Título: **Diseño de una aplicación bioinformática para el estudio de repeticiones de patrones en cadenas de DNA**

Volumen: **1**
Alumno: **Román Roset Mayals**

Director: **Xavier Messeguer Peypoch**
Co-director: **J. A. Subirada Torrent**
Departamento: **LSI**
Fecha: **20-01-2003**

Índice general

Índice de figuras

Índice de cuadros

Parte I

Estudio tecnocientífico

Parte II

Desarrollo práctico

Capítulo 1

Introducción

frase

autor, origen, 1781

En este proyecto se han desarrollado dos aplicaciones: *MREPATT* y *REPATT*. La primera es una generalización de la segunda y, así mismo, *REPATT* es el núcleo de *MREPATT*. Se ha echo esta diferenciación puesto que su diseño e implementación ha sido completamente diferente en las dos.

En los siguientes capítulos vamos a describir cada una de las dos herramientas, empezando por el núcleo, *REPATT*, descrito en el capítulo 4 y seguidamente en el capítulo **??** el programa que lo usará para generalizar los resultados a múltiples patrones y cromosomas: *MREPATT*. En los dos capítulos siguientes se detallan los requerimientos de *MREPATT*, que son en definitiva los resultados que se espera conseguir con la suma de los dos programas, y se explican los conceptos básicos necesarios para seguir los algoritmos.

MREPATT

La aplicación resultante de este proyecto recibe el nombre de *MREPATT* [1]. Se trata de una aplicación Cliente/Servidor sobre la Web, basada en el *Web Server*. La capa de presentación, como se ha dicho, es sobre la Web proporcionando la interficie cualquier Cliente Universal (Web browser) que soporte JavaScript. El paso de peticiones al servidor se hace llamando a programas CGI escritos en Perl.

La aplicación recibe como entrada uno o varios conjuntos de cromosomas, y uno o varios patrones. Dichos conjuntos de cromosomas se agrupan en especies, permitiendo escoger los cromosomas de estas especies que el usuario crea oportunos. En el capítulo 2 podremos ver los datos de salida que ofrece nuestra aplicación.

[1] del ingles "***M**ultiple search for consecutive **RE**peat **PATT**erns*", búsqueda múltiple de patrones repetidos consecutivos.

MREPATT es un conjunto de funciones escritas en lenguaje PERL. Para cada cómputo que se ha de realizar sobre un cromosoma, *MREPATT* hace una llamada al programa *REPATT*.

REPATT

REPATT es el núcleo de la aplicación *MREPATT*. Es un programa escrito en C++ que recibe como entrada un fichero, en general un cromosoma o una parte del cromosoma, en formato *fasta* [2] y un patrón. El programa recorre todo el cromosoma en busca del patrón que ha recibido como entrada y da como salida los siguientes ficheros:

- Número de ocurrencias para todas las repeticiones encontradas del patrón.
- Número de ocurrencias teóricas para las repeticiones encontradas.
- Cadena de Markov asociada al fichero de entrada. [3].
- Tanto por ciento aparición de cada base en el fichero de entrada.
- Fichero de imagen de localización de cada repetición en el fichero de entrada.

Estos ficheros de salida han sido realizados en un formato de fácil uso para herramientas estadísticas y gráficas. Concretamente se ha hecho hincapié en el formato en texto plano para el uso de la herramienta de libre distribución *"gnuplot"*.

[2] ver sección 7.6.1
[3] Ver sección 3.4

Capítulo 2

Análisis de Requerimientos

El objetivo principal de nuestro sistema es desarrollar una aplicación ejecutable por Internet que cuente las ocurrencias de cada repetición de un patrón en uno o varios cromosomas, y nos muestre imágenes de las posiciones de éstas en la región del cromosoma que se pida.Así mismo se pide que se pueda elegir los cromosomas a trabajar, inclusive los que pueda incorporar el biólogo.

A continuación se detallan los requerimientos comentados.

2.1. Requerimientos funcionales

2.1.1. Cromosomas y patrones de entrada

La aplicación debe comparar los resultados obtenidos en diferentes especies. Estos grupos vienen dados por el árbol filo-genético universal [1] que agrupa a los seres vivos en tres categorías o dominios.

El sistema debe contener un base da datos de algunas de las especies del árbol filogenético. Además se ha de permitir escoger grupos dentro del árbol, de estos grupos las especies o subgrupos que se deseen, y dentro de cada especie, los cromosomas que se requieran.

El usuario podrá introducir uno o varios patrones, para cada patrón introducido corresponderá una solución diferente. Por tanto han de aparecer tantas soluciones como patrones introducidos. Cada patrón define el lenguaje de búsqueda como la unión de éste y su complementario.

2.1.2. Gestión de los cromosomas de entrada

Se ha de permitir al usuario:

- Introducir nuevos ficheros de cromosomas e incorporarlos conjuntamente al árbol filo-genético existente. Los cromosomas que se introducen puede estar en un for-

[1] Ver sección **??**

mato *Fasta* o pueden ser un grupo de dichos ficheros en un paquete comprimidio en formato *zip*.

- Borrar los ficheros que ha introducido.
- Listar los cromosomas que tiene dados de alta.

En definitiva un modulo para interactuar con los cromosomas propios de cada usuario. No se pide una gestión privada de éstos, por tanto, los cromosomas introducidos por un usuario podrán ser vistos por otros, siempre y cuando sepan el identificador del usuario al que pertenecen.

2.1.3. Informes de resultados sobre la búsqueda de repeticiones del patrón

Los resultados de la aplicación se dividen claramente en tres partes, de la mas general a la mas concreta.

Primeramente aparecen los siguientes informes generales, válidos para todo el conjunto de patrones que ha introducido el usuario:

- Resumen de resultados: Se trata de una tabla comparativa donde para cada especie y cada patrón aparece el orden máximo y su ratio real-teórico (ver figura 2.1).

Summary of Results

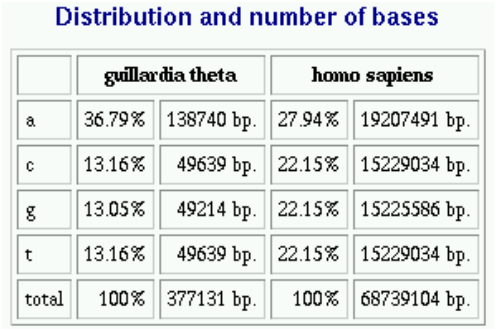
	att		at	
SPECIES	last rep.	last ratio	last rep.	last ratio
guillardia theta	5	8.0210e+00	5	1.0946e+00
homo sapiens	20	1.0180e+28	47	4.0575e+53

Figura 2.1: Pantalla de la página de resumen de resultados

- Distribución y número de bases: Tabla en la que aparecen para cada base b y cada especie $\mathcal{C}$, los escalares $|\mathcal{C}|_b$ y $|\mathcal{C}|$ expresados tanto en valores absolutos como porcentajes (ver figura 2.2).
- Cadena de Markov: Se muestra la matriz de transición de la cadena de Markov asociada a cada grupo de la entrada (ver figura 2.3).

Posteriormente, para cada patrón la aplicación ha de realizar los informes que a continuación se detallan:

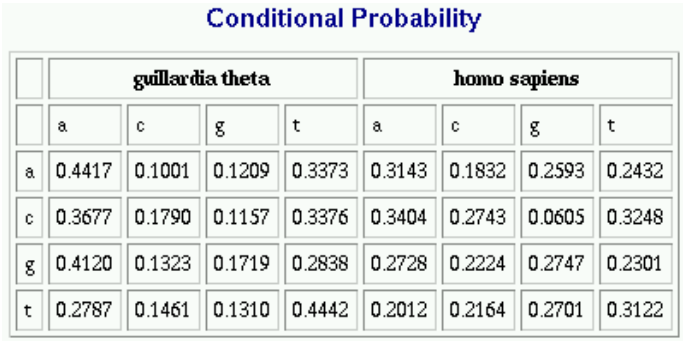
- Tabla de resultados: En esta tabla aparece una comparativa entre los diferentes grupos seleccionados. Se reporta para cada orden k cuyo conjunto de ocurrencias



Distribution and number of bases

	guillardia theta		homo sapiens	
a	36.79%	138740 bp.	27.94%	19207491 bp.
c	13.16%	49639 bp.	22.15%	15229034 bp.
g	13.05%	49214 bp.	22.15%	15225586 bp.
t	13.16%	49639 bp.	22.15%	15229034 bp.
total	100%	377131 bp.	100%	68739104 bp.

