetic cases of random sequences, and with real cases of plant protein sequences. Results show that the proposed algorithm is highly efficient in computational terms and in specificity, especially when compared against a previously published method, which is based on the phase correlation function.

Keywords: approximate string matching, protein sequences, bioinformatics, consensus algorithms

INTRODUCCIÓN

La búsqueda aproximada de cadenas es un problema computacional que tiene múltiples aplicaciones en bioinformática, reconocimiento de patrones, búsquedas en bases de datos, correctores automáticos de ortografía, y detectores de plagio, entre otras [1,2]. En muchos casos, estas aplicaciones requieren que las búsquedas se realicen de la manera más eficiente posible. Por ejemplo, la corrección de ortografía en un procesador de textos moderno suele realizarse en tiempo real, mientras el usuario escribe. Por otra parte, bases de datos de secuencias genéticas como GenBank del Centro Nacional de Información Biotecnológica (NCBI) de Estados Unidos, o buscadores de Internet como Google o Yahoo, reciben un gran número de solicitudes de búsqueda por segundo y requieren algoritmos eficientes de búsqueda para minimizar el tiempo de espera del usuario.

El problema puede definirse como aquél donde uno desea encontrar instancias de una cadena de símbolos $a_1a_2...a_m$, la cual llamaremos cadena patrón, dentro de una cadena de búsqueda o texto $b_1b_2...b_n$, de manera que cada instancia encontrada puede diferir hasta cierto punto de la cadena patrón de una forma cuantificable a través de alguna medida de distancia entre cadenas. Las medidas más comúnmente utilizadas son la distancia de Hamming[3] y la distancia de Levenshtein[4]. La distancia de Hamming indica el número de símbolos no-correspondientes entre dos cadenas de la misma longitud, y la distancia de Levenshtein (también llamada distancia de edición) la cual representa el número mínimo de operaciones de edición (inserciones, supresiones y reemplazos de símbolos) que se requieren para transformar una de las cadenas en la otra. En este último caso, las dos cadenas a comparar no tienen necesariamente la misma longitud. Otras medidas pueden asignar distintos pesos a las operaciones de edición, o establecer equivalencias

entre símbolos, pero este tipo de distancias suelen utilizarse en aplicaciones específicas y dificultan la generalización de los algoritmos de búsqueda.

Matemáticamente, dado un alfabeto A, una cadena patrón $a_1a_2...a_m \in A^*$ (donde * representa la operación estrella de Kleene[5]), un texto de búsqueda $b_1b_2...b_n \in A^*$, una distancia entre cadenas $d : A^* \times A^* \rightarrow \mathbb{R}$, y una distancia máxima $k \in \mathbb{R}$, el problema consiste en encontrar todos los índices $j \in \{1, ..., n\}$ (es decir, posiciones dentro del texto) tales que $d(a_1...a_m, b_j...b_{j+m'}) < k$ para algún entero $m' \geq 0$.

Para hacer la búsqueda mas eficiente, un algoritmo de búsqueda aproximada de cadenas puede pre-procesar de alguna manera la cadena patrón y/o el texto de búsqueda. En el primer caso, la cadena patrón puede ser analizada, codificada, o representada de forma que se facilite la comparación con distintos segmentos del texto. Por ejemplo, los algoritmos basados en correlación pueden codificar el patrón como una secuencia de números para después obtener la transformada de Fourier de la secuencia, y así calcular la correlación en el dominio de la frecuencia. Otros algoritmos dividen el patrón en sub-secuencias (llamados n-gramas) y reducir el problema de búsqueda aproximada del patrón a la búsqueda exacta de los n-gramas. Por otra parte, el texto de búsqueda también puede ser pre-procesado para crear índices (típicamente en forma de árboles de sufijos) que después permiten seleccionar los segmentos del texto donde la búsqueda es relevante, logrando así una disminución substancial en el tiempo de búsqueda a cambio de una costosa etapa de pre-procesamiento. De esta manera, los algoritmos de búsqueda aproximada de cadenas pueden clasificarse en dos grupos: algoritmos en-línea, los cuales pueden pre-procesar de alguna manera la cadena patrón pero no el texto, y algoritmos fuera-de-línea, que son diseñados para buscar eficientemente sobre bases de datos indexadas. Estos últimos se aplican principalmente en bases de datos de gran tamaño donde la información es relativamente estática (e.g., búsquedas en internet y en bases de datos de bioinformática). Sin embargo, el estudio de algoritmos en-línea permanece muy vigente ya que de éstos algoritmos pueden también derivarse nuevos métodos fuera-de-línea. Para tener un panorama más amplio sobre estos métodos, se recomiendan al lector los siguientes trabajos de revisón: Ukkonen, 1985; Jokinen et al., 1996; Navarro, 2001; Navarro et al., 2001; Boytsov, 2011 [6-10].

