ARTÍCULO DE INVESTIGACIÓN

Vol. 40 | No. 1 | **ENERO - ABRIL 2019** | pp 1-13

REVISTA MEXICANA DE Ingeniería Biomédica

dx.doi.org/10.17488/RMIB.40.1.4

E-LOCATION ID: **e201821**

ABPSE: Alineador de ADN Basado en Paralelismo a Nivel de Bit y la Estrategia Siembra y Extiende

ABPSE: DNA Aligner Based on Bit-level Parallelism and the Seed and Extend Strategy

D. Pacheco-Bautista, J. Martínez-Oviedo, R. Carreño-Aguilera, I. Algredo-Badillo, S. Sánchez-Sánchez

Universidad del Istmo

RESUMEN

La alineación de ADN es un proceso clave para la reconstrucción de genomas, a partir de los millones de lecturas cortas producidas por las máquinas de secuenciación paralela masiva. Tal proceso suele realizarse mediante algoritmos con elevada complejidad espacial y temporal, requiriendo varias horas para entregar los resultados, así como decenas de GB de RAM. Esto ha motivado la búsqueda de nuevos algoritmos y/o estrategias que permitan disminuir los tiempos de ejecución, mientras se utilizan recursos mínimos de memoria. En este artículo se presenta ABPSE, un nuevo alineador de ADN que combina el algoritmo de Ferragina y Manzini (o índices de FM) y el algoritmo de Myers, mediante la estrategia siembra y extiende. En la siembra, los índices de FM permiten calcular de manera rápida regiones con alta probabilidad de alineación; mientras que en la extensión, el algoritmo de Myers refina la alineación utilizando operaciones basadas en vectores de bits, calculando simultáneamente varias celdas de la matriz de programación dinámica. Los resultados muestran un 96.1% de lecturas alineadas correctamente, un factor de aceleración de 2.45x en relación a BWA-SW y un uso de memoria de apenas 7.6 GB, cuando se alinea el genoma humano completo.

PALABRAS CLAVE: ADN; Bioinformática; Myers; Siembra y extiende; Índices de FM

ABSTRACT

DNA alignment is a key process in the assembly of genomes from the millions of short reads that are produced by massive parallel sequencing machines. Such a process is usually done by means of high spatial and temporal complexity algorithms, which takes hours to deliver the results as well as tens of GB of RAM. This has prompted the search for new algorithms and/or strategies that allow shorter runtimes, while using minimal memory footprint. In this article, we present ABPSE, a new DNA aligner that combines the Ferragina and Manzini algorithm (or FM indexes) and the Myers algorithm, by means of the seed and extend strategy. In the seeding, the FM indices allow a rapid calculation of the regions with high probability of alignment. In the extension, the Myers algorithm refines the alignment using operations based on bit vectors. It simultaneously calculates several cells of the dynamic programming matrix. The results show 96.1% of correctly aligned reads, an acceleration factor of 2.45x in relation to BWA-SW and a memory footprint of only 7.6 GB when aligning the entire human genome.

KEYWORDS: DNA; Bioinformatics; Myers; Seed-and-extend; FM index

Correspondencia

DESTINATARIO: **Daniel Pacheco Bautista**

INSTITUCIÓN: **Universidad del Istmo**

DIRECCIÓN: **Cd. Universitaria S/N, Bo. Santa Cruz Tagojaba, C. P. 70760, Tehuantepec, Oaxaca, México**

CORREO ELECTRÓNICO:

dpachecob@bianni.unistmo.edu.mx

Fecha de recepción:

30 de mayo de 2018

Fecha de aceptación:

29 de noviembre de 2018

INTRODUCCIÓN

La secuenciación de ADN es un proceso que permite obtener el orden de cada uno de los nucleótidos que conforman la molécula de ADN. Tiene una larga lista de aplicaciones y es tecnología clave para la investigación de algunos tipos de cáncer, así como el desarrollo de la medicina genómica, la biología y la agricultura. Las máquinas de secuenciación paralela masiva son tecnologías capaces de secuenciar millones de cadenas de ADN al día [1-3], sin embargo, procesan fragmentos con un número muy pequeño de nucleótidos (entre 35 y 1100), por lo que el resultado de la secuenciación no es un genoma completo, sino pequeñas lecturas cortas que representan fragmentos del mismo. Una manera de reconstruir el genoma a partir de los millones de lecturas cortas es mediante el proceso de alineación [3], el cual consiste en ubicar cada lectura corta tomando como referencia un genoma secuenciado previamente. No obstante, los billones de lecturas cortas, así como la gran longitud del genoma de referencia (3000 millones de nucleótidos para el genoma humano) complican el proceso, además de que deben tomarse en cuenta las diferencias biológicas entre ambas cadenas (inserciones, supresiones o mutaciones de nucleótidos), así como los posibles errores de las máquinas de secuenciación. Para realizar el proceso de alineación se utilizan diferentes algoritmos de elevada complejidad temporal y espacial, algunos basados en programación dinámica [4-5], siendo muy precisos pero requiriendo gran cantidad de recursos computacionales, y otros basados en heurísticas [6-7], los cuales son menos exactos pero más rápidos. Alternativamente los métodos pueden utilizar estimaciones estadísticas [8], basadas principalmente en métodos bayesianos o de máxima verosimilitud o incluso aplicar herramientas de procesamiento digital de señales [9], entre otras técnicas.

La estrategia siembra y extiende

Recientemente se ha optado por combinar las técnicas heurísticas con aquellas basadas en programación dinámica para acelerar la alineación de lecturas, realizando un balance entre velocidad y precisión. La más importante de tales estrategias se denomina siembra y extiende (Figura 1).

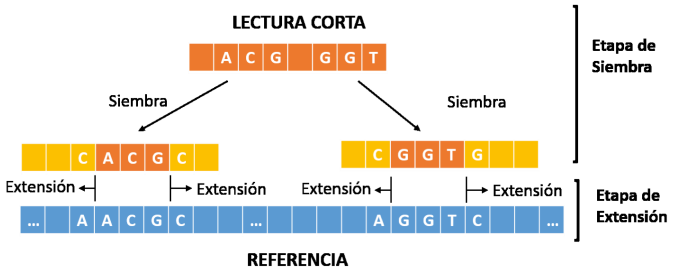

FIGURA 1. La estrategia siembra y extiende.