Figura 2.2: Pantalla de la página de la distribución de las bases



Conditional Probability

	guillardia theta				homo sapiens			
	a	c	g	t	a	c	g	t
a	0.4417	0.1001	0.1209	0.3373	0.3143	0.1832	0.2593	0.2432
c	0.3677	0.1790	0.1157	0.3376	0.3404	0.2743	0.0605	0.3248
g	0.4120	0.1323	0.1719	0.2838	0.2728	0.2224	0.2747	0.2301
t	0.2787	0.1461	0.1310	0.4442	0.2012	0.2164	0.2701	0.3122

Figura 2.3: Pantalla de la matriz de tansición de la cadena de Markov

de ese orden no esté vacío, la cardinalidad de dicho conjunto de ocurrencias, la cardinalidad del conjunto de ocurrencias únicas de orden k y el ratio real-teórico de las ocurrencias de orden k (ver figura 2.4).

- Gráficos de las ocurrencias. Se presentan los siguientes gráficos en dos dimensiones, cuyas abscisas son el orden k y las ordenadas la cardinalidad del conjunto de ocurrencias o el ratio real-teórico, dichas ordenadas siempre en escala logarítmica. En cada gráfico se muestra conjuntamente cada grupo seleccionado.

  - gráfico de ocurrencias acumuladas.
  - gráfico de ocurrencias únicas.
  - gráfico de ocurrencias únicas y de ocurrencias acumuladas.
  - gráfico de ocurrencias teóricas.
  - gráfico de ocurrencias teóricas y ocurrencias acumuladas.
  - gráfico de ocurrencias teóricas y ocurrencias únicas.
  - gráfico del ratio real-teórico.

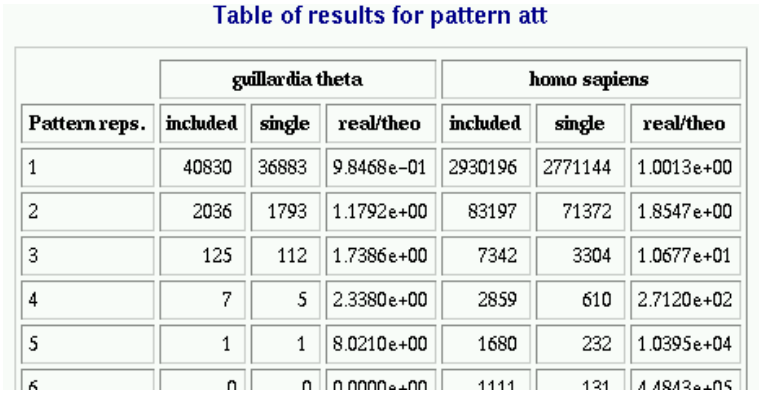


Table of results for pattern att

	guillardia theta			homo sapiens		
Pattern reps.	included	single	real/theo	included	single	real/theo
1	40830	36883	9.8468e-01	2930196	2771144	1.0013e+00
2	2036	1793	1.1792e+00	83197	71372	1.8547e+00
3	125	112	1.7386e+00	7342	3304	1.0677e+01
4	7	5	2.3380e+00	2859	610	2.7120e+02
5	1	1	8.0210e+00	1680	232	1.0395e+04
6	0	0	0.0000e+00	1111	131	4.4843e+05

Figura 2.4: Pantalla de la página de resultados

Finalmente, para cada patrón mostramos el mapa de las ocurrencias de cada uno de los cromosomas del grupo seleccionado. Por tanto habrá tantos módulos de imágenes como cromosomas dentro de los grupos seleccionados.

Las imágenes de la localización de las ocurrencias nos han de mostrar un gráfico detallando las posiciones de cada ocurrencia, el orden a la que pertenecen y la dirección (si la ocurrencia es la complementario o no). Además se ha de diferenciar entre las ocurrencias únicas y las sobrepuestas.

El modulo que muestre dichas imágenes ha de permitir al usuario elegir la zona del cromosoma que se quiere analizar. Dicha zona se podrá elegir tanto seleccionando la posición de la base inicial y final en términos absolutos como el tanto por ciento de estas posiciones.

En definitiva ha de realizarse un modulo capaz de "navegar" a través del cromosoma.

Cuando la longitud de la región a mostrar sea suficientemente pequeña (de 100 bases) también se tienen que mostrar los nucleótidos que forman dicha región.

Podemos ver un ejemplo de imagen en la figura 2.5.

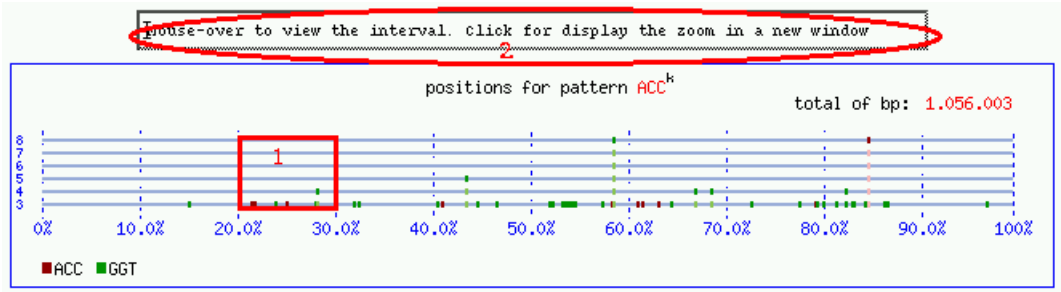


Figura 2.5: Pantalla de un mapa generado por el módulo de zoom

2.2. Requerimientos no funcionales

Los aspectos no funcionales que podríamos destacar de la aplicación son:

- Cambiable:
  - Extensible: El diseño se ha realizado pensando en futuras funcionalidades, tales como nuevos informes, cambio del algoritmo de búsqueda, introducción de nuevas especies en la base de datos, ...
  - Portable: La aplicación es fácilmente portada a cualquier sistema operativo.
  - Mantenible: El diseño se ha pensado para que sea mantenible, para ello se han usado técnicas de programación en web a partir de macros, pudiendo hacer un diseño con arquitectura de tres capas.
- Interoperable: En esta aplicación se comunican los componentes de *MREPATT* y *REPATT* a través de llamadas directas y ficheros de texto plano. *MREPATT* se puede preparar fácilmente para comunicarse con otras aplicaciones directamente a través del protocolo "http".
- Eficiencia: El algoritmo de búsqueda tiene coste temporal mínimo, es decir es lineal, y tiene coste espacial mínimo.
- Fiable: La aplicación es tolerante a fallos, puesto que, aunque se pierda la conexión guarda los datos pedidos. Es robusta en la medida que se comprueban todos los datos de entrada.
- Reusable: Se ha reusado software existente, tanto librerías como programas y los componentes realizados se han pensado para poder ser reutilizados a diferentes niveles: librerías en javascript para la capa de presentación web y librerías en *C++* y *Perl* para capa de dominio.
- Facilidad de uso: La aplicación resultante tiene armonía, simetría y predictibilidad. Se ha puesto hincapié en realizar una capa de presentación intuitiva, sencilla y cómoda. Cuenta con ayuda en línea para que usuario sepa en todo momento qué hacer y cómo hacerlo.

Capítulo 3

Conceptos básicos

Antes de detallar los objetivos es preciso formalizar las definiciones de los términos de *universo de discurso*.

En los siguientos apartados describiremos en términos formales la nomenclatura que se seguirá a lo largo de este capítulo.

Explicaremos qué son los autómatas finitos deterministas, los cuales se han utilizado como base en el algorítmo de búsqueda.

Por último se hace una descripción de las cadenas de markov, las cuales nos han servido para realizar el modelo teórico para nuestros cálculos.

3.1. Definiciones de los términos utilizados

Cromosoma o especie: $\mathcal{C} = \langle b_1, b_2, \ldots, b_n \rangle$ es una palabra de longitud n sobre el alfabeto $\Sigma_x = \Sigma \cup \{x\}$, donde $\Sigma = \{a, c, g, t\}$. A cada símbolo de Σ lo llamamos base, y al símbolo x base desconocida. Dado que nuestra entrada es un cromosoma o grupo de cromosomas (especie), y una especie es una concatenación de cromosomas, anotaremos también como $\mathcal{C}$ a la especie.

Complementario de una base: Definimos el *complementario de una base* $c(b)$ como la aplicación biyectiva de Σ a Σ:

$$c(b) = [a \mapsto t, c \mapsto g, g \mapsto c, t \mapsto a]$$

Patrón primario: $p = \langle b_1, b_2, \ldots, b_l \rangle$ es una palabra no vacía de longitud l sobre el alfabeto Σ.

