Whole genome alignment in high performance computing environments

Informe del progreso de trabajo de investigación

Julio César García Vizcaíno
Departamento de Arquitectura de Computadores y Sistemas Operativos
Universidad Autónoma de Barcelona
jcgarcia@aomail.uab.es

20 de junio de 2012

Datos del doctorando

Nombre Completo: Julio Csar García Vizcaíno
NIE: Y1497678R

Datos del proyecto de tesis

Estudio de doctorado: Computación de Altas Prestaciones

Título del proyecto de tesis doctoral: *Whole genome alignment in high performance computing environments.*

Directores de la tesis: Antonio Espinosa, Juan Carlos Moure.

Línea de investigación: Aplicaciones bioinformáticas.

Rgimen de dedicación: completa.

Año de inscripción: 2011.

Año previsto para la presentación de Tesis Doctoral: 2014

Firmado

Director	Director	Estudiante
Antonio Espinos	Juan Carlos Moure	Julio César García Vizcaíno

1. Resumen

Actualmente la generación de nueva información genómica ha creado la necesidad de almacenar y procesar estos nuevos datos computacionalmente hablando. Una de las tareas en la que nos centramos es en el alineamiento de genomas. Los algoritmos utilizados para el alineamiento de genomas presentan una alta demanda computacional debido al tamaño de los datos genómicos.

Existe una gran cantidad de algoritmos propuestos para el alineamiento de dos secuencias como [Needleman and Wunsch, 1970] y [Smith and Waterman, 1981]. Estos algoritmos funcionan bien con tamaños de secuencias pequeños como un gen. Sin embargo, son ineficaces al alinear genomas enteros. La complejidad computacional y espacial es $O(nm)$, donde n y m son las longitudes de los genomas a alinear. Estos algoritmos tienen problemas de mayores requerimientos de memoria o de tiempos de ejecución inaceptables.

Para grandes secuencias, como el genoma humano (3Gbp), el tiempo de ejecución puede ser extremadamente grande. Es por ello que han surgido alternativas para reducir el tiempo de ejecución del alineamiento de secuencias. Una de esas alternativas es la utilización de la heurística basada en las coincidencias exactas máximas en las secuencias a alinear.

El árbol de sufijos es una estructura de datos que permite la búsqueda de coincidencias exactas de longitud variable en tiempo lineal. Sin embargo los requerimientos de espacio necesario para almacenar el árbol de sufijos en memoria principal la hacen una estructura inviable cuando el árbol de sufijos a almacenar supera la memoria disponible.

Este problema se presenta en aplicaciones de alineamiento de genomas donde el tamaño de los genomas son de millones de caracteres y su representación en un árbol de sufijos se hace inviable si no se tiene la cantidad de memoria disponible. Adicionalmente, la razón de utilizar un árbol de sufijos radica en que la búsqueda de coincidencias exactas de longitud variable en un árbol de sufijos se realiza en tiempo lineal.

En consecuencia se busca **acelerar la búsqueda eficiente de coincidencias exactas y que tenga en cuenta el uso de recursos de cómputo y memoria**. Las consideraciones que se toman en cuenta son el tamaño de la cadena de referencia (sobre la que se realiza la búsqueda) y la longitud de la coincidencia a buscar.

2. Introducción

2.1. MUM y MEM

La heurística basada en la búsqueda de coincidencias exactas máximas para el alineamiento de genomas requiere de la definición formal de una coincidencia exacta máxima (MEM).

Definición 1 *Un MEM (Maximal Exact Match) es una subcadena común a dos genomas que es mayor que una longitud mínima específica* d *de tal manera que es máxima, esto es, que no puede ser extendida en ambos extremos sin incurrir en una discrepancia.*

Adicionalmente, a partir de la definición de MEM surge el concepto de la coincidencia única maxima (MUM).

Definición 2 *Un MUM (Maximal Unique Match) es una subcadena única común a dos genomas que es mayor que una longitud mínima específica* d *de tal manera que es máxima, esto es, que no puede ser extendida en ambos extremos sin incurrir en una discrepancia.*

Identificar las cadenas, k, mas grandes en el genoma Q que tienen una coincidencia de longitud l idéntica en el genoma R, tiene la siguiente complejidad computacional:

- Método simple: $O(nl)$
- Usando árbol de sufijos: $O(l + k)$

El objetivo es buscar todas aquellas subcadenas que son comunes a las secuencias de referencia R y consulta Q que son máximas de longitud.

2.2. Árbol de sufijos

Cualquier cadena de referencia de longitud n puede ser descompuesta en s sufijos, ver Figura 1, y estos sufijos pueden almacenarse en un árbol de sufijos. Para crear esta estructura de datos se requiere de un tiempo $O(n)$ y para buscar una cadena en él requiere de un tiempo $O(l)$ donde l es la longitud de la cadena [Gusfield, 2007]. Estas dos propiedades hacen al árbol de sufijos una estructura útil para un rango diverso de aplicaciones bioinformáticas, incluyendo: alineamiento de genomas [Kurtz et al., 2004].