Uno de los principales problemas de muchos algoritmos de búsqueda aproximada de cadenas es el gran número de falsos positivos que reportan; estos consisten en instancias que por azar son lo suficientemente parecidas a la cadena patrón, pero no son relevantes para la aplicación de interés. En búsquedas dentro de bases de datos grandes, es común observar del orden de decenas de miles hasta centenas de millones de falsos positivos, dependiendo también de la longitud de la cadena patrón[11]. El número de falsos positivos también depende en gran medida de la distancia máxima de edición k que debe haber entre la cadena patrón y cualquier instancia encontrada en el texto. Si k es demasiado pequeño, existe el riesgo de no encontrar algunas instancias relevantes (falsos negativos), mientras que si k es demasiado grande, un gran número de falsos positivos serán reportados. Dependiendo de la aplicación, puede ser preferible minimizar el número de falsos negativos (error de tipo II), aún a costa de un mayor número de falsos positivos, pero entonces se requerirá una etapa de post-procesamiento (automática o asistida por el usuario) para depurar los resultados y descartar las instancias no relevantes. Muchas veces este post-proceso consume más tiempo que el mismo proceso de búsqueda, lo cual puede ser una de las razones por la que los desarrolladores de bases de datos no siempre se molestan en usar métodos de búsqueda modernos y eficientes.

En este trabajo se presenta un nuevo método en-línea para la búsqueda aproximada de cadenas, el cual no se basa en una distancia de edición (como las distancias de Hamming o Levenshtein), sino en una medida de coincidencia c_j que se calcula para cada posición del texto $j = 1, ..., n$. Aquellas posiciones donde el índice sobrepase un umbral dado corresponderán a instancias de la cadena patrón. El algoritmo que se presenta es eficiente, fácil de implementar, y

puede paralelizarse para reducir los tiempos de búsqueda.

El presente manuscrito se organiza de la siguiente manera: en primer lugar, se repasa un método previamente publicado por Baeza-Yates y Perleberg[12] basado en la distancia de Hamming; es decir, donde la única operación de edición permitida es el reemplazo de símbolos. Posteriormente, se presentan las modificaciones realizadas a este algoritmo para generalizarlo a casos donde se permiten las inserciones y supresiones de símbolos. A continuación, se introduce una etapa de post-procesamiento que reducirá el número de falsos positivos de manera significativa y eficiente, completando así el algoritmo propuesto. En seguida, se presentan una serie de resultados obtenidos para casos de prueba sintéticos y para casos reales con cadenas de proteínas. Mediante estas pruebas se evalúa, de manera cualitativa y cuantitativa, la capacidad discriminativa del método, comparando contra un método previamente publicado, el cual se basa en la función de correlación de fase[13]. Finalmente, se presentan las conclusiones.

MÉTODO PROPUESTO

Algoritmo de Baeza-Yates y Perleberg

El algoritmo en el cual se basa el método propuesto[12] considera solamente el caso donde la única operación de edición permitida es la sustitución de símbolos; es decir, cuando la distancia de Hamming d_H entre el patrón y cualquier instancia encontrada es finita. Este algoritmo calcula, para cada posición $j = 1, ..., n$ en el texto, el número de símbolos c_j que coinciden entre la cadena patrón y la subcadena del texto (de la misma longitud) que inicia en j; es decir, se calcula $c_j = m - d_H(a_1...a_m, b_j...b_{j+m-1})$.

Consideremos primero el caso más simple, donde todos los símbolos del patrón $a_1...a_m$ son distintos. El algoritmo realiza primero una etapa de pre-procesamiento donde se genera una lista de las posiciones donde aparece cada símbolo dentro de la cadena patrón, relativas al primer símbolo de la cadena; por ejemplo, la posición relativa correspondiente al primer símbolo es cero. Específicamente, para cada símbolo $s \in A$, se define $\alpha(s) = j - 1$ si $a_j = s$. Si el símbolo s no aparece en la cadena patrón, entonces $\alpha(s)$ queda indefinido, o se le asigna un valor que represente esta situación. Luego, se inicializa un contador c_j con el valor cero para $j = 1, ..., n$. A continuación se recorre cada símbolo b_j del texto, para $j = 1, ..., n$; si $\alpha(b_j)$ está definido, entonces b_j corresponde a uno de los símbolos del patrón y existe la posibilidad de que una instancia de la cadena patrón inicie en la posición $j - \alpha(b_j)$ del texto. Para representar esta posibilidad, el algoritmo emitirá un voto, incrementando el contador correspondiente $c_{\{j-\alpha(b_j)\}}$. Una vez terminado el proceso, la posiciones de inicio de las instancias relevantes son aquellas que obtuvieron suficientes votos, es decir, aquellas j para las cuales $c_j \geq m - k$.

Solo resta extender el algoritmo para el caso en el que uno o más símbolos puedan aparecer múltiples veces dentro de la cadena patrón. En este caso, para cada símbolo $s \in A$ se debe llevar un registro de las posiciones en la cadena patrón donde aparece s. En otras palabras, ahora $\alpha(s)$ es un conjunto definido como $\alpha(s) = \{j : a_j = s\}$. Durante la etapa de recorrido del texto, si un símbolo b_j se encuentra en el patrón, entonces debe emitirse un voto para cada posición del texto donde podría iniciar una instancia, dado b_j; esto se logra incrementando los contadores c_{j-q} para todo $q \in \alpha(b_j)$.

Generalización a casos de inserción y supresión de símbolos

Consideremos ahora el caso de la búsqueda aproximada donde las instancias de interés pueden presentar hasta k inserciones o supresiones con respecto a la cadena patrón. En este caso, al analizar el símbolo b_j durante el recorrido del texto, no solo podría iniciar una instancia en cada posición $j - q$, $q \in \alpha(b_j)$, sino en cualquier posición entre $(j - q - k)$ y $(j - q + k)$, debido a que las inserciones y supresiones podrían modificar la posición relativa de los símbolos en una instancia hasta en k unidades. Por lo tanto, nuestra primer propuesta consiste en modificar el algoritmo

de Baeza-Yates y Perleberg de manera que, durante el recorrido del texto, para cada b_j se incrementen los contadores $c_{\{j-q+r\}}$ para todo $q \in \alpha(b_j)$ y $r = -k, ..., k$.