Patrón complementario de p: $p_c = \langle c(b_l), c(b_{l-1}), \ldots, c(b_1) \rangle$ siendo p el patrón primario de longitud l. El patrón complementario es en términos biológicos la cadena de *DNA* complementaria al patrón primario.

Lenguaje de búsqueda de orden k : Es el lenguaje resultante de la unión de los dos patrones anteriormente definidos, concatenados cada uno k veces. Lo definimos como $\mathcal{L}_k = \{w \in p^k \cup p_c^k\}$.

Lenguaje de búsqueda : El lenguaje de búsqueda es la unión de los lenguajes formados por la repetición consecutiva del patrón primario, que es el que nos dan como entrada,

y su patrón complementario.

$$\mathcal{L} = \bigcup_{n \geqslant 0} \mathcal{L}_n$$

Este es el lenguaje que haremos servir para construir el autómata que hará la búsqueda de coincidencias en el cromosoma.

Ocurrencia: Cada ocurrencia es una coincidencia dentro del cromosoma con el patrón primario o secundario. Formalmente definimos ocurrencia del patrón como el par formado por:

$$o = (i = \langle l, r \rangle, s = \langle b_l, \ldots, b_r \rangle), \mathcal{C} = \alpha s \beta \quad |\alpha|, |\beta| \geqslant 0 \quad |s| > 1 \quad \alpha, \beta \in \Sigma^*$$

Cada ocurrencia esta compuesta por el intervalo formado por la posición inicial y final donde ha aparecido la coincidencia dentro del cromosoma, y por la propia coincidencia. Por motivos de claridad nos referiremos a cada elemento del intervalo como o_l, o_r y a la subpalabra como o_s.

Conjunto de ocurrencias: Formalizaremos el conjunto de ocurrencias **válidas**, o simplemente conjunto de ocurrencias, como:

$$O = \{o \,|\, o_s \in \mathcal{L}\}$$

A este conjunto también lo llamaremos ***conjunto de ocurrencias acumuladas*** .

Conjunto de ocurrencias de orden k El conjunto de ocurrencias de orden k es el subconjunto de O cuyas subpalabras de cada ocurrencia coinciden con alguno de los dos patrones K veces concatenados. Se define como

$$O^k = \{o \,|\, o_s \in \mathcal{L}_k\}$$

Conjunto de ocurrencias únicas de orden k: El conjunto de ocurrencias únicas de orden k es una subconjunto de las ocurrencias de orden k formado por las ocurrencias únicas o^k que no son subpalabra de una ocurrencia de orden superior. Lo definimos como sigue:

$$U^k = \{o \,|\, o = (\langle l, r \rangle, w) \in O^k \,,\, (\langle i, j \rangle, v) \notin O^{k+1} \quad \forall i, j \quad i \leqslant o_l \leqslant o_r \leqslant j\}$$

Orden máximo K: Es el número máximo de veces que se ha encontrado un patrón repetido consecutivamente en el cromosoma. Lo definimos como:

$$K = \text{máx}\{k \,|\, O^k \supset \emptyset\}$$

Número ocurrencias teóricas de orden k: el número ocurrencias teóricas de orden k en el cromosoma $\mathcal{C}$, abreviado no_k, es cardinalidad media en términos estadísticos del conjuntos de ocurrencias de orden k.

Ratio real-teórico de las ocurrencias de orden k: Es el ratio resultante de $\frac{|O^k|}{no_k}$ que servirá para dar una idea de la aparición de las ocurrencias de orden k.

3.2. Autómatas finitos deterministas

Un *autómata finito determinista* (abreviado como DFA, del inglés *deterministic finite automaton*) es una 5-tupla de la forma

$$M = \langle \mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F} \rangle$$

cuyos componentes se definen a continuación:

- $\mathcal{Q}$ es un conjunto no vacío cuyos elementos llamaremos *estados*
- Σ es el alfabeto de entrada
- δ es la función de transición que se define:

  $$\delta : \mathcal{Q} * \Sigma^* \longrightarrow \mathcal{Q},$$

  que representaremos con notación multiplicativa,

  $$\forall q \in \mathcal{Q} \quad \forall w \in \Sigma^* \qquad \delta(q, w) = q \cdot w$$

  y satisface los dos axiomas siguientes:

  $$\forall q \in \mathcal{Q} \quad \forall x, y \in \Sigma^* \begin{cases} (1) & q \cdot \lambda = q \\ (2) & q \cdot (xy) = (q \cdot x) \cdot y, \end{cases}$$

- $q_0 \in \mathcal{Q}$ es el *estado inicial*
- $\mathcal{F} \subseteq \mathcal{Q}$ es el conjunto de *estado finales* o aceptadores

A partir de estos axiomas es inmediato comprobar que cualquier función de transición queda completamente definida especificando simplemente los valores sobre el conjunto finito $\mathcal{Q} * \Sigma$. La función de transición se suele especificar mediante una tabla que llamaremos *tabla de transiciones*, que tiene como entradas los estados de $\mathcal{Q}$ y los símbolos de Σ.

Otra manera de describir un autómata finito es mediante un *diagrama de transiciones* del autómata. Se trata de un grafo dirigido que tiene por vértices los estados del autómata y para cada transición de la forma $q_i \cdot a = q_j$ hay un arco que va de q_i a q_j etiquetado como a. El estado inicial se suele identificar con una flecha que incide y los estados finales con una cruz o un doble círculo.

Podemos ver un ejemplo de un diagrama de transiciones de un autómata en la figura 3.1. En este caso se trata de un autómata reconocedor de palabras que contienen a o b y terminan en ab.

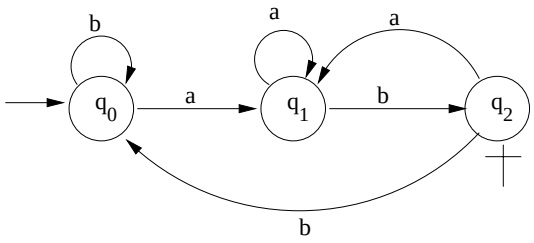


Figura 3.1: Ejemplo de un diagrama de transiciones de un DFA

3.2.1. Lenguajes reconocidos por un DFA. Lenguajes regulares

Dado un autómata finito $M = \langle \mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F} \rangle$, llamaremos *lenguaje reconocido o aceptado* por esta autómata y se representa por $\mathcal{L}(M)$, el conjunto de palabras que en operar sobre el estado inicial dan un estado final. Formalmente:

$$\mathcal{L}(M) = \{w \in \Sigma^* | q_o \cdot w \in \mathcal{F}\}.$$

En un autómata determinista hay una correspondencia biúnivoca entre palabras aceptadas y caminos que van del estado inicial a algún estado final en el grafo de transiciones.

Un lenguaje es *regular* cuando es reconocido por algún autómata finito. En el ejemplo de la figura 3.1 el lenguaje es:

$$\mathcal{L}_l(M) = \{w \in \{a, b\}^* | w = \delta \cdot ab\}, \delta \in \{a, b\}^*.$$

Dado que $\mathcal{L}_l(M)$ es reconocido por un autómata regular, $\mathcal{L}_l(M)$ es un lenguaje regular.Representaremos por $\mathcal{R}eg$ la familia de lenguajes regulares. Diremos que dos autómatas son equivalentes cuando reconocen el mismo lenguaje.

3.3. Uso de autómatas en MREPATT

Se ha elegido el uso de un autómata para hacer la búsqueda del patrón en el cromosoma. Esta estrategia permite realizar la búsqueda en tiempo lineal.

Básicamente las dos operaciones que se han usado en el programa son:

- creación del autómata
- unión de autómatas.

3.3.1. Algoritmo de creación del autómata

La creación recibe como entrada el patrón primario y a partir de éste crea el autómata. Cada nodo está asociado a un prefijo del patrón. Por tanto tenemos tantos estados como prefijos. El nodo asociado al prefijo mas corto ($\langle\rangle$) es el nodo inicial. El nodo

asociado al prefijo mas largo (el patrón) es el nodo final. La idea del algoritmo es la siguiente:

- creamos un nodo por cada prefijo del patrón.
- recorremos la lista de los prefijos del patrón ordenados de menor a mayor longitud.
- para cada prefijo al que llamaremos *origen*:
  - añadimos un arco por cada símbolo de nuestro alfabeto. El arco etiquetado con el símbolo a tiene como destino el nodo asociado al máximo prefijo del patrón que sea igual a un sufijo de *origen* concatenado a a. Entendemos como máximo el prefijo de mayor longitud. Si no se cumple esta condición para ningún prefijo, entonces el destino es el nodo inicial.