El primer algoritmo para la construcción de un árbol de sufijos en tiempo lineal y eficiente en espacio fue propuesto en [McCreight, 1976] , y Ukkonen produjo una variante “on-line” de ese algoritmo en [Ukkonen, 1992]. La clave para la búsqueda veloz en un árbol de sufijos es que hay un camino desde la raíz para cada sufijo del texto. Esto significa que se necesitan l comparaciones para encontrar

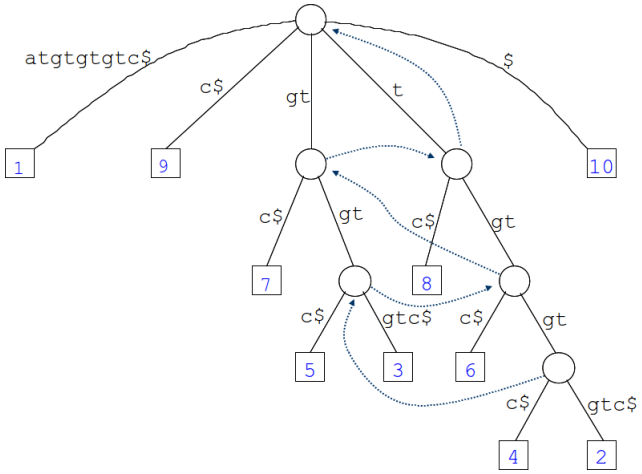

Figura 1: Árbol de sufijos para la palabra atgtgtgtc$.

una cadena de longitud l.

Otra mejora de implementación necesario para conseguir un tiempo y espacio lineal es el uso de enlaces de sufijos. Un enlace de sufijo es un puntero de un nodo interno etiquetado xS a otro nodo interno etiquetado S, donde x es un carácter arbitrario y S es posiblemente una subcadena vacía. Los enlaces de sufijos permiten agilizar el recorrido del árbol sin visitar la raíz del árbol debido a que apuntan a la siguiente extensión del sufijo. Los enlaces de sufijos y la compresión de las etiquetas de las aristas son los requerimientos claves para la implementación del árbol de sufijos en $O(n)$.

Existen diferentes implementaciones de árboles de sufijos, cada uno con complejidades espaciales diferentes como se muestra en el Cuadro 1.

Algoritmo	Complejidad espacial	Tamaño
McCreight	$O(28n)$	81.48GB
Ukkonen	$O(26{,}4n)$	76.95GB
Kurtz	$O(15{,}5n)$	45.31GB
Sadakane[1]	$O(n \log \lvert\Sigma\rvert)$	8.8GB
TRELLIS [Phoophakdee and Zaki, 2007]	$O(24{,}6n)$	71.6GB
TDD [Barsky et al., 2010]	$O(18{,}5n)$	54GB

Cuadro 1: Algoritmos de creación de árboles de sufijos y su uso de memoria al almacenar el genoma humano (n=2.91Gbp).

Los árboles de sufijos son dependientes del alfabeto, Σ, el tamaño del alfabeto,

$|\Sigma|$, afecta los tiempos de creación y búsqueda de cadenas en el árbol, C, para una cadena de referencia de longitud n. Estrictamente hablando:

- Requerimiento de espacio: $O(n)$
- Tiempo de creación: $min(O(n \log n), O(n \log |\Sigma|))$
- Tiempo de búsqueda: $min(O(|C| \log n), O(|C| \log |\Sigma|))$

2.3. Definición del problema a resolver

El problema de la búsqueda de las coincidencias exactas máximas de una longitud mínima han sido identificados en varias aplicaciones, entre ellas MUMmer [Kurtz et al., 2004]. Aunque el algoritmo de MUMmer realiza la búsqueda de coincidencias exactas máximas rápidamente, los requerimientos de recursos de cómputo aún son altos debido al tamaño de los datos de entrada.

Se ha realizado un experimento para comparar las prestaciones de la búsqueda de coincidencias exactas máximas utilizando un árbol de sufijos. En el Cuadro 2 se muestra el tiempo de procesamiento de búsqueda para coincidencias de tamaño mínimo L, utilizando el árbol de sufijos. El tiempo de cómputo invertido en la búsqueda involucra la operación individual de cadenas de longitud mínima L hasta la longitud de la cadena máxima de coincidencia.

El tiempo de búsqueda del Cuadro 2 para cadenas de tamaño de 30Mbp es

Estructura	L [bp]	Cantidad de búsquedas	Búsqueda [s]	Uso de memoria [MB]
Árbol de sufijos	20	$344{,}46 \times 10^6$	1679,2	483

Cuadro 2: Búsqueda coincidencias exactas para una cadena de referencia de 29,38Mbp y una cadena de consulta de 29,69Mbp

relativamente alto debido a que involucra realizar millones de búsquedas en serie. La cantidad de operaciones de búsqueda que se realizan son 11 veces la longitud de las cadenas de referencia y consulta. En consecuencia, la búsqueda de coincidencias exactas máximas es una operación computacionalmente intensiva. La razón de este uso intensivo de cómputo se debe a la cantidad de operaciones de búsqueda.

Al aumentar la longitud de las cadenas de referencia y consulta por un factor de 10, los tiempos de búsqueda se incrementan linealmente por un factor de 10, el uso de memoria aumenta aproximadamente 8 veces y la cantidad de búsquedas a realizar se multiplica $\times 10^3$ veces, como se observa en el Cuadro 3.