Una vez terminado el recorrido, el valor de cada c_j es el censo de todos los símbolos que podrían pertenecer a alguna instancia aproximada de la cadena patrón, con un máximo de k inserciones ó supresiones, que inicie en la posición j del texto. Nuevamente, las instancias de interés serán aquellas subcadenas del texto que inician en las posiciones j tales que $c_j \geq U$, para un cierto umbral U. Desafortunadamente, c_j ya no está relacionado directamente con la distancias de Hamming o Levenshtein, por lo cual no es del todo claro cuál debe ser el umbral óptimo para un valor dado de k.

Reducción de falsos positivos

Al igual que muchos algoritmos de búsqueda aproximada de cadenas, el algoritmo propuesto reporta un gran número de falsos positivos; sin embargo, la mayor parte de éstos corresponden a secuencias de posiciones adyacentes, alrededor de la ubicación de un verdadero positivo. Por ejemplo, consideremos una instancia que inicia en la posición j, digamos $b_j...b_{\{j+r\}}$, y supongamos que la distancia de Levenshtein entre esta instancia y el patrón es $d < k$. Entonces, la posición $j-1$ será también reportada como un positivo, ya que la subcadena $b_{\{j-1\}}...b_{\{j+r\}}$ solo requiere de una inserción más para ser idéntica a $b_j...b_{\{j+r\}}$, por lo que la distancia entre esta segunda instancia y el patrón será $d+1 \leq k$. De manera similar, es posible que un algoritmo reporte positivos en posiciones $j-2, j-3, ...$, así como en $j+1, j+2$, etc.

Con el objetivo de eliminar el número de positivos adyacentes, proponemos, a manera de post-procesamiento, conservar únicamente aquellos positivos j tales que $c_j \geq U$ y $c_{\{j-1\}} < U$; es decir, si se detectan múltiples positivos consecutivos, solamente se conservará el primero de ellos. Una desventaja de hacer esto es que posiblemente los positivos que se conserven no corresponden exactamente al verdadero inicio de las instancias, pero sabemos que los verdaderos inicios estarán cuando mucho k posiciones delante de los positivos obtenidos. En la mayoría de las aplicaciones, ubicar las verdaderas instancias a partir de los positivos que resulten del post-proceso es una tarea sencilla, incluso si se realiza de manera visual, y mucho menos tediosa que descartar los cientos o miles de falsos positivos que se obtienen sin la reducción propuesta.

Mejorando la localización

El post-proceso propuesto en la sección anterior reduce en gran medida el número de falsos positivos sin afectar seriamente la sensibilidad del algoritmo. Sin embargo, los positivos reportados pueden estar alejados hasta k posiciones con respecto a los verdaderos positivos correspondientes. Una manera simple de mejorar la localización de los positivos consiste en modificar la etapa de post-procesamiento de manera que para cada secuencia de positivos consecutivos, se elija aquél que corresponde al máximo valor de censo, en lugar de simplemente elegir el primer positivo de la secuencia. Esta modificación se propone a manera de regla práctica, mas es necesario realizar un mayor número de pruebas para determinar si es adecuado aplicarla de manera general.

RESULTADOS Y DISCUSIÓN

Para evaluar el algoritmo propuesto, se ha desarrollado una implementación eficiente de éste en lenguaje C. Como punto de comparación, se cuenta también con una implementación eficiente en C de un algoritmo para la búsqueda aproximada de cadenas basada en el método de correlación de fase, el cual fue recientemente publicado[13] y se basa también en un criterio distinto a las distancias de Hamming o Levenshtein para comparar dos cadenas. Para que la comparación sea justa, ninguno de los algoritmos se ha paralelizado, y todas las pruebas se ejecutan (de manera secuencial, en un solo núcleo del CPU) en la misma computadora, basada en un procesador Intel Core2Duo a 2.4 Ghz con 4 Gb en RAM. El algoritmo basado en correlación de fase hace uso de la librería

FFTW[14] para el cálculo eficiente de la FFT.

Pruebas con cadenas aleatorias

La primer prueba consiste en una serie de 100 casos sintéticos donde tanto la cadena patrón como la cadena de búsqueda se han generado aleatoriamente, a partir de un alfabeto de 26 símbolos. La longitud de la cadena patrón es $m = 32$, mientras que la cadena de búsqueda contiene $n = 2^{20}$ símbolos (1 Mbyte). La cadena de búsqueda contiene $N = 256$ instancias aproximadas (no traslapadas) de la cadena patrón, cada una de las cuales se obtiene aplicando un máximo de k operaciones de edición (inserciones, supresiones y substituciones) al patrón. Para cada caso de prueba se conocen las posiciones de inicio $I_r, r = 1, ..., N$ de cada una de las verdaderas instancias presentes en la cadena de búsqueda.