En el algoritmo, que se presenta en la figura 3.2, se hacen las siguientes suposiciones:

- PREFIX(p) : es la secuencia ordenada de mayor a menor longitud de los prefijos de p.
- SUFIX(p): es la secuencia de los sufijos de p.
- crear_nodo_por_cada_prefijo() : función que crea por cada prefijo del patrón un nodo asociado a este prefijo.
- máx(S) : devuelve la palabra mas larga del conjunto S de palabras.
- Cuando nos referimos a un nodo, lo expresamos directamente como su prefijo asociado.

---

procedure crear(**in** $patron = \langle b_1, b_2, \ldots, b_l \rangle$) **is**

$crear\_nodo\_por\_cada\_prefijo()$

$\{INV : \forall nodo \in PREFIX(origen)$: se han creado todos sus arcos$\}$
for each $origen \in PREFIX(patron)$ **do**
  for each $a \in \Sigma$ **do**
    $destino = \text{máx}(\{p \in PREFIX(patron) | s \in SUFIX(origen), s \cdot a = p\} \cup \{<>\})$
    $creaArco(origen, destino, a)$
  end for
end for
end

Figura 3.2: Algoritmo de creación del autómata

En el algoritmo hay dos bucles anidados, y un tercero implicito en la función máx(S). Dado que el segundo bucle recorre los símbolos y nuestro alfabeto se compone simpre de 4, el coste asintótico temporal del algoritmo es $\Theta(n^2)$. El coste espacial es $\Theta(n)$, siendo n la longitud del patrón, teniendo siempre $n+1$ estados en nuestro autómata.

Una precondición que se ha impuesto en los patrones que recibimos como entrada es que no superen una longitud de 20 caracteres sabiendo que en la mayoría de casos no superarán los 10.

A continuación veremos el funcionamiento con un ejemplo.

Ejemplo 3.3.1. Consideramos el patrón "ATTAT"su diagrama es el de la figura 3.3 , la tabla de transiciones resultante es la que se muestra en el cuadro 3.1. La longitud del patrón es 5 y el número de estados del autómata es efectivamente 6. Analicemos qué ocurre en el algoritmo cuando llega a los estados 1 y 5 respectivamente. Para

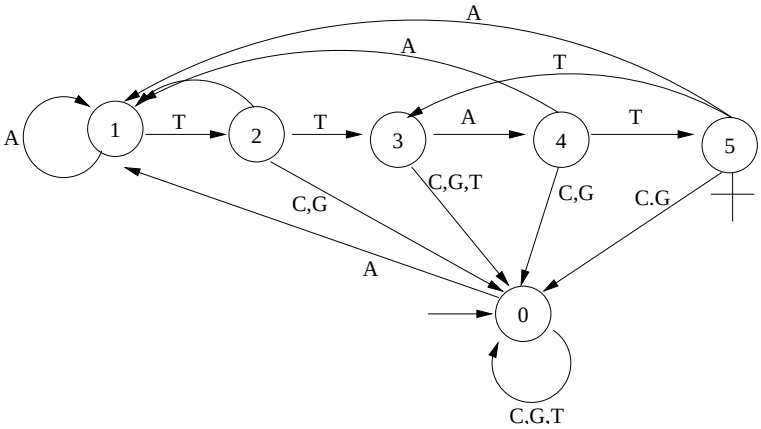


Figura 3.3: Diagrama de transiciones del DFA del ejemplo 3.3.1

	A	C	G	T	prefijo asociado
0	1	0	0	0	$\langle\rangle$
1	1	0	0	2	$\langle a\rangle$
2	1	0	0	3	$\langle at\rangle$
3	4	0	0	0	$\langle att\rangle$
4	1	0	0	5	$\langle atta\rangle$
†5	1	0	0	3	$\langle attat\rangle$

Cuadro 3.1: Tabla de transiciones del DFA de la figura 3.3

facilitar la lectura usaremos la siguiente nomenclatura:

- *Candidatos* : Lista de los los possibles destinos

  $Candidatos(a) = \{p \in PREFIX(patron) | s \in SUFIX(origen), s \cdot a = p\} \cup \{<>\}$

estado 0: origen = $\langle\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle a\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle\rangle \xrightarrow[a]{} \langle a\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle, \langle t\rangle$)= $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle\rangle \xrightarrow[c,g,t]{} \langle\rangle}$.

estado 1: origen = $\langle a\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle a\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle a\rangle \xrightarrow[a]{} \langle a\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle$) = $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle a\rangle \xrightarrow[c,g]{} \langle\rangle}$.

3. Candidatos($\langle t\rangle$) = $\{\langle at\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle a\rangle \xrightarrow[t]{} \langle at\rangle}$.

estado 2: origen = $\langle at\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle a\rangle, \langle\rangle\}$. Por tanto añadimos el arco $\boxed{\langle at\rangle \xrightarrow[a]{} \langle a\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle$) = $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle at\rangle \xrightarrow[c,g]{} \langle\rangle}$.

3. Candidatos($\langle t\rangle$) = $\{\langle att\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle at\rangle \xrightarrow[t]{} \langle att\rangle}$.

estado 3: origen = $\langle att\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle atta\rangle, \langle a\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle att\rangle \xrightarrow[a]{} \langle atta\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle, \langle t\rangle$) = $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle att\rangle \xrightarrow[c,g,t]{} \langle\rangle}$.

estado 4: origen = $\langle atta\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle a\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle atta\rangle \xrightarrow[a]{} \langle a\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle$) = $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle atta\rangle \xrightarrow[c,g]{} \langle\rangle}$.

3. Candidatos($\langle t\rangle$) = $\{\langle attat\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle atta\rangle \xrightarrow[t]{} \langle attat\rangle}$.

estado 5: origen = $\langle attat\rangle$ Para cada símbolo a,c,g,t escogemos su destino a partir de los candidatos:

1. Candidatos($\langle a\rangle$) = $\{\langle a\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle attat\rangle \xrightarrow[a]{} \langle a\rangle}$.

2. Candidatos($\langle c\rangle, \langle g\rangle$) = $\{\langle\rangle\}$. Por tanto añadimos los arcos $\boxed{\langle attat\rangle \xrightarrow[c,g]{} \langle\rangle}$.

3. Candidatos($\langle t\rangle$) = $\{\langle att\rangle, \langle\rangle\}$. Cogemos el máximo de dicho conjunto y añadimos el arco $\boxed{\langle attat\rangle \xrightarrow[t]{} \langle att\rangle}$.

3.3.2. Unión de autómatas

Dado dos lenguajes regulares como lo son los patrones primario y complementario, obtenemos sus autómatas con el algoritmo explicado en el apartado anterior. Como el lenguaje de búsqueda es la unión entre los dos lenguajes, aplicaremos la operación de unión entre autómatas para obtener un solo autómata con el que realizar la búsqueda.

Puesto que nos encontramos ante dos DFA, M_1 y M_2, se ha planteado la reunión vía producto cartesiano de los estados. La idea básica es considerar como estados del nuevo autómata los pares del producto cartesiano de los estados de M_1 por los de M_2 y la función de transición del autómata M_1 sobre el primer componente y la función de transición del autómata M_2 sobre el segundo. El conjunto de estados finales tendrá que ser el conjunto de pares que tienen estados finales en alguna de los dos componentes.

La figura 3.4 muestra el algoritmo de unión de dos autómatas.

Vemos que el coste asintótico temporal del algoritmo es $\Theta(n^3)$, no obstante, puesto que el alfabeto consta de cuatro símbolos podemos reducir el bucle interno con lo que el coste mas exacto es de $\Theta(n^2)$. el coste espacial es también de $\Theta(n^2)$, que son los estados resultantes de aplicar el producto cartesiano entre los dos autómatas.

En el caso de una unión se suele aplicar posteriormente una minimización del autómata, uniendo los estados que pertenecen a una misma clase de equivalencia (que son en realidad los mismos). La minimización reduce el coste espacial, no obstante no se ha utilizado, dado que en el caso peor tendremos 400 estados, y se ha considerado que éste se puede asumir.

A continuación veremos el funcionamiento con un ejemplo.

Ejemplo 3.3.2. Consideramos el patrón "att" y su complementario "aat" cuyas tablas aparecen en la tabla 3.2 y sus diagramas de transiciones en la figura 3.5.