Si la longitud de las cadenas crece por un factor de 10, ver Cuadro 4, la can-

Estructura	L [bp]	Cantidad de búsquedas	Búsqueda [s]	Uso de memoria [MB]
Árbol de sufijos	20	$3{,}78\times10^{9}$	13013,5	3890

Cuadro 3: Búsqueda coincidencias exactas para una cadena de referencia de 227,69Mbp y una cadena de consulta de 238,62Mbp

tidad de búsquedas son aproximadamente $\times10^{18}$, el tiempo de búsqueda crece linealmente y el uso de memoria se convierte en un problema a analizar si se considera la disponibilidad finita de memoria en la mayoría de sistemas de cómputo personales.

De acuerdo a los datos de los Cuadros 2, 3 y 4 se hace necesario la búsqueda

Estructura	L [bp]	Cantidad de búsquedas	Búsqueda [s]	Uso de memoria [MB]
Árbol de sufijos	20	$9{,}87\times10^{18}$	169189,4	48665,12

Cuadro 4: Búsqueda coincidencias exactas para una cadena de referencia de 2960,21Mbp y una cadena de consulta de 2716,96Mbp

de alternativas que permitan llevar a cabo la búsqueda de manera eficiente. El uso de HPC nos permitiría resolver los problemas de escalabilidad (tiempo de búsqueda, uso de memoria) cuando se tienen cadenas muy grandes, como el genoma humano.

Problema

La búsqueda de coincidencias exactas máximas es un problema computacionalmente intensivo. Dadas una cadena de referencia R de longitud n y una cadena de consulta Q de longitud m, encontrar todas las coincidencias de longitud mínima l de Q en la cadena de referencia R implica recorrer la cadena de consulta Q para $todas$ sus subcadenas. Para cada subcadena se determina la posición o posiciones en donde existe una coincidencia, de longitud mínima l, entre la subcadena y la cadena de referencia R. La posición de la subcadena en Q es determinada.

En la Figura 2 se muestra gráficamente el problema de la búsqueda de coincidencias exactas máximas.

El total de búsquedas, indicado en los Cuadros 2 y 3 a realizar está determina-

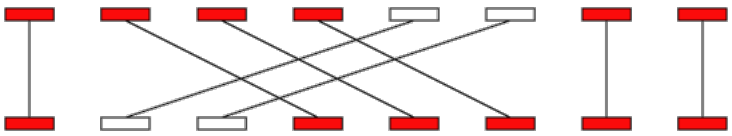

Figura 2: Problema de la búsqueda de coincidencias exactas máximas.

da por la fórmula 1.

$$\sum_{i=0}^{i \leq m} \frac{n}{L+i} \tag{1}$$

Realizar una búsqueda de coincidencia exacta de longitud fija es sencilla computacionalmente hablando. Pero al realizar una búsqueda de coincidencia exacta de longitud variable ocurren los siguientes problemas:

- Acceso aleatorio a la cadena de referencia R: el conjunto de coincidencias podría estar en posiciones de la cadena R distantes.
- Longitud variable de la cadena a buscar: la búsqueda a realizar se incrementa con la longitud de la cadena.
- Uso de memoria: para poder buscar una coincidencia se requiere volcar la cadena de referencia R en memoria. Dependiendo del método utilizado la cantidad de memoria se podría incrementar e incluso agotar la memoria disponible. Adicionalmente el tiempo de búsqueda se incrementará, como se observa en el Cuadro 4,debido al acceso a disco cuando no se tiene toda la cantidad de memoria necesaria.

Para resolver este problema de forma distribuida existen diferentes estrategias. La solución mas obvia es realizar una división de las cadenas de referencia y de consulta, ver Figura 3.
Esta técnica ocasiona que exista una fase de procesamiento posterior a la búsqueda de coincidencias exactas máximas en cada segmento de la cadena procesada, para determinar las coincidencias máximas de forma global. Adicionalmente, puede haber pérdidas de coincidencias máximas exactas y coincidencias máximas que solo son válidas dentro del segmento evaluado y no de manera global.
En consecuencia, para resolver este problema se utiliza alguna estructura de datos que permita realizar una búsqueda distribuida de coincidencias exactas máximas de forma ágil en la cadena entera. Los árboles de sufijos permiten hacer búsquedas con complejidad $O(m+k)$, donde m es el tamaño de la cadena a buscar y k es el número de ocurrencias.
En resumen la búsqueda de coincidencias exactas máximas en una cadena, como

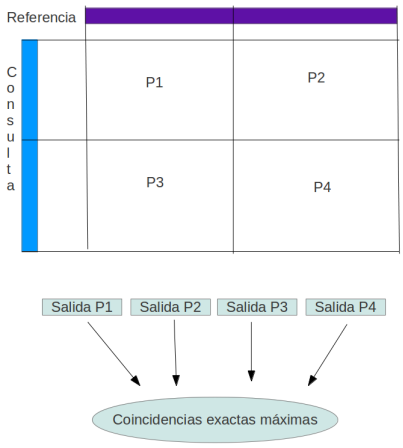

Figura 3: Solución obvia de paralelización de búsqueda de coincidencias exactas máximas.

el genoma humano, es un problema computacionalmente intensivo que es posible resolverse con estructuras de datos como el árbol de sufijos.
Independientemente de la estructura de datos utilizada, lo que se desea es poder realizar búsquedas de coincidencias exactas a esa estructura de manera eficiente. Una de las consultas en que se requiere realizar rápidamente, para el alineamiento de genomas, es la búsqueda exacta de coincidencias máximas.