Durante la etapa de siembra se intenta encontrar una subcadena de la lectura corta (semilla) que se alinee exactamente en uno o más lugares del genoma de referencia. Esta aproximación se basa en la premisa de que si existe un alto grado de similitud de una subcadena de la lectura en una región de la referencia, entonces es más probable que exista un buen alineamiento de toda la lectura corta en esta zona. Para la etapa de extensión se intenta extender la semilla en ambas direcciones, en esta etapa toda la lectura corta es alineada respecto a la cadena de referencia en las zonas encontradas durante la siembra. La extensión permite determinar de forma precisa la existencia de mutaciones, inserciones o supresiones en la lectura respecto a la referencia. Muchos alineadores modernos implementan la técnica siembra y extiende, tal es el caso de BWA-SW [10], BWA-MEM [11], Bowtie2 [12] y Cushaw2 [13]. Para la etapa de siembra, dichos programas realizan un pre-procesamiento del genoma, obteniendo índices de búsqueda eficientes mediante los algoritmos basados en Tablas Hash [14], o en la Transformada de Burrows-Wheeler [15]. Posteriormente, para la extensión implementan algoritmos basados en programación dinámica, tales como el Smith-Waterman [16], o el Needleman-Wunsch [17].

En este artículo se presenta el desarrollo de ABPSE, un alineador eficiente en tiempo y espacio, que realiza la alineación de lecturas cortas mediante la estrategia

siembra y extiende. La siembra utiliza los índices de FM [18] para realizar las búsquedas exactas de semillas, y la extensión el algoritmo de programación dinámica basado en paralelismo a nivel de bit propuesto por Myers [19]. En particular, el algoritmo de Myers está basado en la distancia de Levenshtein para calcular la similitud entre cadenas, tal algoritmo toma las ventajas del paralelismo a nivel de bit del tamaño de palabra del procesador de una computadora, esto es eficiente debido a que los procesadores realizan cálculos con un tamaño entero de palabra en un ciclo de memoria. Básicamente, los cálculos de las puntuaciones y las comparaciones de cadenas son obtenidos simultáneamente mediante una serie de operaciones binarias que comprenden AND, OR, XOR, complementos, desplazamientos y sumas. Después de una profunda revisión de la literatura, se encontró que el algoritmo de Myers no ha sido usado antes en estas aplicaciones.

El resto del artículo se organiza de la siguiente manera, en la siguiente sección se presenta el diseño del software de alineación, iniciando con la descripción del esquema a bloques general propuesto, y culminando con la descripción detallada de los algoritmos y estrategias en cada módulo dentro del mismo. En una sección posterior, se presentan, analizan y comparan los resultados de ejecución obtenidos. Finalmente, se establecen las conclusiones principales.

METODOLOGÍA

El modelo a bloques del programa de alineación desarrollado se muestra en la Figura 2. La implementación del mismo fue realizada utilizando el modelo secuencial de desarrollo de software. Los datos de entrada del alineador son: los archivos con los índices de FM del genoma de referencia, el archivo con las lecturas cortas en formato FASTQ, la cadena de referencia comprimida y un valor de error opcional *n* proporcionado por el usuario, que limita el número de errores permitidos en la alineación. La salida es un archivo con la posición y la trayectoria de alineación en el formato SAM [20].

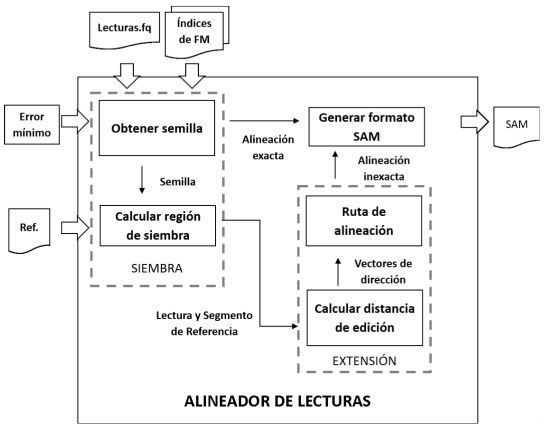

FIGURA 2. Esquema general del alineador genérico.

El proceso inicia en el módulo de siembra, obteniendo las semillas a partir de la lectura corta. Para determinar las semillas, primero se intenta alinear la lectura entera respecto a la cadena de referencia, mediante el algoritmo de búsqueda hacia atrás de Ferragina y Manzini [18]. Si la lectura se alinea de forma exacta entonces se excluye todo el proceso de extensión y se procede de inmediato a generar los resultados de la alineación en formato SAM. En caso contrario, se obtienen las subcadenas de la lectura que se alineen de manera exacta a la referencia, siendo consideradas semillas si tienen una longitud no menor a cierta longitud mínima precalculada. Una vez obtenida una semilla, el segundo paso es calcular la región real donde ésta se ha alineado en la cadena de referencia. El resultado de este paso es una subcadena de la referencia que será usada como entrada en la fase de extensión.

Una vez finalizada la etapa de siembra se inicia la etapa de extensión, la cual tiene como objetivo alinear de forma inexacta la lectura corta y el segmento de la referencia encontrado en la etapa de siembra. El proceso inicia con el cálculo de la distancia de edición entre ambas cadenas utilizando el algoritmo de Myers [19], obteniendo los vectores de bits que representan la matriz de programación dinámica. Posteriormente, se procede a calcular la ruta de alineación utilizando los vectores de bits en un recorrido de la matriz hacia atrás. Finalmente, a partir de los resultados de la ali-

neación se construye el archivo en formato SAM, que representa la salida del alineador y contiene toda la información de la alineación de la lectura corta.

El módulo de siembra

El módulo de siembra obtiene las semillas a partir de cada lectura corta y posteriormente los segmentos de la referencia en las regiones de siembra donde las semillas se han alineado. ABPSE utiliza semillas con máxima coincidencia exacta (SMEM, del inglés *Super Maximal Exact Match*), pues proveen mayor exactitud en la alineación y utilizan menos tiempo de cómputo que otras semillas, como las de longitud fija [21]. En particular, la longitud de una semilla SMEM puede ser tan grande como la lectura corta, de tal manera que se puede determinar inmediatamente cuándo una lectura corta se alinea exactamente a la cadena de referencia sin necesidad de realizar la etapa de extensión.