La tabla de transiciones resultante es la que se muestra en la tabla 3.3 y su diagrama es el de la figura 3.6.

```
function union(in a is automata, in b is automata) return c is automata
    abiertos ← [⟨estado_inicial_a, estado_inicial_b⟩]
    while abiertos ≠ ∅ do
        ⟨e_a, e_b⟩, abiertos = [l] ← abiertos = [⟨e_a, e_b⟩ : l]
        if (estado_a[e_a] ∈ estados_finales_a ∨ estado_b[e_b] ∈ estados_finales_b) then
            estados_finales_c ← estados_finales_c ∪ {⟨e_a, e_b⟩}
        end if
        for each bp in Σ do
            siguiente_a ← estado_a[e_a](bp)
            siguiente_b ← estado_b[e_b](bp)
            if ⟨e_a, e_b⟩ ∉ abiertos then
                abiertos ← [⟨siguiente_a, siguiente_b⟩ : abiertos]
            end if
            estado_c[⟨e_a, e_b⟩](bp) ← ⟨siguiente_a, siguiente_b⟩
        end for
    end while
end
```

Figura 3.4: Algoritmo de unión de autómatas

	A	C	G	T	punto de interés
0	1	0	0	0	.aat
1	2	0	0	0	a.at
2	2	0	0	3	aa.t
†3	1	0	0	0	aat.

	A	C	G	T	punto de interés
0	1	0	0	0	.att
1	1	0	0	2	a.tt
2	1	0	0	3	at.t
†3	1	0	0	0	att.

Cuadro 3.2: Tablas de transiciones de los DFA que aceptan "aat" y "tta"

	A	C	G	T	punto de interés
00	11	00	00	00	.aat , .att
11	21	00	00	02	a.at , a.tt
21	21	00	00	32	aa.t , a.tt
02	11	00	00	03	.aat , at.t
†32	11	00	00	03	aat. , at.t
†03	11	00	00	00	.aat , att.

Cuadro 3.3: Tabla de transiciones del DFA resultante de la unión de los autómatas de la figura 3.5

La longitud de los patrón es 3 y el número de estados de los autómatas a unirse es efectivamente 4. Analicemos qué ocurre en hace el algoritmo en cuando llega a los estados 00 y 32 respectivamente.

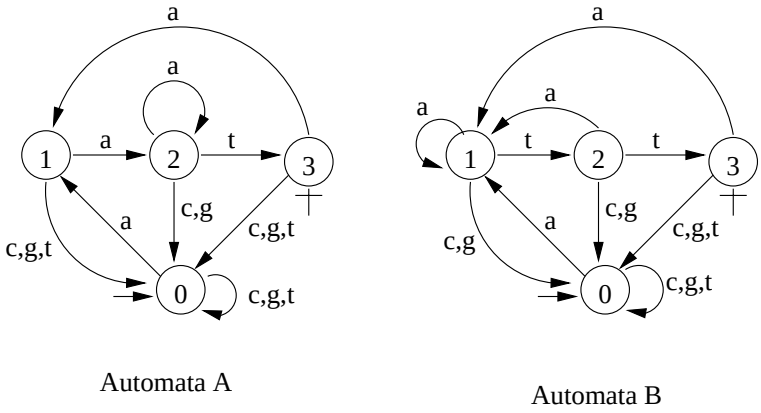


Figura 3.5: Diagrama de transiciones de los DFA que aceptan "aat" y "tta" respectivamente

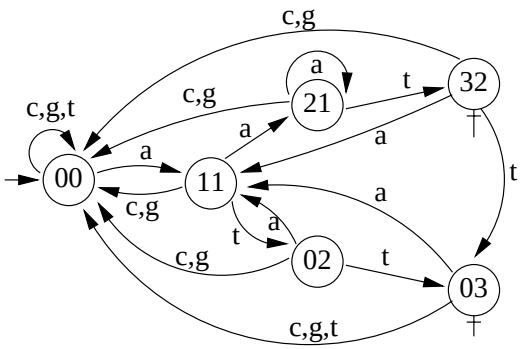


Figura 3.6: Diagrama de transiciones del DFA resultante de la unión de los autómatas de la figura 3.5

estado 00

1. Se comprueba que ninguno de los estados de la tupla sea final.
2. Para cada símbolo del alfabeto se crea un nuevo estado destino formado por la tupla de estados destino de cada uno de los autómatas, es decir, su producto cartesiano. En el caso del símbolo "a", por ejemplo, formamos la tupla $\langle 1, 1 \rangle$.
3. Se comprueba que el nuevo estado no esté en la cola de abiertos. En esta caso no es así e introducimos el nuevo estado en la cola.

estado 32

1. Se comprueba que ninguno de los estados de la tupla sea final. En este caso, el estado 3 es final para el autómata A, por tanto el estado 32 debe ser final.

Los siguientes pasos proceden igual que en el parágrafo anterior para el estado 00.

3.4. Cadenas de Markov

El nombre de esta módelo de la investigación operativa proviene del matemático ruso *A. A. Markov* (1856-1922) que estudió un tipo de procesos temporales donde la probabilidad de que en un tiempo t una variable esté en un determinado estado o adquiera un determinado valor depende del valor en el paso anterior $t-1$.

Formalmente podemos citar la definición siguiente: Una máquina de estados finitos con probabilidades para cada transición, esto es, una probabilidad que el siguiente estado es s_j dado que el estado actual es s_i [KS76]

Otros libros mas modernos que profundizan en las cadenas de markov y en la probabilidad en general son [Ste95], [Hle97].

Una cadena de Markov es una secuencia de estados discretos en el tiempo o en el espacio con un valor de probabilidad asignado a cada posible transición. La probabilidad de que ocurra un estado depende del estado inmediato anterior. Esta dependencia del estado anterior distingue a las cadenas de Markov de las series de estados independientes, como tirar una moneda al aire o un dado.

Los modelos de Markov están formados esencialmente por tres objetos:

$$\lambda = \langle Q, \Pi, A \rangle$$

los componentes de la cual se definen a continuación, donde q_t designa el estado en el instante t:

- Q es un conjunto no vacío cuyos elementos llamaremos *estados*, donde $Q = \{1, 2, \ldots, N\}$.
- Π es la probabilidad inicial para entrar al sistema:

$$\Pi = \{\pi_i\}, \ , \pi_i = \mathbb{P}[q_1 = i]$$

- A es la probabilidad de transición de un estado a otro.

$$\begin{aligned} A &= \{a_i j\} \\ a_i j &= \mathbb{P}[q_{t+1} = j | q_t = i],\ 1 \leqslant i, j \leqslant |N| \end{aligned}$$

La cadena de Markov se ha utilizado para calcular la probabilidad del patrón que estamos buscando en el cromosoma. Vamos a introducir un ejemplo para describir las propiedades que vamos a usar de la cadena de markov.

Usaremos la misma entrada que en el ejemplo 4.3.1. Los resultados de las diferentes probabilidades que salen son:

$\mathbb{P}[q_{t+1} = a \mid q_t = a]$	0.4
$\mathbb{P}[q_{t+1} = t \mid q_t = a]$	0.6
$\mathbb{P}[q_{t+1} = a \mid q_t = t]$	0.5
$\mathbb{P}[q_{t+1} = t \mid q_t = t]$	0.5

El conjunto de todos los posibles estados que un proceso puede adoptar se llama *espacio de estados*. Un espacio de estados puede ser finito (como el anterior, que sólo tiene cuatro) , numerable o no numerable.

3.4.1. Probabilidades de transición en un paso

Las probabilidades anteriores, son probabilidades de transición en un paso. La probabilidad de pasar del estado s_i al s_j suele representarse como p_{ij}. Si estas probabilidades son conocidas para todos los pares de estados podríamos ordenarlas como una matriz cuadrada. Esta matriz se llama *matriz de transición*. En nuestro ejemplo tendríamos:

$$A = \begin{pmatrix} p_{aa}{=}0.4 & p_{ac}=0 & p_{ag}=0 & p_{at}{=}0.6 \\ p_{ca}{=}0.0 & p_{cc}=0 & p_{cg}=0 & p_{ct}{=}0.0 \\ p_{ga}{=}0.0 & p_{gc}=0 & p_{gg}=0 & p_{gt}{=}0.0 \\ p_{ta}{=}0.5 & p_{tc}=0 & p_{tg}=0 & p_{tt}{=}0.5 \end{pmatrix}$$

Si $A(t) = a_{ij}(t)_{nxn}$ es independiente del tiempo entonces el proceso se llama *homogéneo* y las probabilidades de transición de estados son de la forma $a_{ij}(t) = \mathbb{P}[q_{t+1} = j|q_t = i]$ con las siguientes propiedades:

$$\begin{array}{ll} i) & 0 \leqslant a_{ij} \leqslant 1, \qquad 1 \leqslant i, j \leqslant N \\ ii) & \sum_{j=1}^{n} a_{ij} = 1, \quad 1 \leqslant i \leqslant N \end{array}$$

La condición fundamental de que sea una cadena de Markov establece que las probabilidades de transición y emisión dependen solamente del estado actual y no del pasado, esto es, $\mathbb{P}[q_{t+1} = j|q_t = i, q_{t-1} = k, \ldots] = \mathbb{P}[q_{t+1} = j|q_t = i] = p_{ij}(t)$.