3. Estado de la investigación

3.1. Búsqueda distribuida de coincidencias exactas: árbol de sufijos distribuido

El problema de mejorar el tiempo de cómputo de las búsquedas de coincidencias exactas nos da una oportunidad de utilizar el poder del HPC para reducir el tiempo de procesamiento. Se propone modificar la estructura de datos utilizada en MUMmer (árbol de sufijos) para mejorar las búsquedas de coincidencias exactas en grandes volúmenes de datos, considerando un uso eficiente de los recursos. Para ello se tomarán en cuenta los siguientes parámetros:

- Tamaño del genoma de referencia y consulta.
- Memoria principal disponible en cada nodo de cómputo.

Basado en dichos parámetros se distribuirá el árbol de sufijos en subárboles, ver Figura 4. Dicha aproximación ha sido llevada a cabo de distintas maneras por

[Mansour, 2012], [Japp, 2004], [Ghoting and Makarychev, 2010] y [Sadakane,]. Los subárboles aprovecharán una característica de la búsqueda de coincidencias: el acceso frecuente a la parte superior del árbol de sufijos.

Para definir formalmente el árbol de sufijos distribuido es necesario definir algu-

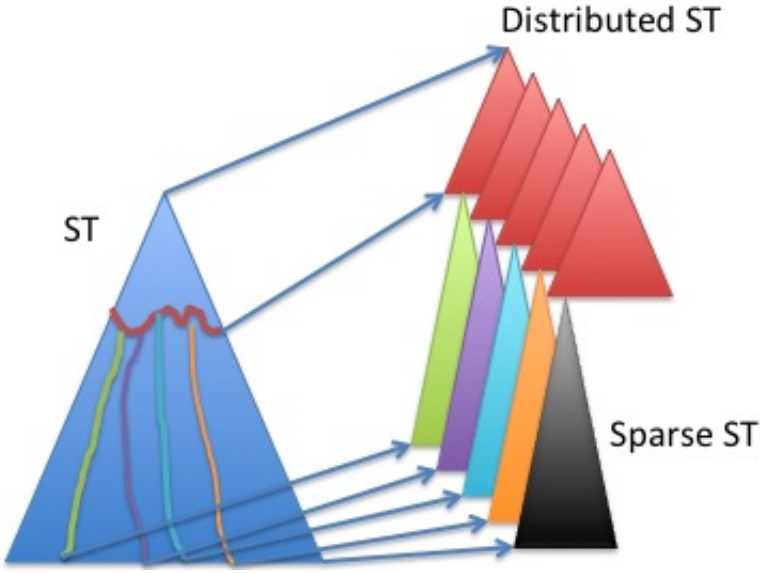

Figura 4: Árbol de sufijos distribuido.

nos conceptos, extraídos de [Clifford, 2005] y [Ghoting and Makarychev, 2010].

Definición 3 *Si $t = uvw$ entonces u es un prefijo y w es un sufijo de t.*

Un sufijo de t se repite si ocurre al menos una vez como una subcadena no sufijo de t. Además requerimos poder especificar que sufijos de t se incluyen en un subárbol de sufijos. Esto se hace considerando un prefijo corto z, y un conjunto de posiciones iniciales, V_z.

Definición 4 *Para aquellos sufijos de la referencia que tienen la cadena z como un prefijo.V_z es el conjunto válido.*

Definición 5 *Una subcadena s de t es válida si z es un prefijo de s y que un intervalo de enteros $I[i, j]$ es válido (con respecto a V_z y t) si $i \epsilon V_z$.*

Definición 6 *s es un sufijo válido para $I[i, j]$ si $s = t[k, j]$ para algún $k \epsilon V_z$ y $k > i$.*

Es posible que un sufijo válido para un intervalo puede ser mas corto que el prefijo fijo z, dependiendo de los caracteres que siguen directamente en la referencia.

Definición 7 *Sea z una cadena de longitud mayor o igual a 1. Decimos que (t, V_z) es un par de entrada si V_z es el conjunto válido (para t con respecto a z).*

Escribimos $sst(t, V_z)$ para el árbol de sufijos poco denso de t usando el conjunto válido V_z.

Definición 8 *Para un árbol de sufijos poco denso $T = sst(t, V_z)$ y un conjunto arbitrario de cadenas S, decimos que $T' = Taumentado$ por S si T' es una trie compactada de la unión de S y el conjunto sufijos válidos de t (con respecto a V_z).*

Para construir el árbol de sufijos distribuido DST en tiempo lineal se construye un SST en tiempo lineal y se ejecuta el algoritmo de construcción del árbol para diferentes prefijos escogidos. El algoritmo resultante usa el espacio en cada nodo que es proporcional al tamaño del SST construido.

3.2. Importancia del enlace de sufijo

Los enlaces de sufijos juegan un rol crítico en la construcción lineal de árboles de sufijos y en su recorrido.