Para asegurar hallar al menos una semilla que se alinee en forma exacta a la cadena de referencia, la lectura, al inicio, se divide en (n+1) regiones de longitud fija como en la Figura 3, basado en el principio del palomar [22]. También se estableció la longitud mínima de la semilla igual al tamaño de estas regiones, como lo indica la Ecuación 1.

$$\text{L\_mínima\_semilla} = \frac{\text{longitud de lectura corta}}{(n+1)} \qquad (1)$$

La búsqueda de semillas inicia en uno de los extremos de la lectura, intentando encontrar intervalos de alineación exacta mayores a la longitud de semilla mínima. Por ejemplo, si la búsqueda termina dentro de la primera región, la semilla se descarta al no cumplir con el tamaño mínimo (Figura 3a), reanudando una nueva búsqueda en el carácter inmediato a su izquierda. Si la búsqueda, por el contrario, termina en la región dos, se ha encontrado una semilla candidata, y ésta se almacena y se procede a reiniciar una nueva búsqueda a partir del inicio de la región 2 (Figura 3b).

Por otra parte, tomando en cuenta que mientras el valor de n aumenta, el tamaño de la semilla disminuye y en consecuencia se provocan alineaciones falsas en múltiples regiones de la referencia, ABPSE permite valores de n entre 0 y 3, y lecturas cortas con longitud mínima de 60 nucleótidos.

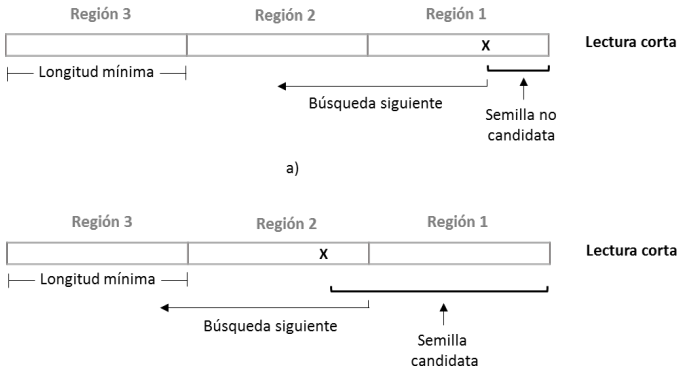

FIGURA 3. Búsqueda de semillas.

Para implementar la etapa de siembra se utiliza el algoritmo de búsqueda hacia atrás basado en los índices de FM. Estos índices son una estructura de datos conformada por la Transformada de Burrows-Wheeler (TBW) de la cadena de referencia, el Arreglo de Sufijos (AS) que contiene todas las posiciones de inicio de los sufijos en la referencia, una Matriz de Ocurrencias (Occ) de los cuatro caracteres del alfabeto Σ = {A, C, G, T} en la TBW y un vector de frecuencia (C). Dichos índices proporcionan una estructura eficiente en espacio para realizar búsquedas exactas de cadenas.

El Algoritmo 1 adaptada de la referencia [18], realiza la búsqueda exacta de cadenas. Las variables k y l representan el intervalo donde aparece el sufijo buscado, P representa la cadena patrón a buscar, i es un apuntador y σ el carácter del patrón que se está procesando actualmente. Las líneas del 5 al 10 representan el núcleo del proceso de búsqueda, básicamente, el ciclo toma cada carácter del patrón una por una y va encontrando los intervalos donde aparece el sufijo que se va formando. Finalmente, el algoritmo retorna el intervalo [k, l] si $k \leq l$ o un intervalo vacío en caso contrario (líneas 11 al 15).

ALGORITMO 1. Búsqueda exacta de cadenas.

```
1   function EXACTMATCH (R, P, C, Occ)
2     i = |P|-1
3     k = 0
4     l = |R|- 1
5     while k ≤ l && i ≥ 0 do
6       σ = P [i]
7       k = C[σ] + Occ[σ,k-1 ] + 1
8       l = C[σ] + Occ[σ,l ]
9       i = i-1
10    end while
11    if k ≤ l then
12      return {k,l}
13    else
14      return {Φ}
15    end if
16  end function
```

Por cada semilla obtenida mediante el algoritmo 1 se almacenan 3 datos importantes: la posición de inicio de la semilla en la lectura corta, el tamaño de la semilla y los valores del intervalo (*k* y *l*). La posición de inicio y el tamaño de la semilla, permiten saber con exactitud a partir de dónde debe extenderse la semilla hacia ambos lados, mientras que el intervalo determina las (*l*-*k*+1) regiones donde la semilla se ha alineado, utilizándose como índices del arreglo de sufijos (AS). En específico, el valor del arreglo de sufijos es utilizado para calcular la posición de inicio y fin del segmento de la referencia que será utilizado en la etapa de extensión, tal como se muestra en la Figura 4.

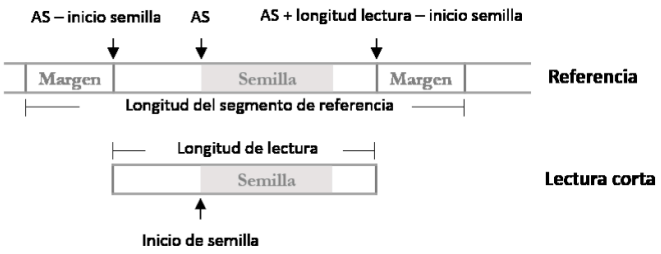

FIGURA 4. Cálculo del segmento de la referencia a utilizar en la etapa de extensión.

Es posible que durante la alineación se realicen operaciones de eliminación de nucleótidos en la lectura corta, lo cual significa que hay inserciones de caracteres en la cadena de referencia y por consiguiente la región requiere un espacio mayor al tamaño de la lectura corta. Es por esta razón que se agregan márgenes en cada extremo de la región previamente encontrada usando un valor de margen igual al número de errores permitidos para la alineación.