Una cadena de Markov de primer orden es aquella que presenta las siguientes propiedades:

- Tanto estados como transiciones son discretos (no continuos).
- El número de estados es finito.
- Las transiciones dependen sólo del estado actual, no de los anteriores.
- Las probabilidades de transición se mantienen constantes en el tiempo

Así pues en nuestro caso, trabajaremos con una cadena de Markov de primer orden.

Las matrices de transición pueden representarse gráficamente en forma de diagramas de transición. Podemos ver el diagrama de transición para la matriz A de nuestro ejemplo en la figura 3.7.

Podemos observar que en las transiciones nulas no se representan los arcos, pero hemos dejado los dos estados no conectados para enfatizar el hecho que siempre trabajaremos con los cuatro estados símbolos de nuestro alfabeto.

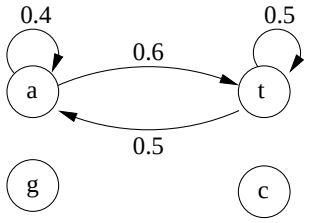


Figura 3.7: Ejemplo de diagrama de transiciones de una cadena de markov

3.4.2. Cálculo de la probabilidad de un patrón

Siguiendo con nuestro ejemplo, lo que nos interesa saber es la probabilidad que el patrón "aat" (y su complementario "att") aparezca en el cromosoma.

Para ello seguiremos el camino en el diagrama de transiciones que pase por cada uno de los símbolos que comprenden el patrón. El camino es:

$$\mathrm{a} \underset{0,4}{\longrightarrow} \mathrm{a} \underset{0,6}{\longrightarrow} \mathrm{t} \qquad\qquad \mathrm{a} \underset{0,6}{\longrightarrow} \mathrm{t} \underset{0,5}{\longrightarrow} \mathrm{t}$$

Multiplicando cada una de las probabilidades definidas en las transiciones obtenemos el valor que estamos buscando. Queda multiplicar el resultado por las probabilidades iniciales. En nuestro caso el vector de probabilidades en el estado 0 será la media de apariciones de cada símbolo en el cromosoma. Así tenemos:

$$\Pi_0 = \langle p_a = 0{,}5, p_c = 0{,}0, p_g = 0, p_t = 0{,}5 \rangle,$$

y por tanto las probabilidades de aparición de los patrones son:

$$\begin{aligned} \mathbb{P}[aat] &= 0{,}12 \\ \mathbb{P}[att] &= 0{,}15 \\ \mathbb{P}[aat \vee att] &= 0{,}27 \end{aligned}$$

3.4.3. Otros usos de las cadenas de Markov en aplicaciones biológicas

En la sección **??** en la página **??** aparecen diferentes usos de las cadenas de markov en aplicaciones biológicas.

En nuestro caso, otros usos que podemos observar son:

- Cálculo teórico de las apariciones de patrones con símbolos "comodín" (por ejemplo "a*t").
- Inferir cual es el patrón de longitud k que mas se repite teóricamente en el cromosoma.

- Inferir cual es el patrón de longitud k que menos se repite teóricamente en el cromosoma.
- Estimación de las probabilidades de aparición de las cadenas proteicas del cromosoma.

Dichos usos se han dejado abiertos para posible versiones posteriores del programa.

Debido a los usos que se han comentado, y a la verificación de resultados, la tabla de transiciones se presenta también como salida del programa.

Capítulo 4

Implementación de REPATT

4.1. Introducción

En esta sección se describe la implementación de la aplicación *REPATT*.

Los componentes del programa se realizaron en *C++* y su código esta disponible en la dirección de internet: `www.lsi.upc.es/~alggen/recerca/search/mrepatt/source`.

Se utilizaron las siguientes librerías de libre distribución:

- gd 1.8 : Librería para realizar las imágenes de posición de las ocurrencias. Su autor es Thomas Boutell.
- png 1.2.5: Librería para realizar gráficos en formato png. Sus autores son Guy Eric Schalnat, Andreas Dilger, Glenn Randers-Pehrson y otros.

Dejamos para el próximo capítulo la implementación de las imágenes para la localización de las ocurrencias encontradas (los *"zooms"*), puesto que merecía una especial atención. Como ya se verá, realizamos una nuevo programa a partir de *Repatt*, *Repatt_zoom*; una aplicación que tan solo realiza *zooms*. Esta especialización nos va a permitir mejorar el coste temporal, en comparación a una sola aplicación que implementara las dos funcionalidades de *Repatt* (el conteo de ocurrencias y el *zoom*) y permitiera escoger que funcionalidad realizar.

4.2. Búsqueda de patrones en un texto

Nuestro programa se enmarca dentro de los mas conocidos y antiguos problemas de la informativa, el *"string matching"*[1]: encontrar un patrón con alguna propiedad en una secuencia de símbolos.

Se suelen diferenciar los siguientes casos dentro de este problema:

[1]el autor ha visto traducciones al español de este término tales como "emparejar secuencias" o "casamiento de secuencias", no obstante se suele designar habitualmente el término en inglés para referirse a la búsqueda de una palabra en un texto.

- Single string Matching:[2] Un *string* es una secuencia de caracteres sobre un alfabeto finito (en nuestro caso $\{a, c, t, g\}$). El problema *Single string matching* es encontrar todas las ocurrencias de un *string* p dentro de un texto mucho mas largo que el patrón. Para solucionar este problema existen principalmente tres esquemas para atacarlo, en función de la manera en que se busque el patrón en el texto.

  - El primer esquema consiste en leer todos los caracteres del texto uno a uno, modificando en cada paso algunas variables que permitan identificar posibles ocurrencias. El algoritmo *Knuth-Morris-Pratt* [KMP77] , *Shift-Or* [BYG92] o *búsqueda con autómata determinista* son algunos de este tipo. Todos estos algoritmos tienen dos fases, la primera es un preproceso que tiene coste $\Theta(m+\sigma)$ donde m es la longitud del patrón y σ una constante diferente para cada algoritmo, la segunda fase es la búsqueda con coste $\Theta(n)$ donde n es la longitud del patrón.
  - El segundo esquema consiste en buscar el patrón p en una *ventana* que se desliza a lo largo del texto. Para cada posición de esta ventana buscamos de derecha a izquierda un sufijo de la ventana que corresponda a un sufijo de p. El principal algoritmo con este esquema es *Boyer-Moore* [BM77], pero generalmente es más lento que una de sus simplificación, *Horspool* [Hor80] . Estos algoritmos en el caso mejor son subliniales con coste $O(\frac{n}{m})$ y en el caso peor $O(3n)$. Dichos algoritmos consiguen la sublinialidad gracias a los saltos que pueden dar cuando los sufijos de la ventana no coinciden con los del patrón. Aún así,no funcionan bien cuando el tamaño del patrón es pequeño y hay una probabilidad alta de encontrarlo en el texto.
  - El tercer esquema es mas reciente y permite construir algoritmos mas eficientes en la práctica para p suficientemente grandes. Así como en el segundo esquema, la búsqueda se realiza de derecha a izquierda dentro de la ventana, en este esquema se busca el sufijo mas largo en la ventana que es un factor de p (un factor es cualquier subpalabra de p, es decir, sufijos, prefijos o infijos). El primer algoritmo que uso este método fue *BDM* [CCG$^+$94], pero cuando el patrón es corto es mas eficiente el *BNDM* [vR98] . Para patrones largos es mas eficiente el algoritmo *BOM* [ACR99] . Estos algoritmos en el caso mejor son subliniales con coste $O(\frac{n\log(m)}{m})$ y en el caso peor $O(mn)$. Como en el caso anterior, para patrones pequeños, no suelen trabajar bien.