Luego, se introduce cada caso de prueba como entrada del algoritmo de búsqueda, el cual devuelve como resultado una lista $\hat{I}_s, s = 1, ..., P$ de las P posiciones donde se estima que existe una instancia (resultados positivos). Recordemos que la posición reportada por el algoritmo, en caso de corresponder a un verdadero positivo, puede diferir hasta en k posiciones de la verdadera posición. Por lo tanto, para evaluar los resultados se establece una función parcial $f : \{1, ..., P\} \rightarrow \{1, ..., N\}$ donde $f(s) = r$ si $|\hat{I}_s - I_r| \leq k$ y $f(s') \neq r$ para todo $s' < s$; de otra manera, $f(s) = 0$. En otras palabras, si $\hat{I}_s$ corresponde a un verdadero positivo, digamos I_r, entonces no será posible asociar ningún otro positivo $\hat{I}_t$, $t \neq s$ con I_r, aún cuando ambos sean suficientemente cercanos. De esta forma, evitamos que durante la evaluación se reporte un número de verdaderos positivos mayor que N. A partir de lo anterior, podemos definir el número de verdaderos positivos (TP), falsos positivos (FP), y falsos negativos (FN) como sigue:

$$P = \sum_{s=1}^{p} (1 - \delta(f(s))), FP = P - TP, FN = N - TP \quad (1)$$

donde $\delta(x)$ es 1 si $x = 0$, y 0 en cualquier otro caso (función delta de Kronecker).

En una primer instancia se comparan los resultados del algoritmo propuesto antes y después de aplicar la etapa de post-procesamiento (PP) para reducir los falsos positivos. El post-proceso considera la regla práctica descrita anteriormente para mejorar la localización de los positivos; sin embargo, esto no afecta de ninguna manera el conteo de verdaderos y falsos positivos obtenidos.

Los resultados de las pruebas sintéticas se muestran en la Figura 1, para valores de $k = 3, 6, 10$ y distintos valores para el umbral U. La columna izquierda muestra las gráficas de TP en función del umbral obtenidas en cada caso. Es importante notar que solamente se distingue una gráfica, debido a que los valores de TP son prácticamente iguales con y sin etapa de post-procesamiento; en otras palabras, el post-procesamiento no parece afectar la sensibilidad del algoritmo.

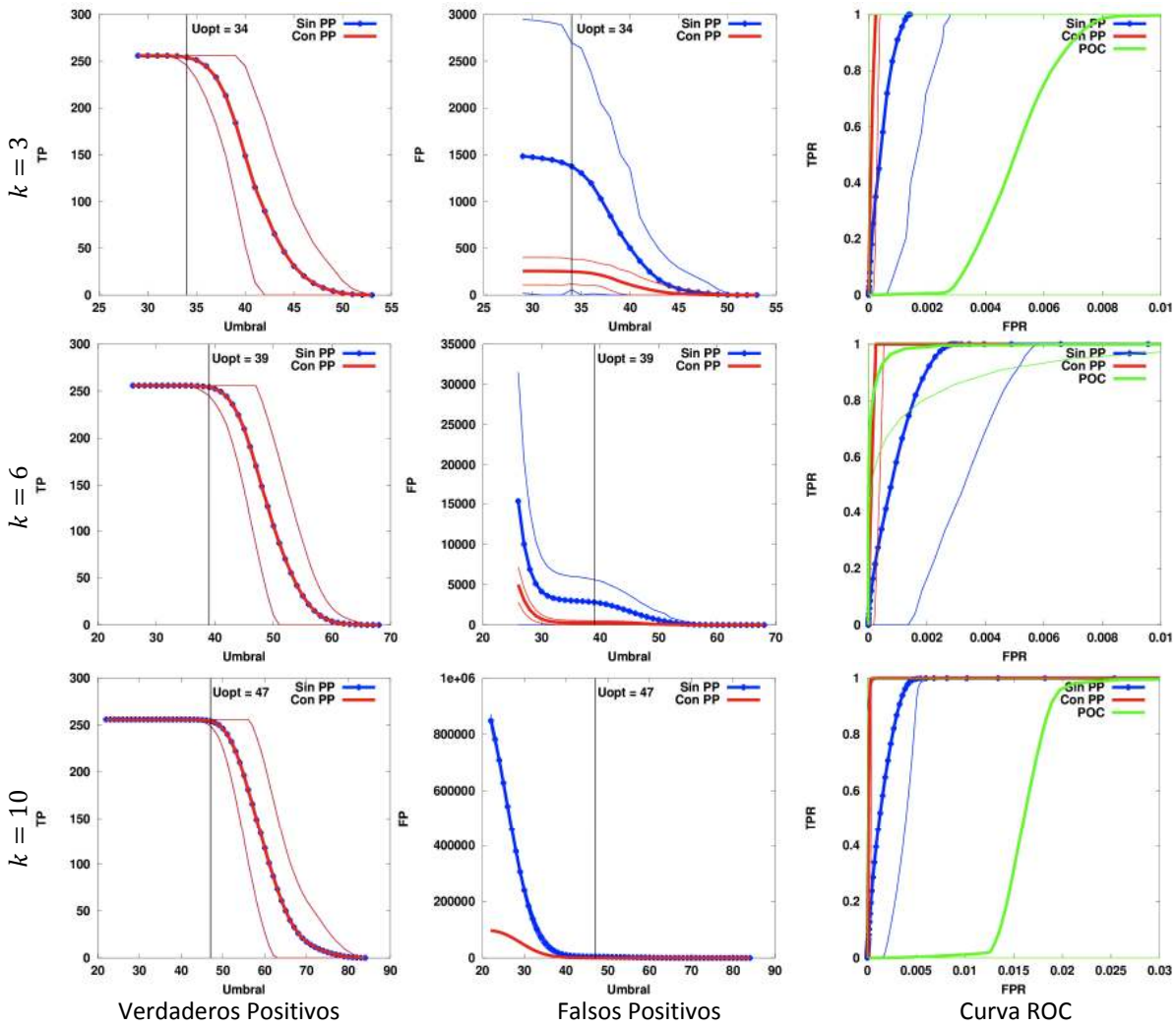

Fig. 1. Resultados obtenidos mediante el algoritmo propuesto (con y sin etapa de post-procesamiento (PP) para reducción de falsos positivos) para casos de prueba sintéticos. Las curvas ROC (columna derecha) muestran también el resultado obtenido con un algoritmo basado en correlación de fase (POC)[13]. Las líneas gruesas representan el promedio sobre 100 casos de prueba, mientras que las líneas delgadas representan una desviación estándar sobre el promedio.