El módulo de extensión

La etapa de extensión se implementa utilizando el algoritmo de Myers, el cual se basa en el cálculo de la distancia de edición de Levenshtein, definida como el número mínimo de operaciones de inserción, eliminación o sustitución, necesarias para transformar una cadena a otra. La manera formal de calcular la distancia de edición de Levenshtein es mediante el algoritmo de Programación Dinámica (PD) que utiliza las fórmulas recursivas de la Ecuación (2).

$$C[i,j]=\min\begin{cases} C[i-1,j-1]+\delta_{i,j} \\ C[i-1,j]+1 \\ C[i,j-1]+1 \end{cases} \qquad (2)$$

$$\text{Con:} \quad \delta_{i,j}=\begin{cases} 0, & \text{si } p_i=t_j \\ 1, & \text{en otro caso} \end{cases}$$

Donde C[i,j] contiene la mínima distancia de edición entre el segmento del patrón $P_1,P_2...P_i$ y todos los posibles sufijos del texto T_1, T_2,...Tj. Por ejemplo, dadas dos cadenas similares T= "ATCATGAA" y P= "TCAGT", entonces, la matriz de distancia de edición que representa el cálculo de la similitud entre las dos cadenas es la que se muestra en la Figura 5.

		A	T	C	A	T	G	A	A
	0	0	0	0	0	0	0	0	0
T	1	1	0	1	1	0	1	1	1
C	2	2	1	0	1	1	1	2	2
A	3	2	2	1	0	1	2	1	2
G	4	3	3	2	1	1	1	2	2
T	5	4	3	3	2	1	2	2	3

FIGURA 5. Matriz de distancia de edición entre las cadenas T = "ATCATGAA" y P = "TCAGT".

En tal caso, examinando la última fila de la matriz puede notarse que las subcadenas de T que terminan en la posiciones 4, 6 y 7 están a solo dos transformaciones del patrón P.

Puede notarse que cada celda de la matriz difiere de las celdas vecinas únicamente por 3 valores (1, 0, -1). Mediante esta observación, Myers recodificó la matriz de PD representando solo las diferencias entre las celdas vecinas en cada fila y columna sucesiva, utilizando las fórmulas en las Ecuaciones (3) y (4). Así se obtienen las matrices de deltas mostradas en la Figura 6.

$$\Delta v[i,j]=C[i,j]-C[i-1,j] \quad (3)$$

$$\Delta h[i,j]=C[i,j]-C[i,j-1] \quad (4)$$

		A	T	C	A	T	G	A	A
	0	0	0	0	0	0	0	0	0
T	1	1	0	1	1	0	1	1	1
C	1	1	1	-1	0	1	0	1	1
A	1	0	1	1	-1	0	1	-1	0
G	1	1	1	1	1	0	-1	1	0
T	1	1	0	1	1	0	1	0	1

a)

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

b)

FIGURA 6. Representación de la matriz de programación dinámica en deltas. a) Delta vertical Δv. b) Delta horizontal Δh.

Posteriormente, cada columna se almacena mediante dos vectores de bits, VP y VN para la matriz de deltas verticales y, HP y HN para la matriz de deltas horizontales como se muestra en la Figura 7. Cada posición de estos vectores almacenan un 1 cuando se cumplen sus igualdades de acuerdo a las Ecuaciones (5), (6), (7), y (8), en caso contrario almacenan un 0, donde la notación $W_{i,j}$ indica el bit de la i-ésima posición en el entero W de la j-ésima columna. De esta manera, una columna de la matriz original se ha representado en 4 vectores de bits, lo cual reduce el espacio utilizado para calcular la distancia de edición.

$$VP_{i,j}=1 \leftrightarrow \Delta v[i,j]= +1 \quad (5)$$

$$VN_{i,j}=1 \leftrightarrow \Delta v[i,j]= -1 \quad (6)$$

$$HP_{i,j}=1 \leftrightarrow \Delta h[i,j]= +1 \quad (7)$$

$$HN_{i,j}=1 \leftrightarrow \Delta h[i,j]= -1 \quad (8)$$

	A	T	C	A	T	G	A	A
T	1	0	1	1	0	1	1	1
C	1	1	0	0	1	0	1	1
A	0	1	1	0	0	1	0	0
G	1	1	1	1	0	0	1	0
T	1	0	1	1	0	1	0	1

a)

	A	T	C	A	T	G	A	A
T	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0
A	0	0	0	1	0	0	1	0
G	0	0	0	0	0	1	0	0
T	0	0	0	0	0	0	0	0

b)

	A	T	C	A	T	G	A	A
T	0	0	1	0	0	1	0	0
C	0	0	0	1	0	0	1	0
A	0	0	0	0	1	1	0	1
G	0	0	0	0	0	0	1	0
T	0	0	0	0	0	1	0	1

c)

	A	T	C	A	T	G	A	A
T	0	1	0	0	1	0	0	0
C	0	1	1	0	0	0	0	0
A	1	0	1	1	0	0	1	0
G	1	0	1	1	0	0	0	0
T	1	1	0	1	1	0	0	0

d)

FIGURA 7. Representación en vectores de bits de la matriz de programación dinámica. a) VP, b) VN, c) HP y d) HN.

Myers demostró que estos vectores pueden obtenerse recursivamente mediante las Ecuaciones (9-14), donde Xv y Xh son vectores auxiliares, y Eq es un vector que codifica la igualdad de caracteres. El análisis se basa en que una columna de la matriz de edición solo puede calcularse a partir de los valores de la columna previa.

$$Xv = Eq \text{ OR } VN_{in} \quad (9)$$

$$VP_{out} = HN_{in} \text{ OR NOT } (Xv \text{ OR } HP_{in}) \quad (10)$$

$$VN_{out} = HP_{in} \text{ AND } Xv \quad (11)$$

$$Xh = Eq \text{ OR } HN_{in} \quad (12)$$

$$HP_{out} = VN_{in} \text{ OR NOT } (Xv \text{ OR } VP_{in}) \quad (13)$$

$$HN_{out} = VP_{in} \text{ AND } Xh \quad (14)$$

El Algoritmo 2 representa el algoritmo de Myers al estilo de programación en C adaptado de [19]. Donde la variable score contiene el último valor de cada columna de la matriz de distancia de edición original, calculada a partir del previo valor de score y el vector HP.