  Exiten también otros esquemas, como los basados en hashing, pero no son tan eficientes.

- Multiple string Matching:[3] La diferencia entre este caso y el anterior es que ahora no tenemos un sólo patrón a buscar, sino que contaremos con conjunto $P = \{p_1, \ldots, p_l\}$ de patrones. La solución que se suele adoptar es la extensión de los es-

[2] emparejamiento de secuencias simple
[3] emparejamiento múltiple de secuencias

quemas anteriores para el caso múltiple. Así, siguiendo en la línea del caso anterior, tenemos:

- En la extensión del primer esquema aparecen los algoritmos de *Aho-Corasick* [AC75], *Multiple Shft-And* y *búsqueda con autómata determinista*. Todos ellos respetando el mismo coste que en las versiones simples.
- En la extensión del segundo esquema aparecen los algoritmos de *Commentz-Walter* [CW79], que no es muy eficiente en la práctica, el algoritmo *Set Horspool*, extensión de *Horspool*, es eficiente para conjuntos muy pequeños y alfabetos grandes. Podemos añadir también el algoritmo *Wu-Manber* [WM94] que mezcla este segundo esquema con el paradigma *hashing*. Dicho algoritmo es que esta implementado en la utilidad *agrep* de los sistemas *UNIX*. Es un algoritmo de búsqueda aproximada con k errores, pero podemos usarlo en una búsqueda exacta si igualamos k a 0. El coste del algoritmos es $\Theta(mn)$.
- Por último, en la extensión de los algoritmos del tercer esquema, encontramos el algoritmo *SBOM* que funciona bien cuando el tamaño mas pequeño del conjunto de patrones es grande. Por última está el algoritmo *Multiple BNDM*, descrito por M.Crochemore y W.Ritter en [CW94] que funciona bien para alfabetos grandes. Este último algoritmo inspirado en uno anterior llamado DAWG-MATCH [CCG$^+$99].

Nuestro problema pertenece al caso múltiple, puesto que buscaremos el patrón y su complementario. El algoritmo que hemos elegido es el de la búsqueda con autómata determinista, dado que, el alfabeto es muy pequeño, la longitud de los patrones se prevee pequeña (la mayoría de veces de 3 bases), y el conjunto de patrones a buscar es sólo de dos elementos. Con estas características, los algoritmos del segundo y tercer esquema no son tan eficientes. Por otro lado, de los algoritmos con este mismo esquema, se ha optado por la búsqueda con autómata por las siguientes razones:

- Se trata de buscar repeticiones consecutivas de patrones. En los siguientes apartados describiremos el problema de las ocurrencias consecutivas sobrepuestas. El uso de un autómata facilita la comprensión y mantenimiento de la solución propuesta.
- El uso de un autómata facilita la extensión del programa a buscar expresiones regulares.

4.3. Algoritmo usado para la búsqueda de patrones repetidos

El algoritmo usado para la búsqueda de patrones repetidos se basa en la utilización del autómata resultante de la unión de los autómatas aceptadores del patrón primario y complementario explicados en el apartada anterior.

Cuando nuestro autómata encuentra dos ocurrencias seguidas, dichas ocurrencias pueden encontrarse en alguna de las tres situaciones de la figura 4.1. Vemos que sólo en un caso se cumple que sean ocurrencias consecutivas.

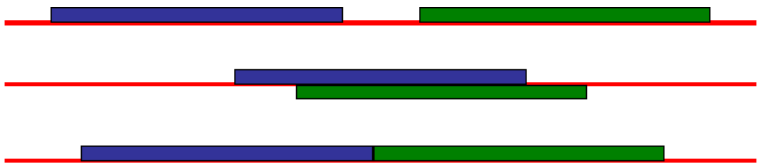


Figura 4.1: Posiciones relativas de dos ocurrencias

Es necesario guardar información sobre la última ocurrencia para saber si la siguiente que encuentra el autómata es consecutiva a la anterior. Pero dicho autómata no guarda memoria de los estados por los que va pasando, y no es capaz por sí mismo de saber en qué caso nos encontramos.

También podemos observar en la figura 4.1 que puede darse el caso que hayan ocurrencias sobrepuestas. Cuantas ocurrencias pueden estar sobrepuestas antes de formar dos ocurrencias consecutivas? Si nos fijamos en la figura 4.2, vemos que pueden haber como mucho l ocurrencias sobrepuestas entre dos ocurrencias consecutivas, donde l es la longitud del patrón.

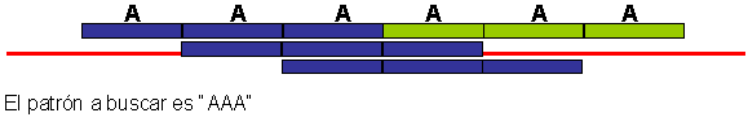


Figura 4.2: Posiciones relativas de varias ocurrencias

Dado que cada una de las ocurrencias sobrepuestas puede tener una ocurrencia consecutiva, debemos guardar información de cada una de estas ocurrencias, para determinar si las siguientes son o no consecutivas. Por tanto, vemos que necesitamos recordar las l últimas ocurrencias encontradas en el autómata (l es la longitud del patrón).

Podemos interpretar el cromosoma como una composición de l secuencias de grupos de l bases, dónde cada uno de estos grupos puede ser o no una ocurrencia del patrón. Dicha idea queda reflejada en la figura 4.3

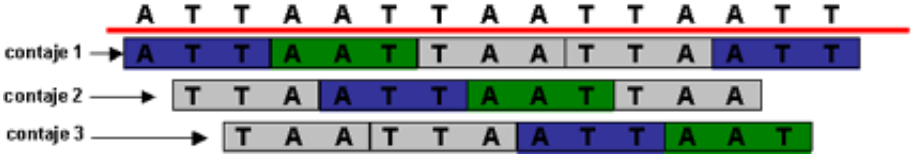


Figura 4.3: Interpretación del cromosoma como composición de secuencias

Entonces, para llevar el contaje de cada una de estas l-secuencias el algoritmo sólo se necesita recordar las ocurrencias consecutivas en curso en el momento de tratar un

nuevo grupo. Así, por ejemplo, si el último grupo tratado ha formado una repetición p^i y el grupo actual es una ocurrencia de p, entonces apuntamos que las ocurrencias consecutivas en curso para la l-secuencia son p^{i+1}. Si el siguiente grupo dejase de formar repeticiones del patrón p, se contabiliza p^{i+1} y se inicia las ocurrencia consecutivas en curso como p^0. Dado una l-secuencia, las ocurrencias consecutivas en curso y el grupo actual, el algoritmo se puede encontrar ante dos posibles situaciones. En la figura 4.4 vemos los posibles casos de la situación en que el grupo actual sigue de una repetición consecutiva en curso ,siendo el patrón a buscar "ATT".

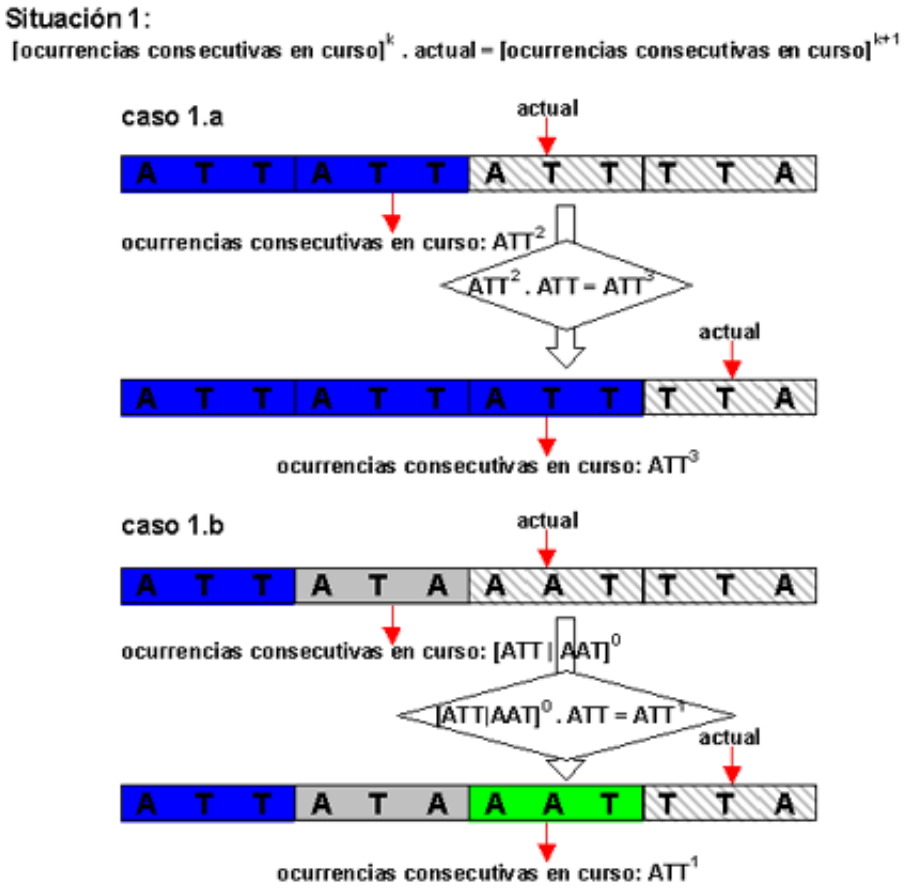


Figura 4.4: Situación 1. El grupo actual sigue de una repetición consecutiva en curso