Tabla 1: Resultados de TP (promedio y desviación estándar) y FP (promedio y desviación estándar), y tiempo de cómputo obtenidos con los umbrales óptimos (U_{opt}), tanto para el algoritmo propuesto (con y sin post-procesamiento) como para el algoritmo basado en correlación de fase (POC)[13].

	$k=3$			$k=6$			$k=10$		
	Sin PP	Con PP	POC	Sin PP	Con PP	POC	Sin PP	Con PP	POC
U_{opt}	34	34	0.14	39	39	0.12	47	47	0.10
TP	254.37	254.33	254.49	254.37	254.28	254.81	254.16	253.93	253.09
σ_{TP}	8.50	8.57	6.64	9.07	9.12	1.27	5.12	5.16	2.91
FP	1378.03	250.95	5693.25	2808.39	260.81	2605.93	4780.45	333.06	30931.49
σ_{FP}	1321.24	133.48	43924.62	2802.03	281.09	13302.69	742.61	100.90	63768.21
t (ms)	34.8	36.8	316.5	39.8	41.4	314.1	51.6	55.2	316.2

Aquí se muestra también el umbral óptimo U_{opt}, el cual definimos como el mayor umbral para el cual el número promedio de falsos negativos es menor a uno. En todas las gráficas se muestra con una línea gruesa el valor promedio (estimado sobre los 100 casos de prueba), y con líneas delgadas el valor del promedio mas/menos una desviación estándar.

La segunda columna de la Figura 1 muestra las gráficas de FP en función del umbral. Aquí se aprecia claramente la ventaja de utilizar el post-proceso para reducir de manera significativa el número de falsos positivos.

En la tercera columna de la Figura 1 se muestran las curvas ROC (tasa de TP contra tasa de FP, obtenidas al variar el umbral) para cada caso. Aquí hemos decidido incluir los resultados obtenidos con el algoritmo basado en correlación de fase, como punto de comparación. Un algoritmo de detección o clasificación se considera mejor que otro si el área bajo su curva ROC es mayor. Por lo tanto, las gráficas sugieren que el algoritmo propuesto, con un umbral adecuado, puede tener una mayor sensibilidad y especificidad que el método de correlación de fase, en particular cuando se aplica la etapa de post-procesamiento. Cabe mencionar que, aunque el post-procesamiento propuesto puede aplicarse también a los resultados obtenidos por el método de correlación de fase, en las pruebas realizadas no se apreciaron beneficios significativos debido a que los positivos reportados por este método no suelen ser consecutivos.

Finalmente, la Tabla 1 muestra los resultados numéricos (promedios y desviaciones estándar) de las pruebas realizadas obtenidos con el umbral óptimo encontrado para $k = 3, 6, 10$, incluyendo el tiempo de cómputo promedio para cada caso de prueba. Claramente, el tiempo requerido por el post-procesamiento es casi despreciable con respecto al tiempo total de cómputo, pero el número de FP es significativamente menor, en algunos casos hasta por un orden de magnitud. Por otra parte, si bien el costo computacional del método propuesto se incrementa ligeramente (de manera sub-lineal) con respecto a k, aún así es considerablemente más eficiente que el método basado en correlación de fase.

Comparación con el método shift-or

Uno de los métodos mas populares para la búsqueda aproximada de cadenas es el método shift-or, originalmente propuesto por Baeza-Yates y Gonnet[15]. El método se basa en calcular y actualizar un conjunto de máscaras binarias que indican cuáles prefijos de la cadena patrón coinciden con una cierta subcadena del texto de búsqueda. La mayor parte de los cálculos realizados por el algoritmo pueden representarse mediante operaciones binarias tales como corrimientos y disyunción (de ahí el nombre "shift-or"), lo cual permite implementaciones extremadamente eficientes. Posiblemente la implementación mas conocida reside en el programa agrep escrito por Wu y Manber[16]. Este programa busca, de manera aproximada, una cadena patrón dentro de uno o más archivos que contienen registros, donde un registro es típicamente una línea de texto. El programa reporta aquellos registros donde se encuentre por lo menos una instancia aproximada del patrón, aunque no reporta la posición exacta de las instancias.