ALGORITMO 2. El algoritmo de Myers.

```
1   Precomputo de Peq[c]
2   VP = 1^m
3   VN = 0^m
4   Score = m
5   for j=1,2,…,n
6       Eq = Peq[tj]
7       Xv = Eq | VN
8       Xh = (((Eq & VP) + VP) ^ VP) | Eq
9       HP = VN | ~(Xh | VP)
10      HN = VP & Xh
11      if HP & 10m-1 then
12         Score += 1
13      else if HN & 10m-1 then
14         Score -=1
15      end if
16      HP <<= 1
17      HN <<= 1
18      VP = HN | ~(Xv | HP)
19      VN =   HP & Xv
20      if Score ≤ k then
21         printf "Coincidencia en " . j
22      end if
23  end for
```

Cálculo de la trayectoria de alineación

El algoritmo previo obtiene únicamente la distancia de edición entre las cadenas, sin embargo, en esta aplicación se requiere de la trayectoria de alineación completa, por lo que fueron necesarias algunas modificaciones. En la Figura 8 se muestran todas las rutas posibles de alineación entre las cadenas P y T, donde las flechas de dirección indican la celda de la cual puede preceder la celda C[i,j] durante el cálculo de la matriz.

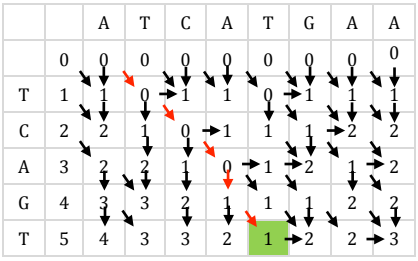

FIGURA 8. Ruta de alineación a partir de la distancia de edición mínima.

En el ejemplo, la mejor distancia de edición es 1, entonces, a partir de la casilla con este valor puede recorrerse la matriz hacia atrás y determinar la ruta de alineación; el proceso se realiza de derecha a izquierda y se apoya en flechas verticales, horizontales y diagonales. Finaliza cuando llega a una casilla de la primera fila de la matriz o se hayan recorrido todas las columnas. Una flecha en diagonal significa una coincidencia o sustitución de caracteres entre las dos cadenas; una vertical, una inserción respecto al patrón; y una horizontal la eliminación de un carácter en el patrón.

Como puede verse, las flechas horizontales y verticales coinciden perfectamente con los unos dentro de los vectores HP y VP de la Figura 7, por lo que pueden utilizarse en el recorrido hacia atrás. Sin embargo, no hay información sobre las diagonales, solo la del vector Eq, lo cual no es suficiente si los caracteres comparados no coinciden entre ellos. Para remediarlo, fue extendida la tabla presentada en el artículo original del algoritmo [19] con las posibles combinaciones de entrada, tal como se muestra en la Tabla 1, donde para clarificar el concepto se han incluido a la derecha ejemplos de combinaciones de entrada que justifican los valores de la salida D. A partir de ahí, pueden derivarse de forma inmediata el vector de la Ecuación 15, el cual representa los movimientos diagonales.

$$D = Eq \text{ or not } (VN_{in} \text{ or } HN_{in}) \quad (15)$$

El vector D, para el caso que se ha tratado como ejemplo en este artículo, se muestra en la Figura 9.

TABLA 1. Tabla de verdad para obtener el vector D.

No.	Δh_{in}	Δv_{in}	D	Δh_{out}	Δv_{out}	Vista de las 4 casillas								
1	-1	-1	1	1	1	1	0		1	1		1	2	1
2	0	-1	1	1	0	0	1		0	1		0	1	0
3	1	-1	1	1	-1									
4	-1	0	1	0	1	1	0		0	0		0	1	1
5	0	0	1	0	0	1	1		0	0		0	0	1
6	1	0	1	0	-1									
7	-1	1	1	-1	1	1	0		0	0		0	1	1
8	0	1	1	-1	0	2	1		1	0		1	0	2
9	1	1	1	-1	-1									
10	-1	-1	0	1	1	1	0		1	1		1	2	1
11	0	-1	0	1	0	0	1		0	1		0	1	0
12	1	-1	0	1	-1									
13	-1	0	0	0	1	1	0		0	0		0	1	1
14	0	0	1	1	1	1	1		0	1		0	1	1
15	1	0	1	1	0									
16	-1	1	0	-1	1	1	0		0	0		0	1	1
17	0	1	1	0	1	2	1		1	1		1	1	2
18	1	1	1	0	0									

	A	T	C	A	T	G	A	A
T	1	1	1	1	1	1	1	1
C	1	0	1	0	0	1	1	1
A	1	0	0	1	0	1	1	1
G	0	1	0	0	1	1	0	1
T	0	1	0	0	1	1	1	1

FIGURA 9. Direcciones en diagonal usando el vector D.

A partir de los vectores VP, HP y D, pueden obtenerse con facilidad todas las rutas de alineación. Por ejemplo, el Algoritmo 3 realiza el recorrido hacia atrás para obtener una posible ruta de alineación. Las entradas del algoritmo son los vectores D y VP que representan las direcciones en diagonal y vertical, *m* es la longitud de la lectura corta, *dmin* es la distancia mínima de edición y *col* la columna donde se encontró dicha distancia. La definición de las variables se realiza en las líneas 2 a 4, donde la variable *CIGAR* almacena todas las operaciones de la alineación, la variable *bit_actual* es una máscara auxiliar que permite identificar en que bit se encuentra el recorrido dentro de cada vector de bits e *i* es un apuntador de índice de la variable *CIGAR*.