Definición 9 *Si hay un nodo v en el árbol de sufijos con la etiqueta $c\alpha$, donde c es un carácter y $alpha$ es una cadena (no vacía), entonces el enlace de sufijo de v apunta a s(v), que es un nodo con la etiqueta α. Si α está vacío, entonces el enlace de sufijo de v, por ejemplo s(v) es la raíz.*

Los enlaces de sufijo existen para cada nodo interno (no hoja) de un árbol de sufijo. Siguiendo un enlace de sufijo, se puede saltar de un sufijo a otro, cada sufijo iniciando exactamente un carácter siguiente del primer carácter de su sufijo precedente. De tal manera que usando enlaces de sufijos es posible obtener un algoritmo lineal para la búsqueda de coincidencias exactas máximas. Esto es porque los enlaces de sufijos hacen un seguimiento de que o cuanto de la cadena ha coincidido hasta ahora. Cuando encontramos una falta de coincidencia, podemos saltar a lo largo del enlace de sufijo para ver si hay cualquier coincidencia después en la referencia y en la consulta.

Definición 10 *Considerar un par de entrada (t, V_z) y un árbol de sufijos poco denso $T = sst(t, V_z)$. Un enlace de sufijo disperso es un borde sin etiquetar de $\overline{aw}$ a la raíz si $|v| < |z|$ y de $\overline{aw}$ a $\overline{v}$, de otra manera.*

Mediante la organización del nuevo árbol de sufijos distribuido de esta manera, podemos realizar búsquedas de coincidencias exactas máximas de forma distribuida. Al utilizar una búsqueda distribuida es posible mejorar el tiempo de cómputo de las búsquedas exactas. Se requiere que el tiempo de cómputo sea menor que el mostrado por propuestas existentes en el estado del arte y que pueda manejar genomas de gran tamaño.

3.3. Evaluación experimental

Para determinar el par de entrada (t, V_z) es necesario determinar el acceso a los prefijos dentro del árbol de sufijos. Es por ello que se realizó un experimento para ubicar la longitud necesaria para la elección de los prefijos. En la siguiente figura 5 se muestra en la zona azul la parte superior del árbol de sufijos y las líneas rojas muestran el pintado del árbol para indicar la ubicación de una coincidencia máxima en el árbol.Para determinar el par de entrada (t, V_z) es necesario determinar el acceso a los prefijos dentro del árbol de sufijos. Es por ello que se realizó un experimento para ubicar la longitud necesaria para la elección de los prefijos. Este experimento nos permite confirmar que la ubicación de cada uno de

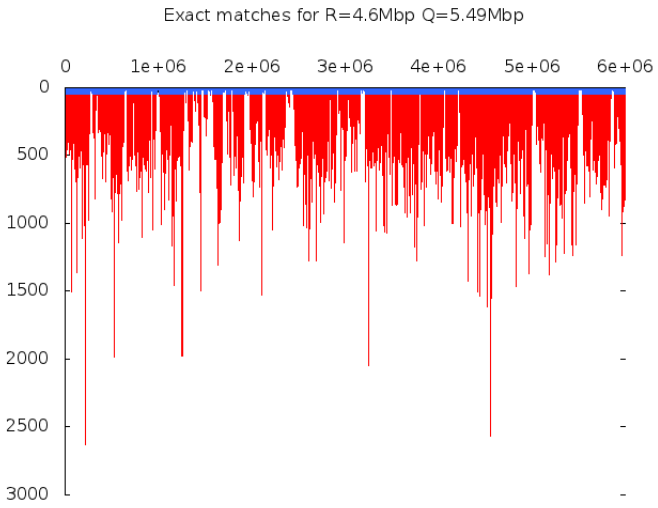

Figura 5: Identificación de coincidencias máximas en árbol de sufijos.

los prefijos difiere en longitud, debido a esto se hace necesario definir prefijos de longitud variable. Al utilizar prefijos de longitud variable es posible tener árboles de sufijos dispersos balanceados debido a que se podrían agrupar aquellos prefijos con mayor frecuencia de ocurrencia.

Es por ello que debemos encontrar un conjunto valido de prefijos para dividir el árbol de sufijos en árboles de sufijos dispersos con el requerimiento de que puedan construirse en memoria principal.

Definición 11 *Sea $f(p)$ el número de veces que un prefijo ocurre en S. Sea MSSTS (tamaño máximo del árbol de sufijos disperso) la cantidad máxima de espacio de memoria en bytes que puede ser reservado al árbol de sufijos disperso durante la construcción del árbol. Sea NS el tamaño de un nodo del árbol de sufijos en bytes.*

El objetivo es encontrar un conjunto de pares de entrada valido tal que $\forall p \epsilon V_z, 2x\, f(p) < \frac{MSSTS}{NS}$.
Una vez definido el conjunto de prefijos se procede a la construcción del árbol de sufijos disperso. Para ello se ubican en cada árbol de sufijos disperso el conjunto de pares de entrada valido que serán las raíces de cada árbol de sufijos disperso. Una vez definido el conjunto de prefijos se procede a la construcción del árbol de sufijos disperso.
La construcción del árbol de sufijos disperso se realiza siguiendo el algoritmo de Ukkonen y removiendo aquellos nodos y bordes que no tienen en común el prefijo de la raíz del árbol de sufijo disperso.
En la figura 6 se muestra el árbol de sufijos distribuido con cada uno de los árboles de sufijos disperso. Para una cadena aacacccacacaccacaaa$ con sus respectivos nodos raíz etiquetados como $r_{aa}, r_{ac}, r_{ca}, r_{cc}, r_{a\$}$ and $r_{\$}$. Una característica que el