Para comparar el método propuesto con el método shift-or, hemos implementado un programa similar a agrep, pero basado en el método propuesto de consenso. De manera similar a la prueba anterior, hemos generado series de 100 casos de prueba, cada uno de los cuales consiste en una cadena patrón de longitud $m = 30$ generada de manera aleatoria a partir de un alfabeto de 26 símbolos, y un archivo de registros donde exactamente 1,024 de los registros contienen instancias aproximadas del patrón. El número total de registros en cada archivo varía ligeramente pero es en promedio 16,384. En una primer instancia se generaron series (de 100 casos cada una) para valores de $k = 0, 1, ..., 8$, donde k representa tanto el número de operaciones de edición aplicadas a cada instancia de la cadena patrón, como el parámetro de distancia máxima utilizado en ambos algoritmos (algoritmo propuesto y agrep). Finalmente, para el algoritmo propuesto se ha establecido de manera empírica un umbral $U = (k + 1)(m + k)$, considerando que existe un máximo de $m + k$ símbolos en una instancia del

patrón, cada uno de los cuales incrementa un número de contadores proporcional a k, y que cuando $k = 0$ (es decir, en una búsqueda exacta) se requiere tener $U = m$.

Para estas pruebas, el número de verdaderos positivos (TP) es el número de registros reportados por cada uno de los programas que corresponden a registros donde se ubicaron las instancias verdaderas, mientras que un falso positivo (FP) se cuenta como un registro reportado por los programas que no contiene una instancia verdadera, salvo por azar (aunque dada la longitud del patrón y el tamaño del alfabeto, la probabilidad de generar una instancia por azar es insignificante).

Los resultados se presentan en las gráficas superiores de la Figura 2. Nuevamente se utiliza una línea gruesa para indicar el promedio de TP (gráfica superior izquierda) o promedio de FP (gráfica superior derecha), y líneas delgadas para representar una desviación estándard con respecto a la media. Se puede observar que ambos algoritmos mantienen una sensibilidad promedio muy cercana al 100% (TP=1024), aunque el algoritmo propuesto es ligeramente menos sensible. Por otra parte, mientras agrep no reporta falsos positivos en ningún caso, el algoritmo propuesto reporta un número creciente de FPs conforme aumenta el error máximo permitido k. Creemos que esto se debe a que agrep se adapta bien a los datos, debido a que ambos están basados en la distancia de edición; sin embargo, el algoritmo propuesto utiliza una métrica diferente para estimar la similaridad entre cadenas.

En un segundo experimento, se repitió la prueba anterior pero ahora fijando el número máximo de ediciones aplicadas a las instancias en 8, y variando únicamente el parámetro k utilizando en ambos programas, de manera que con valores de k menores a 8, los algoritmos encontrarán solamente una fracción de las verdaderas instancias. Los resultados de este experimento se muestran en las gráficas inferiores de la Figura 2. Claramente se puede observar que el algoritmo propuesto muestra una sensibilidad considerablemente mayor al método shift-or en el caso donde uno no conoce a priori el valor de k que debe utilizarse. Nuevamente esto refleja el hecho de que la métricas utilizadas por el algoritmo propuesto, y el papel de k dentro de esas métricas, son muy distintos a la distancia de Levenshtein, pero igualmente útiles para la búsqueda aproximada de patrones.

Aunque el programa basado en el método propuesto no ha sido optimizado a fondo, pudimos observar tiempos de cómputo muy competitivos, donde el tiempo promedio para cada caso de prueba con $k = 8$ fue de 80 ms con el algoritmo propuesto y 57 ms con agrep; sin embargo, estos tiempos corresponden a la ejecución total de cada uno de los programas, incluyendo la carga de los archivos y la impresión de resultados.