El núcleo del programa se encuentra entre las líneas 5 y 17, donde se realiza el recorrido de la matriz hacia atrás, revisando los vectores D y VP hasta que ya no pueda desplazarse más el bit de la máscara auxiliar o cuando el contador de columna *col* sea igual a cero. La ruta de alineación o *CIGAR*, almacena la serie de operaciones que transforman la lectura al texto de referen-

ALGORITMO 3. Recorrido hacia atrás para obtener una ruta de alineación.

```
1   function RUTA_DE_ALINEACION (D,VP,m,
    dmin, col)
2     CIGAR [m + dmin + 1]
3     bit_actual = 1 << (m – 1)
4     i = 0
5     while (bit_actual && col >=0) do
6       if (D[col] & bit_actual) then
7           bit_actual >>= 1
8           col --
9           CIGAR[i++] = 'M'
10      else if (VP[col] & bit_actual) then
11          bit_actual >>= 1;
12          CIGAR[i++] = 'I'
13      Else
14         col--;
15         CIGAR[i++] = 'D'
16      end if
17    end while
18
19  while (bit_actual) do
20      CIGAR[i++] = 'I'
21      bit_actual >>= 1
22    end while
23
24    revertir(CIGAR,i)
25
26    return CIGAR
```

cia, una coincidencia o sustitución de nucleótidos se representa con la letra M, una inserción con la letra I y una eliminación con la letra D. En el proceso, es probable que la lectura corta deba alinearse más a la izquierda del segmento de referencia, lo cual sucede cuando la columna llega a cero antes que la máscara auxiliar, por lo que es necesario agregar operaciones de inserción en el *CIGAR*, lo anterior se realiza en las líneas 19 a 22. Finalmente, en la línea 24 se invierte la cadena resultante antes de ser retornada por la función. El *CIGAR* para el ejemplo que se ha tratado en este artículo se muestra en la Figura 10, donde puede verse claramente la alineación y las operaciones realizadas. Luego, el *CIGAR* se compacta según las especificaciones del formato SAM, obteniendo una versión corta del resultado; en este caso es "3M1I1M". Evidentemente, ambas operaciones pueden ser realizadas por la misma función, lo cual ocurre en el código real del alineador.

Texto	A	T	C	A		T	G	A	A
Cigar		M	M	M	I	M			
Patrón		T	C	A	G	T			

FIGURA 10. El *CIGAR* de la alineación.

Extendiendo el algoritmo para lecturas con longitud mayor a *w*

La descripción anterior es válida cuando el tamaño de la lectura *m* es mayor al tamaño de la palabra *w* del procesador, la cual está limitada a 64 en los procesadores modernos. Para eliminar dicha limitante fue necesario el procesamiento a bloques como se muestra en la Figura 11.

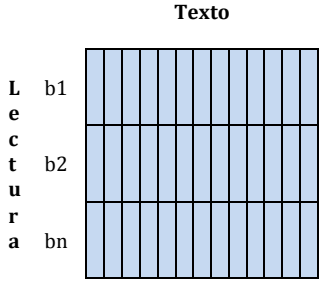

FIGURA 11. Modelo basado en bloques de vectores.

Por cada vector de bits en el algoritmo original, ahora se requieren $b = m/w$ vectores. El cálculo se realiza columna a columna, comenzando con el vector de bits del primer bloque y continuando en forma descendente, tomando en cuenta que, a diferencia de los vectores de bits en los bloques superiores de la matriz, las cuales contienen valores de 0 en el bit menos significativo, el resto de los bloques pueden contener valores de 0 o 1 debido al desplazamiento de bits de los bloques que los anteceden en la columna. Además, los vectores de bits en los bloques del último nivel pueden extenderse más allá del bit que corresponde a la longitud m de la cadena patrón, es decir $bitmax = w - m \pmod w$. Por lo que para calcular el valor de la distancia de edición es necesario un seguimiento preciso de dicho bit, lo cual se realiza mediante una máscara binaria apropiada.

RESULTADOS Y DISCUSIÓN

Para probar la funcionalidad y eficiencia del programa desarrollado se alinearon varios conjuntos de prueba, con un millón de lecturas cortas cada uno, y longitudes de 64, 100 y 128 nucleótidos, a los cromosomas 19, 20, 21 y 22 del genoma humano. Las lecturas cortas fueron generadas de forma artificial utilizando el programa wgsim con una razón de mutaciones de 0.4%, la cual representa el límite de variaciones en el caso del genoma humano [23], donde el 25% de esas mutaciones son indels (inserciones y eliminaciones) y el 70% de los indels son extendidos. El error de secuenciación fue configurado a 0.1%, valor típico en las máquinas de secuenciación actuales [1]. Todas las pruebas fueron realizadas en una computadora con procesador Intel Core i7 de Sexta generación, 16 GB en RAM y 1 TB en disco duro, con sistema operativo Ubuntu 14.04.

En la Tabla 2 se muestran los tiempos de alineación obtenidos, donde puede observarse que incluso los conjuntos de lecturas cortas de 128 nucleótidos, requieren menos de 150 segundos de procesamiento. Lo cual se debe principalmente al paralelismo a nivel

de bit del algoritmo utilizado en la etapa de extensión. También puede observarse una dependencia directa del tiempo de alineación con respecto a la longitud de la lectura, y que los tiempos de alineación permanecen prácticamente constantes aun cuando se incrementa la longitud de las cadenas de referencia.

TABLA 2. Tiempos de ejecución de ABPSE.

Referencia	Longitud de lectura (nts)		
	64	100	128
Chr21	97,61 s.	111,16 s.	140,35 s.
Chr22	106,11 s.	113,87 s.	137,21 s.
Chr19	101,20 s.	131,41 s.	145,19 s.
Chr20	107,33 s.	120,88 s.	143,42 s.

Esto último es muy apropiado, ya que permite la alineación de lecturas a cadenas de referencia muy grandes sin tiempos extras de cómputo, y se debe a la complejidad temporal del algoritmo usado en la etapa de siembra (búsqueda basada en índices de FM), la cual es una función de la longitud de las lecturas cortas y no de la longitud de la cadena de referencia.

TABLA 3. Exactitud de alineación con ABPSE.

Referencia	Tipo de resultado	Longitud de lectura (nts)		
		64	100	128
Chr21	Alineadas	986558	976349	967040
	No alineadas	13442	23651	32960
	Correctas	976101	970002	961044
	Incorrectas	10457	6347	5996
Chr22	Alineadas	985954	976409	967229
	No alineadas	14046	23591	32771
	Correctas	956513	955025	947981
	Incorrectas	29441	21384	19248
Chr19	Alineadas	985117	975625	967397
	No alineadas	14883	24375	32603
	Correctas	962164	963039	956562
	Incorrectas	22953	12586	10835
Chr20	Alineadas	986079	976309	967061
	No alineadas	13921	23691	32939
	Correctas	972866	968010	959421
	Incorrectas	13213	8299	7640

Adicionalmente, se determinó la cantidad de lecturas que se alinearon correcta e incorrectamente a su origen. Esto fue posible debido a que el generador de lecturas artificiales, wgsim, proporciona la ubicación exacta de donde es extraída cada una de sus lecturas. De esta manera, mediante una función de comparación se validan los resultados de ABPSE. En esta prueba, se consideran correctas aquellas alineaciones en un intervalo de ±3 nucleótidos de su posición original, considerando la posibilidad de inserciones o eliminaciones en relación al genoma de referencia. Los resultados se muestran en la Tabla 3, donde puede notarse un porcentaje promedio de 96.2% de alineaciones correctas, 1.39% de alineaciones incorrectas y 2.41% de lecturas no alineadas. Las alineaciones incorrectas son provocadas principalmente por regiones repetidas, lo cual puede mejorarse posteriormente al utilizar lecturas apareadas.