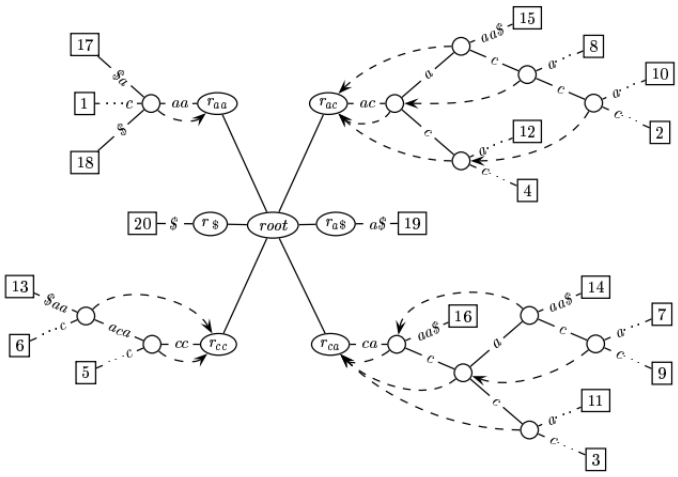

Figura 6: Árbol de sufijos distribuido para la cadena: aacacccacacaccacaaa$.

árbol de sufijos distribuido tiene es que únicamente mantiene los enlaces de sufijos que pertenecen al árbol de sufijos disperso. Aquellos enlaces de sufijos que

apunten a otros nodos que no están en el árbol de sufijos disperso son descartados en la fase de construcción del árbol.
Como se mencionó anteriormente el enlace de sufijo no solo permite la construcción en tiempo lineal del árbol de sufijos sino que hace que las búsquedas de coincidencias sean también lineales. En consecuencia surge ahora un nuevo problema al proponer esta estructura de datos modificada:

¿Es posible mantener la complejidad de búsqueda de coincidencias en un árbol de sufijos $O(m + k)$, en donde m es la longitud de la secuencia de consulta y k es el número de ocurrencias al utilizar un árbol de sufijos distribuido?

Para determinar el impacto del uso de los enlaces de sufijos en un árbol de sufijos se ejecutaron algunos experimentos para conocer la utilización de los enlaces de sufijos al hacer la búsqueda de coincidencias y la ubicación de los enlaces de sufijos dentro del árbol de sufijos.
En las figuras 7 y 8 se muestra el recorrido del árbol de sufijos. Para cada uno de los sufijos de la secuencia de consulta al utilizar el enlace de sufijo se avanza en el árbol de sufijos hacia arriba o abajo. Al navegar con el enlace de sufijo se evita hacer comparaciones de acuerdo a la ubicación del nodo dentro del árbol de sufijos, es decir, se consumen x caracteres. Una vez definido la utilización de los

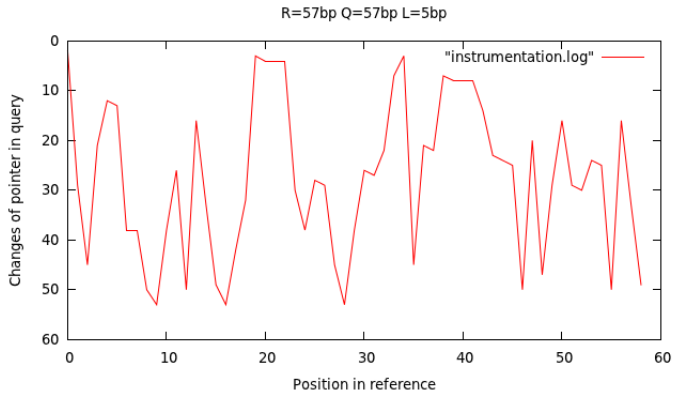

Figura 7: Recorrido del árbol de sufijos al utilizar los enlaces de sufijos.

enlaces de sufijos se procede a determinar en que posiciones dentro del árbol de sufijos se ubican los enlaces de sufijos. Para ello se realizó un conteo del número de enlaces de sufijos. En las figuras 9 y 10. Esta evaluación nos permite identificar que el uso de los enlaces de sufijos es dependiente de la secuencia de referencia y que la mayoría de los enlaces de sufijos se encuentran en la zona inferior del árbol de sufijos.

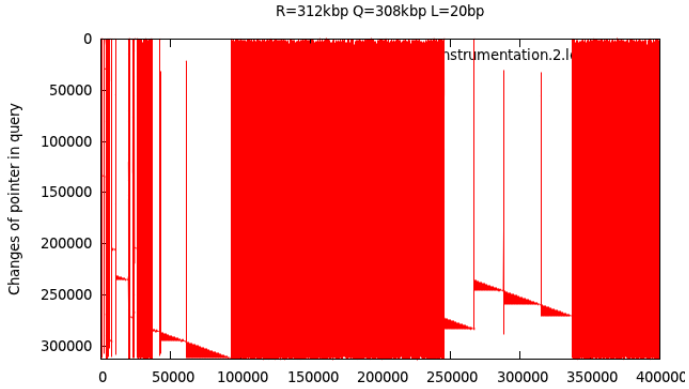

Figura 8: Recorrido del árbol de sufijos al utilizar los enlaces de sufijos.