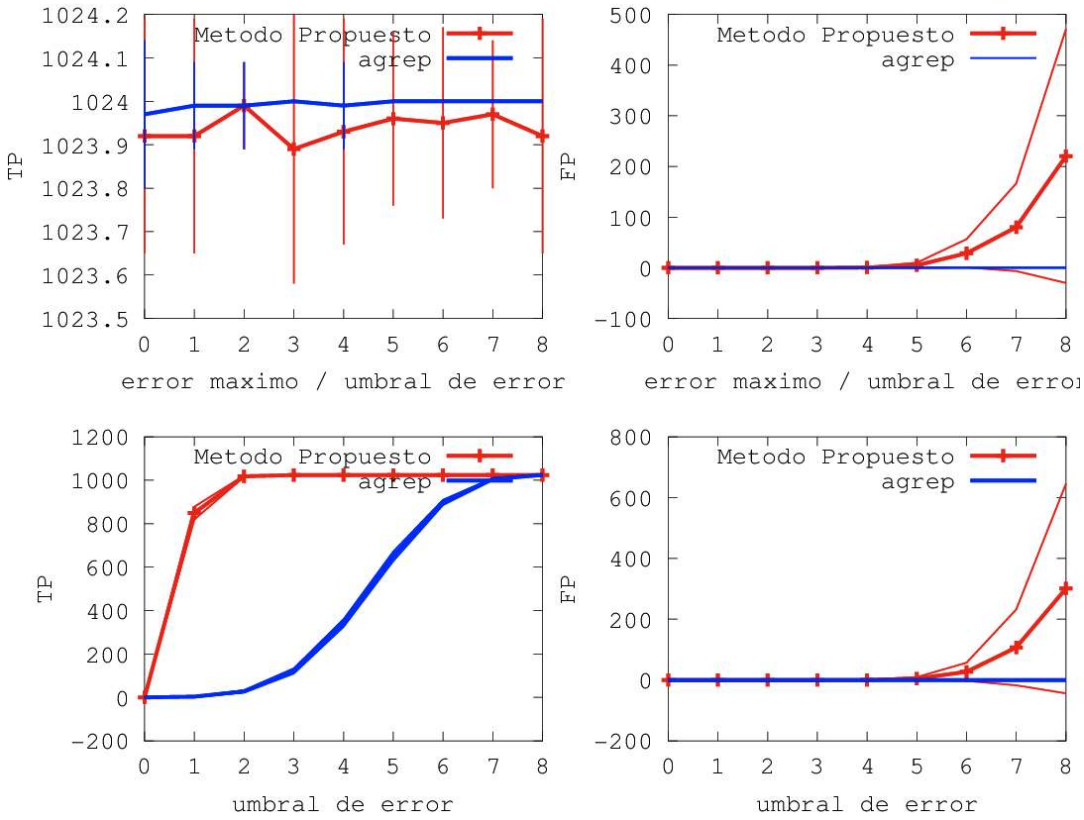

Fig. 2. Resultados de la comparación entre el método propuesto y el método shift-or (implementado en el programa agrep). Las líneas gruesas representan el promedio sobre 100 casos de prueba, mientras que las líneas delgadas representan una desviación estándar sobre el promedio. Para mayor legibilidad, en la primer gráfica se utilizan bigotes para mostrar la desviación estándar.

Aplicación a la búsqueda de patrones en secuencias de proteínas reales

Para evaluar la eficiencia del algoritmo propuesto en casos reales, se realizaron búsquedas dentro de seis secuencias de proteínas de las plantas *Opuntia streptacantha* (nopal), *Arabidopsis thaliana* y *A. lyrata* [17,18,19]. Las proteínas dehidrinas (DHN) se caracterizan por la presencia de uno o más segmentos-K que consisten en aproximadamente 15 residuos de aminoácidos [EKKGIM(E/D)KIKEKLPG], los cuales forman una probable estructura de α-hélice amfipática[20,21]. Las proteínas DHN pueden también contener segmentos-S [LHRSGS4-10(E/D)3] formados por una serie de 4 a 10 residuos de serina, y segmentos-Y [(V/T)D(E/Q)YGNP] que se localizan cerca del extremo amino terminal de estas proteínas[20,21]. En general, las proteínas DHN comparten baja identidad de secuencia, y la identificación apropiada de los motivos conservados es una tarea importante para la clasificación de estas proteínas en los sub-tipos: Y_nSK_2, K_n, SK_n, K_nS, y Y_2K_n.

Instancias encontradas		
Segmento-S	Segmento-K	Segmento-Y
LHRSGS	EKKGIMDKIKEKLPG	DEYGNP
TP = 4, FP = 1, FN = 0	TP = 17, FP = 5, FN = 0	TP = 7, FP = 0, FN = 1

Fig. 3. Resultados obtenidos con secuencias de proteínas reales. Las posiciones subrayadas corresponden a los positivos encontrados por el algoritmo propuesto. Aquellos positivos a una distancia menor o igual que k de una verdadera instancia se consideran verdaderos positivos, de otra manera se cuentan como falsos positivos. La cadena dentro del recuadro verde corresponde a un falso negativo; es decir, una instancia no detectada por el algoritmo.

En cada una de las seis secuencias de proteínas seleccionadas, se realizó una búsqueda de los siguientes patrones de aminoácidos: EKKGIMDKIKEKLPG (segmento-K), LHRSGS (segmento-S), y DEYGNP (segmento-Y). Los parámetros utilizados en todas las secuencias fueron los siguientes: $k = [0.3\text{m}]$ para el segmento-K y segmento-S, donde m es la longitud de la cadena patrón a buscar, y $k = 0$ para el segmento-Y. El umbral utilizado en todos los casos fue $U = m + 2(k - 1)$. También se utilizó la versión

modificada del post-procesamiento para mejorar la localización de los verdaderos positivos. El tiempo total de procesamiento (para un total de 18 búsquedas) fue menor a 2 ms (al algoritmo basado en la POC le toma 30 ms hacer la misma búsqueda). Los resultados se muestran en la Figura 2: cada uno de los patrones se representan con un color distinto, según la leyenda, y las posiciones marcadas por el algoritmo como positivos se muestran mediante el subrayado del símbolo correspondiente. Todas, excepto una de las verdaderas instancias fueron localizadas con un error máximo de localización de k posiciones (donde $k = 5$ para el segmento-K, $k = 2$ para el segmento-S, y k=0 para el segmento-Y). El algoritmo reportó también cinco falsos positivos durante la búsqueda del segmento-K, posiblemente debido al alto nivel de tolerancia impuesto por fijar k=5; aún así, la mayor parte de estos falsos positivos son descartables mediante simple inspección visual. Por otra parte, la búsqueda del segmento-S generó también un falso positivo en la secuencia At4g39130. El error mas grave de toda la prueba fue un falso negativo (FN) en la misma secuencia At4g39130, correspondiente a un segmento-Y que no fue reportado por el algoritmo (marcado en la Figura 3 dentro de un recuadro); esta instancia es difícil de detectar ya que tiene 3 diferencias con respecto al patrón, el cual tiene una longitud de 6 símbolos, dando como resultado un índice de similaridad relativamente bajo.

Las pruebas sintéticas realizadas para evaluar el método propuesto muestran un amplio margen de ventaja, en términos de especificidad y tiempo de cómputo, con respecto al método basado en correlación de fase. Comparando contra un algoritmo ubicuo como agrep de Wu y Manber, el método propuesto muestra un buen desempeño en términos de sensibilidad y tiempo de cómputo cuando el error máximo permitido no es demasiado grande; sin embargo, cuando este parámetro se desconoce, el método propuesto puede tener una mejor sensibilidad a las instancias de interés. Por otra parte, a diferencia de la mayoría de las implementaciones basadas en el método shift-or, el algoritmo propuesto no impone limitaciones técnicas a la longitud de la cadena patrón.