Finalmente, se compararon los tiempos de ejecución del alineador propuesto y el de los populares programas de alineación BWA-SW y BWA-MEM. Ambos programas utilizan la estrategia siembra y extiende, y los índices de FM para la etapa de siembra. En la extensión BWA-SW utiliza el algoritmo de Smith-Waterman mientras que BWA-MEM utiliza una combinación de alineación local y global. La comparación fue realizada utilizando un solo núcleo del procesador en cada uno de los programas. La Figura 12 muestra los resultados al alinear conjuntos de 1 millón de lecturas al genoma humano completo.

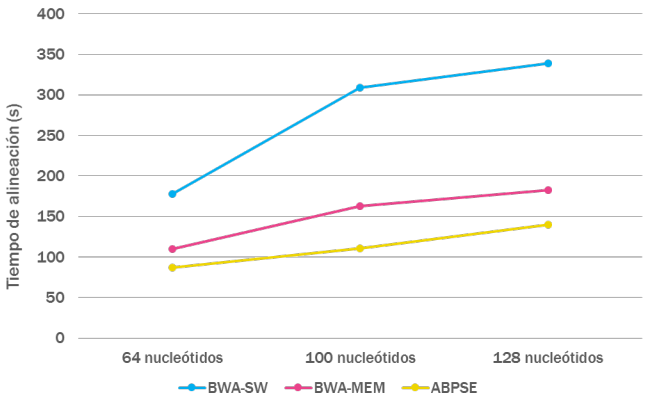

FIGURA 12. Comparación de los tiempos de alineación de ABPSE con los alineadores BWA-SW y BWA-MEM.

Puede observarse como el alineador propuesto proporciona un factor de aceleración superior a 2.45x respecto al programa BWA-SW y 1.36x respecto al alineador BWA-MEM, ambos programas han sido catalogados entre los más rápidos y exactos alineadores de ADN en la actualidad [24-25]. Estas mejoras en la velocidad son muy significantes, puesto que en un proceso normal de alineación, los billones de lecturas que deben alinearse implican días completos de procesamiento, lo cual con ABPSE podría realizarse en solo unas horas. Factores de alineación superiores solo se han reportado mediante aceleradores Hardware basados en unidades de procesamiento gráfico GPUs [26], o en FPGAs [27], aunque estas son excelentes opciones, su elevado costo suele ser una limitante. En esta prueba ABPSE utilizó en promedio solo 7.6 GB de memoria RAM, lo cual permite su ejecución en cualquier computadora convencional.

El número promedio de lecturas mapeadas correcta e incorrectamente, también fue contrastado en esta prueba, obteniendo los resultados de la Figura 13.

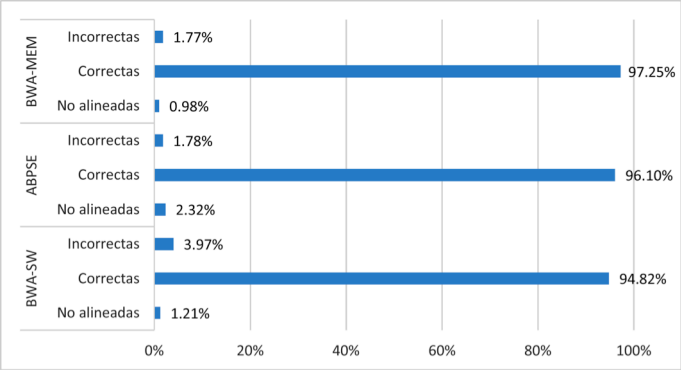

FIGURA 13. Comparación de la eficiencia de ABPSEcon los alineadores BWA-SW y BWA-MEM.

Puede observarse una eficiencia comparable entre los tres programas. En particular, al calcular el error de alineación definida mediante la Ecuación 16, se obtienen valores de 0.0401, 0.0181 y 0.0178 para BWA-SW, ABPSE Y BWA-MEN respectivamente.

$$Error = \frac{No.\ de\ Lecturas\ alineadas\ incorrectamente}{No.\ de\ Lecturas\ alineadas} \qquad (16)$$

Lo anterior refleja una eficiencia muy cercana entre BWA-MEM y ABPSE, siendo mayor a la de BWA-SW. La ligera desventaja en relación a BWA-MEN puede superarse en versiones posteriores al refinar la técnica de sembrado.

CONCLUSIONES

En este artículo se presentó el desarrollo de un programa de alineación de lecturas cortas de ADN que implementa la estrategia siembra y extiende, utilizando la combinación de dos algoritmos muy eficientes. En la etapa de siembra se utilizó el algoritmo de búsqueda hacia atrás basado en los índices de FM que permite realizar búsquedas exactas de cadenas de forma muy rápida independientemente de la longitud de la cadena de referencia. Para la etapa de extensión se implementó el algoritmo de programación dinámica de Myers que calcula la distancia de edición entre dos cadenas. Este último algoritmo fue extendido para obtener la trayectoria de alineación de las cadenas utilizando vectores de bits. Mediante las pruebas realizadas se demuestra que la combinación de dichos algoritmos permite el desarrollo de alineadores de alta velocidad, gracias al paralelismo a nivel de bit en la etapa de extensión. A pesar de que la exactitud del alineador es muy bueno, ésta podría mejorarse si se exploran otros tipos de semillas en la etapa de siembra. Por otra parte, la etapa de extensión podría ser implementada directamente en hardware, donde el tamaño del entero no es un factor limitante, evitando de esta manera la programación a bloques y en consecuencia permitiendo la aceleración al máximo del programa, así como la alineación de lecturas mucho más largas.