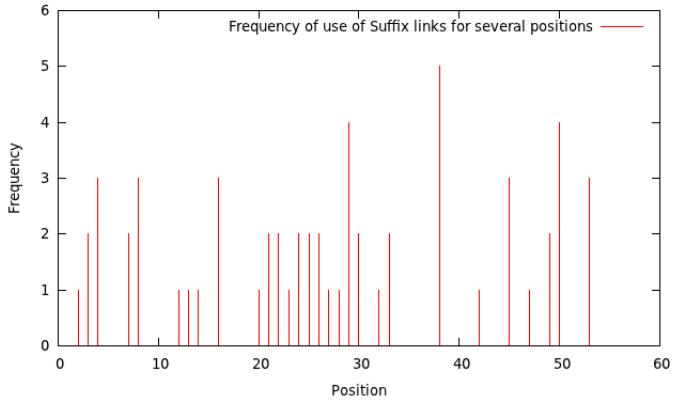

Figura 9: Frecuencia de enlaces de sufijos.

3.4. Búsqueda de coincidencias exactas en un DST

El propósito de adaptar una estructura de datos como el árbol de sufijos es lograr reducir el tiempo de búsqueda de coincidencias exactas. Para ello se propone una primera aproximación a la búsqueda de coincidencias exactas máximas en un árbol de sufijos distribuido.
La idea general se basa en dividir la secuencia de consulta en segmentos y asignar la búsqueda de cada sufijo dentro de cada segmento en el árbol de sufijos disperso correspondiente. La búsqueda de coincidencias exactas sigue en el árbol de sufijos y posteriormente determina las posiciones en las cuales se encuentran las coincidencias. Al evaluar el siguiente sufijo pueden ocurrir dos casos:

1. El siguiente sufijo se encuentra en el mismo árbol de sufijos disperso. La

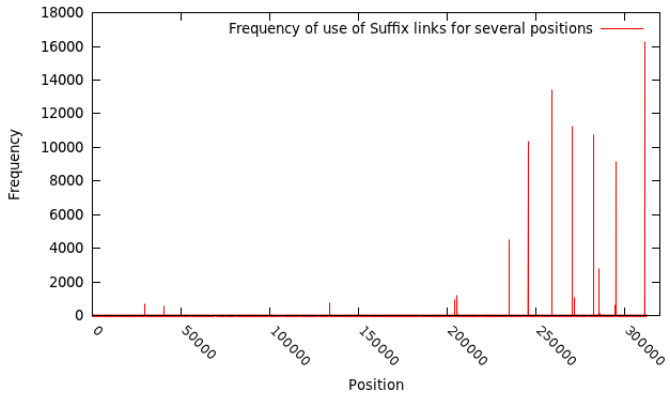

Figura 10: Frecuencia de enlaces de sufijos.

búsqueda continúa de manera local.

2. La búsqueda del siguiente sufijo se realiza en otro árbol de sufijos disperso.

El segundo caso es de especial relevancia ya que implica el proponer una estrategia de búsqueda del árbol de sufijos disperso en donde se debe realizar la búsqueda y adicionalmente continuar la búsqueda en ese árbol.
Para solucionar este caso, se propone el envío de la petición de búsqueda del sufijo en el árbol correspondiente. Para determinar el árbol de sufijos disperso que le corresponde el árbol consulta el prefijo del sufijo y envía la ubicación del sufijo al árbol de sufijos disperso correspondiente. El árbol que recibe la petición la encola en la lista de sufijos que tiene que procesar y resuelve la búsqueda de coincidencias exactas máximas del sufijo que se le ha asignado.
En la figura 11 se muestra de manera gráfica la búsqueda de coincidencias exactas para una secuencia de consulta, la división en segmentos de la misma y la búsqueda del primer sufijo de cada segmento en el árbol de sufijos distribuido.

4. Publicaciones

Ninguna.

5. Seguimiento

En el primer año se ha cumplido con la adaptación de una estructura de datos que permita su utilización en el alineamiento de datos biológicos a gran escala.

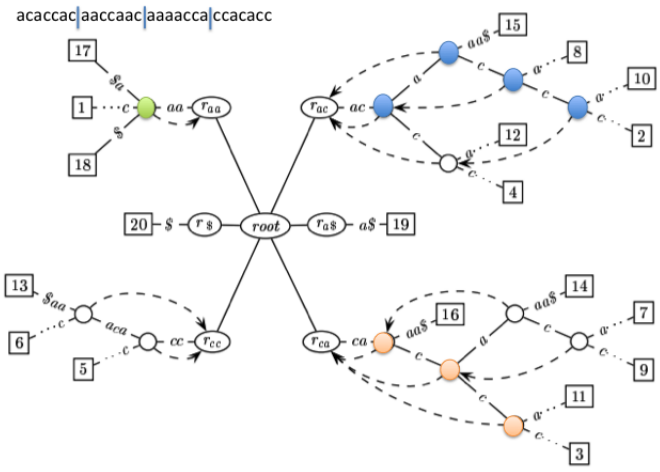

Figura 11: Búsqueda de coincidencias exactas máximas en un árbol de sufijos distribuido.