Una desventaja del algoritmo propuesto radica en que al aplicarlo a secuencias donde el alfabeto es reducido, el número de falsos positivos aumenta considerablemente. Esto limita la aplicabilidad del algoritmo en su estado actual a secuencias de ADN donde el alfabeto consta solamente de 4 símbolos. En el futuro, se investigará la manera de mejorar la especificidad para alfabetos pequeños, por ejemplo, mediante una recodificación de las cadenas en un alfabeto de mayor tamaño.

Finalmente, las pruebas realizadas con secuencias de proteínas reales sugieren que aún es necesario mejorar la etapa de post-procesamiento para eliminar falsos positivos que son demasiado cercanos entre sí.

CONCLUSIONES

En este trabajo se ha presentado un nuevo método para la búsqueda aproximada de cadenas basado en dos ideas novedosas: la primera consiste en una generalización del algoritmo de Baeza-Yates y Perleberg para calcular la distancia de Hamming, a los casos donde se desean considerar también las inserciones o supresiones de símbolos, lo cual resulta en una búsqueda extremadamente eficiente. La segunda contribución consiste en una etapa de post-procesamiento para reducir de manera significativa el número de falsos positivos reportados por el algoritmo.

AGRADECIMIENTOS

Para el desarrollo de este trabajo se ha contado con el apoyo del Consejo Nacional de Ciencia y Tecnología (CONACyT) mediante el proyecto de Ciencia Básica #154623.

REFERENCIAS

1. Smith TF, Waterman MS. “Identification of common molecular subsequences”. Journal of Molecular Biology, 1981; 147: 195-197.

2. Altschul SF, Gish W, Miller W, Myers EW,

Lipman DJ. "Basic local alignment search tool". Journal of Molecular Biology, 1990; 215: 403-410.

3. Hamming RW. "Error detecting and error correcting codes". Bell System Technical Journal, 1950; 29(2): 147-160.

4. Levenshtein VI. "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady, 1966; 10: 707-710.

5. Howie JM. Automata and Languages. Oxford University Press, 1991.

6. Ukkonen E. "Algorithms for approximate string matching". Inf. Control, 1985; 64(1-3): 100-118.

7. Jokinen P, Tarhio J, Ukkonen E. "A comparison of approximate string matching algorithms". Softw. Pract. Exper., 1996; 26(12): 1439-1458.

8. Navarro G. "A guided tour to approximate string matching". ACM Computing Surveys, 2001; 33(1): 31-88.

9. Navarro G, Baeza-Yates RA, Sutinen E, Tarhio J. "Indexing methods for approximate string matching". IEEE Data Engineering Bulletin, 2001; 24(4): 19-27.

10. Boytsov L. "Indexing methods for approximate dictionary searching: Comparative analysis". J. Exp. Algorithmics, 2011; 16: 1.1:1.1-1.1:1.91.

11. Buhler J. "Efficient large-scale sequence comparison by locality sensitive hashing". Bioinformatics, 2001; 17(5): 419-428.

12. Baeza-Yates RA, Perleberg CH. "Fast and practical approximate string matching". Inf. Process. Lett., 1996; 59(1): 21-27.

13. Alba A, Rodriguez-Kessler M, Arce-Santana ER, Mendez MO. "Approximate string matching using phase correlation". Proceedings of the 34th Annual International Conference of the IEEE EMBS, 2012; pp. 6309-6312.

14. Frigo M, Johnson SG. "The Design and Implementation of FFTW3". Proc. IEEE, 2005; 93(2): 216-231.

15. Baeza-Yates RA, Gonnet GH. "A new approach to text searching". Proceedings of the 12th Annual ACM-SIGIR Conference on Information Retrieval, 1989; pp. 168-175.

16. Wu S, Manber U. "Fast Text Searching With Errors". Technical Report TR 91-11, Department of Computer Science, University of Arizona, 1991.

17. Ochoa-Alfaro A, Rodríguez-Kessler M, Pérez-Morales M, Delgado-Sánchez P, Cuevas-Velazquez C, Gómez-Anduro G, Jiménez-Bremont J. "Functional characterization of an acidic SK3 dehydrin isolated from an Opuntia streptacantha cDNA library". Planta, 2012; 235: 565-578.

18. Hundertmark M, Hincha DK. "LEA (late embryogenesis abundant) proteins and their encoding genes in Arabidopsis thaliana". BMC Genomics, 2008; 9: 118.

19. Jiménez-Bremont JF, Maruri-López I, Ochoa-Alfaro A, Delgado-Sánchez P, Bravo J, Rodríguez-Kessler M. "LEA gene introns: is the intron of dehydrin genes a characteristic of the serine-segment?". Plant Mol Biol Rep. (DOI: 10.1007/s11105-012-0483-x). In press.

20. Allagulova CR, Gimalov FR, Shakirova FM, Vakhitov VA. "The plant dehydrins: structure and putative functions". Biochemistry (Moscow), 2003; 68: 945-951.

21. Kosová K, Prásil IT, Vítámvás P. "Role of dehydrins in plant stress response" En: Pessarakli M, editor, Handbook of Plant and Crop Stress, 3rd ed., CRC Press (Florida), 2010: 239-285.