REFERENCIAS

[1] Goodwin S, McPherson JD, McCombie WR. Coming of age: ten years of next-generation sequencing technologies. Nat. rev. genet. 2016; 17(6): 333-351. DOI: 10.1038/nrg.2016.49

[2] Yohe S, Thyagarajan B. Review of Clinical Next-Generation Sequencing. Arch. Pathol. Lab. Med. 2017; 141(11): 1544-1557. DOI: 10.5858/arpa.2016-0501-RA

[3] Pacheco-Bautista D, González-Perez M, Algredo-Badillo I. De la secuenciación a la aceleración hardware de los programas de alineación de ADN, una revisión integral. Rev. mex. ing. bioméd. 2015; 36(3):259-277. DOI: 10.17488/RMIB.36.3.6

[4] Liu Y, Tran TT, Lauenroth F, Schmidt B. Smith-Waterman algorithm on Xeon Phi coprocessors for long DNA sequences. In Cluster Computing (CLUSTER), 2014 IEEE International Conference on; pp. 257-265. IEEE; 2014. DOI: 10.1109/CLUSTER.2014.6968772

[5] Salavert J, Tomás A, Tárraga J, Medina I, Dopazo J, Blanquer I. Fast inexact mapping using advanced tree exploration on backward search methods. BMC Bioinformatics. 2015; 16 (1): 18. DOI: 10.1186/s12859-014-0438-3

[6] Langmead B, Trapnell C, Pop M, Salzberg S. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology. 2009; 10(3). DOI: 10.1186/gb-2009-10-3-r25

[7] Drucker TM, Johnson SH, Murphy SJ, Cradic KW, Therneau TM, Vasmatzis G. BIMA V3: an aligner customized for mate pair library sequencing. Bioinformatics. 2014; 30(11): 1627-1629. DOI: 10.1093/bioinformatics/btu078

[8] Agrawal A, Huang X. Pairwise statistical significance of local sequence alignment using sequence-specific and position-specific substitution matrices. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB). 2011; 8(1): 194-205. DOI: 10.1109/TCBB.2009.69

[9] Borrayo E, Mendizabal-Ruiz EG, Vélez-Pérez H, Romo-Vázquez R, Mendizabal AP, Morales JA. Genomic signal processing methods for computation of alignment-free distances from DNA sequences. PloS one, 2014; 9(11): e110954. DOI: 10.1371/journal.pone.0110954

[10] Li H DR. Fast and accurate long-read alignment with Burrows-Wheeler trasform. Bioinformatics. 2010; 26(5): 589-595. DOI: 10.1093/bioinformatics/btp698

[11] Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. preprint. 2013;: p. arXiv:1303.3997. [Online]. Available: *hup://arxiv.orglabs/1303.3997*

[12] Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nature methods. 2012; 9(4): 357-359. DOI: 10.1038/nmeth.1923

[13] Liu Y, Schmidt B, Maskell DL. Cushaw: a cuda compatible short read aligner to large genomes based on the burrows-wheeler transform. Bioinformatics. 2012; 28(14): 1830-1835. DOI: 10.1093/bioinformatics/bts276

[14] Wu TD. Bitpacking techniques for indexing genomes: I. Hash tables. 2016; 11(5): p. 1748-7188. DOI: 10.1186/s13015-016-0069-5

[15] Burrows M, Wheeler DJ. A block sorting lossless data compression algorithm. Reporte Técnico. Palo Alto, California: Digital Equipment Corporation, Systems Research Center; 1994. Report Number: 124.

[16] Smith TF, Waterman MS. Identification of common molecular subsequences. J. Mol. Biol. 1981; 14(1): 195-197. DOI: 10.1016/0022-2836(81)90087-5

[17] Needleman N, Wunsch C. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of two Proteins. Journal of Molecular. 1970; 48(3): 443-453. DOI: 10.1016/0022-2836(70)90057-4

[18] Ferragina P, Manzini G. Opportunistic data structures with applications. In Foundations of computer science; 2000; Redondo Beach, CA: IEEE. 390-398. DOI: 10.1109/SFCS.2000.892127

[19] Myers G. A fast bit-vector algorithm for approximate string matching based on dynamic programming. Journal of the ACM (JACM). 1999; 46(3): 395-415. DOI: 10.1145/316542.316550

[20] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009; 25(16): 2078-2079. DOI: 10.1093/bioinformatics/btp352

[21] Ahmed N, Bertels K, Al-Ars Z. A comparison of seed-and-extend techniques in modern DNA read alignment algorithms. In Bioinformatics and Biomedicine (BIBM) 2016 IEEE International Conference on. 2016; p. 1421-1428. DOI: 10.1109/BIBM.2016.7822731

[22] Ahmadi A, Behm A, Honnalli N, Li C, Weng L, Xie XH. Optimized gram-based methods for efficient read alignment. Nucleic Acids Res. 2012; 40(6). DOI: 10.1093/nar/gkr1246

[23] Tishkoff SA, Kidd KK. Implications of biogeography of human populations for 'race' and medicine. Nature Genetics. 2004; 36(11): S21-S22. DOI: 10.1038/ng1438

[24] Fonseca NA, Rung J, Brazma A, Marioni JC. Tools for mapping high-throughput sequencing data. Bioinformatics. 2012; 28(24): 3169-3177. DOI: 10.1093/bioinformatics/bts605

[25] Policriti A, Prezza N. Fast randomized approximate string matching with succinct hash data structures. BMC bioinformatics. 2015; 16(9): S4. DOI: 10.1186/1471-2105-16-S9-S4

[26] Hung CL, Lin YS, Lin CY, Chung YC, Chung YF.. CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs. Computational biology and chemistry. 2015; 58: 62-68. DOI: 10.1016/j.compbiolchem.2015.05.004

[27] Pacheco-Bautista D., Carreño-Aguilera, R., Cortés-Pérez E, Gónzalez-Pérez, M, Medel JJ., Acevedo MA, YU W. Nonlinear FM index application for alignment of short DNA sequences using re-parametrization of algorithms. Fractals. 2018; 26(3): 1850023. DOI: 10.1142/S0218348X18500238