Dicha estructura será la base del desarrollo de MUMmer en HPC. La estructura es capaz de manejar grandes volúmenes de datos y se pueden realizar operaciones de consulta sobre ella.

6. Planificación

En el segundo año se realizará el diseño y evaluación experimental de un algoritmo paralelo y distribuido de búsquedas de coincidencias exactas máximas. Para ello se aprovechará la estructura de datos diseñada en el primer año. Se adaptará este algoritmo a la aplicación MUMmer para su ejecución en entornos de HPC. En el tercer año se refinará la ejecución de MUMmer en entornos de HPC. Se tomará como base el programa original, MUMmer, y se realizarán las modificaciones pertinentes para mejorar la fase de agrupamientos de MUMs y MEMs. Estancia en otra unidad de investigación, EMBL Heidelberg Alemania. Adaptar la propuesta de búsqueda distribuida de coincidencias exactas desarrollado para su ejecución en las instalaciones de EMBL. Redactar la tesis doctoral y preparar su defensa.

7. Análisis de las publicaciones pendientes de publicar

Ninguna.

8. Publicaciones previstas

European Symposium on Algorithms - ESA - Abril 2013
International Symposium on Algorithms and Computation - ISAAC - 2013
International Conference on Parallel Processing - Euro-Par - 2013
Asia Pacific Bioinformatics Conference - APBC - Julio 2012
IEEE International Symposium on Parallel and Distributed Processing with Applications - ISPA - 2013

9. Conclusiones

Durante este primer año de investigación doctoral se ha estudiado y evaluado la estructura de datos utilizada en la aplicación MUMMer. Se han diseñado diferentes experimentos para determinar la mejor manera de dividir dicha estructura y aprovechar la nueva estructura en entornos de HPC. Se evalúo la utilidad de los enlaces de sufijos en un árbol de sufijos y como se pueden aprovechar para la búsqueda de coincidencias exactas máximas.

Se ha adaptado el árbol de sufijos para su ejecución en entornos de HPC y se ha propuesto una solución al problema de la restricción de memoria cuando la cadena de referencia crece. La estructura de datos adaptada se ha utilizado para proponer una primera aproximación a la solución de mejorar el tiempo de búsqueda de coincidencias exactas máximas.

En el siguiente año se busca determinar si la propuesta de búsqueda de coincidencias exactas máximas en el árbol de sufijos distribuido permite mejorar el tiempo de ejecución bajo entornos de HPC. Adicionalmente se tendrán que resolver los siguientes problemas

- Política de carga de trabajo para cada árbol de sufijos disperso: garantizar que todas las búsquedas se resuelvan de manera local.
- Búsqueda de coincidencias exactas máximas en el árbol de sufijos disperso de forma paralela, es decir, realizar múltiples búsquedas para distintos sufijos.

Referencias

[Barsky et al., 2010] Barsky, M., Stege, U., and Thomo, a. (2010). A survey of practical algorithms for suffix tree construction in external memory. *Software: Practice and Experience*, 40(11):965–988.

[Clifford, 2005] Clifford, R. (2005). Distributed suffix trees. *Journal of Discrete Algorithms*, (October 2006).

[Ghoting and Makarychev, 2010] Ghoting, A. and Makarychev, K. (2010). I/O efficient algorithms for serial and parallel suffix tree construction. *ACM Transactions on Database Systems*, 35(4):1–37.

[Gusfield, 2007] Gusfield, D. (2007). *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge Univ. Press.

[Japp, 2004] Japp, R. (2004). The Top-Compressed Suffix Tree A Disk-Resident Index for Large Sequences. *Bioinformatics Workshop 21st Annual British national Conference on Databases*, (January).

[Kurtz et al., 2004] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., Antonescu, C., and Salzberg, S. L. (2004). Versatile and open software for comparing large genomes. *Genome Biology*, 5(R12):R12.1–R12.9.

[Mansour, 2012] Mansour, E. (2012). ERA : Efficient Serial and Parallel Suffix Tree Construction for Very Long Strings. 5(1):49–60.

[McCreight, 1976] McCreight, E. M. (1976). A space-economical suffix tree construction algorithm. *J. ACM*, 23:262–272.

[Needleman and Wunsch, 1970] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

[Phoophakdee and Zaki, 2007] Phoophakdee, B. and Zaki, M. J. (2007). Genome-scale disk-based suffix tree indexing. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, page 833.

[Sadakane,] Sadakane, K. Compressed Suffix Trees with Full Functionality. *Communication*, 4:1–18.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197.

[Ukkonen, 1992] Ukkonen, E. (1992). Constructing Suffix Trees On-Line in Linear Time. In Van Leeuwen, J., editor, *Proceedings of the IFIP 12th World Computer Congress on Algorithms Software Architecture Information Processing 92 Volume 1*, volume 1, pages 484–492. North-Holland Publishing Co.