Podemos ver que en los dos casos, no se contabilizan las ocurrencias en curso, esperaremos que se llegue a la finalización de las repeticiones del patrón para contabilizar una ocurrencia no acumulada de orden k. En el caso $1.b$ estamos usando la propiedad $p^0 = p_c^0 = \langle\rangle$ para inicializar las ocurrencias consecutivas en curso, sin tener que realizar un caso especial. En la figura 4.5 podemos observar los dos casos de la situación en que el grupo actual no sigue de una repetición consecutiva en curso, siendo el patrón a buscar

"ATT".

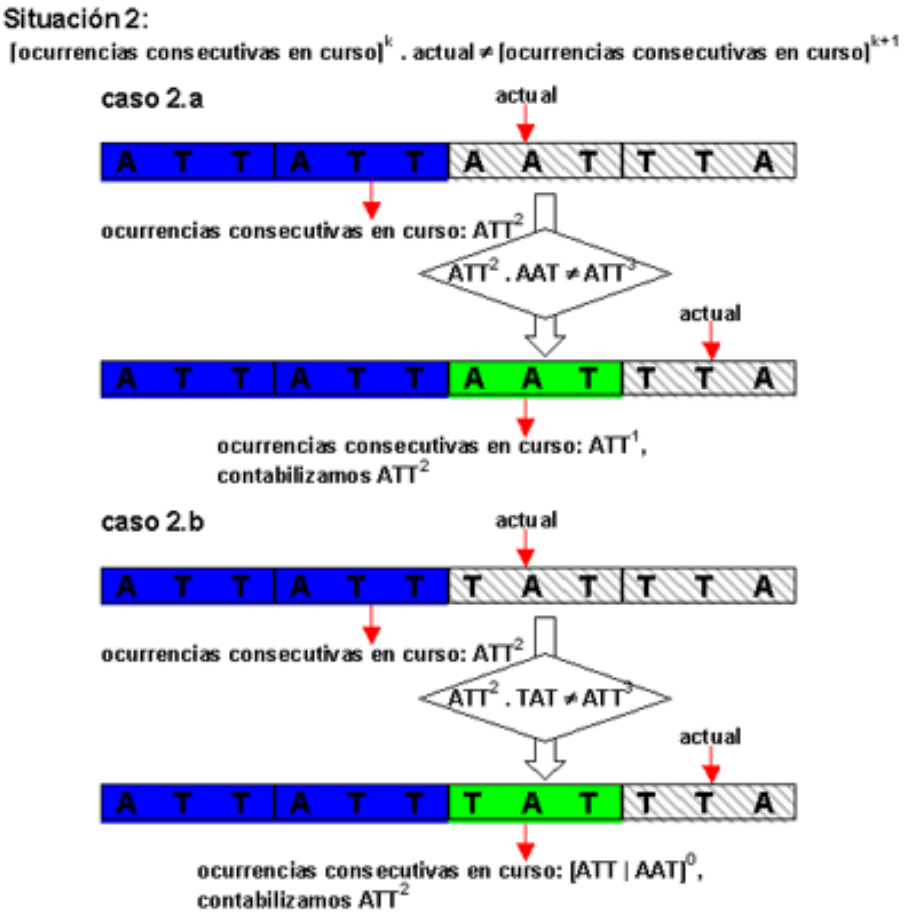


Figura 4.5: Situación 2. El grupo actual no sigue de una repetición consecutiva en curso

En estos casos si que contabilizamos la ocurrencia en curso antes de inicializarla. Como se ha dicho, contabilizaremos las ocurrencias no acumuladas. Sólo podemos saber que una ocurrencia es única cuando el siguiente grupo *rompe* la secuencia de patrones repetidos.

Dada la interpretación del cromosoma como composición de l-secuencias, el algoritmo entenderá cada base como final de un grupo de la l-secuencia i. Para llevar la l-contabilidad se utiliza una cola de longitud l. A cada paso del bucle que recorre las bases (o grupos) se desencola las ocurrencia consecutivas en curso, se actualiza el elemento y se vuelve a encolar. De esta manera, el primer elemento de la cola siempre pertenece a la l-secuencia del grupo en curso.

Puesto que el autómata con el que trabajamos acepta tanto los patrones primarios como complementarios, se ha modificado el algoritmo de unión 3.4, explicado en el capítulo anterior, para que nos indique cual es el patrón que se ha aceptado. El autómata

nos devuelve un patrón si el estado donde ha avanzado es final, o una secuencia vacía si el estado no es final. Si llega al estado final del patrón no complementario devuelve dicho patrón, mientras que si llega al estado final del patrón complementario, devuelve el patrón complementario.

En el algoritmo, que se presenta en la figura 4.6, se han utilizado las siguientes funciones:

- crear_automata_union_patron_y_complementario() : Se crean el autómata que fruto de la unión entre el autómata del patrón de entrada y su complementario.
- crea_cola_con_ocurrencias_vacias() : función que crea la cola que mantiene la *l-contabilidad* de las ocurrencias. Se encolan l secuencias vacías, donde l es la longitud del patrón:
$$ocurrencias\_en\_curso \leftarrow \langle\langle\rangle, \ldots^{l}, \langle\rangle\rangle$$
- automata(c) : avanza al siguiente estado del autómata y nos devuelve un patrón si el estado donde ha avanzado es final, o una secuencia vacía si el estado no es final. Si llega al estado final del patrón no complementario devuelve dicho patrón, mientras que si llega al estado final del patrón complementario, devuelve el patrón complementario.
- tratamos_ocurrencia(o^k) : procedimiento que se encarga del tratamiento de una ocurrencia de orden k. Es aquí donde se hace el contaje y se recogen las posiciones de la ocurrencia. La precondición de la función es que la ocurrencia o^k sea una ocurrencia única. Si k es 0 no se hace nada.

La versión que se presenta es la versión final del algoritmo.

Básicamente la diferente entre ésta y las que se realizaron previamente es que el algoritmo presentado realiza el conteo sobre la ocurrencias únicas y no sobre las acumuladas. Al contabilizar sólo las ocurrencias únicas realizamos menos operaciones de inserción en la tabla de ocurrencias. Si contásemos las acumuladas, perderíamos mucho tiempo en ir añadiendo cada vez una ocurrencia en la tabla. Aunque la inserción tiene tiempo constante (veremos que en la realización de la imagen no es así), los cromosomas tienen tamaños muy grandes y cualquier operación que introduzcamos, aunque sea de tiempo constante, se traduce en una variación del tiempo total importante.

A continuación veremos el funcionamiento con un ejemplo.

Ejemplo 4.3.1. Consideramos el patrón "aat" y su complementario "att". El autómata que haremos servir en el algoritmo es la unión de los dos autómatas que aceptan sendos lenguajes, y aparecen en la tabla 3.3 y en la figura 3.6.

Como cromosoma consideraremos la secuencia "aattattaat" de longitud 10.

Podemos ver los datos de entrada que se han explicado en la tabla 4.1.

Inicializamos la cola *ocurrencias_en_curso* con l secuencias vacías.

ocurrencias_en_curso	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Analicemos qué ocurre en el algoritmo símbolo a símbolo.

```
procedure mrepatt(in cromosoma = ⟨c_1, ..., c_n⟩,   in patron = ⟨p_1, ..., p_m⟩) is

    automat ← crear_automata_union_patron_y_complementario()
    ocurrencias_en_curso ← crea_cola_con_ocurrencias_vacias()
    i ← 1

    {Inv : Se han tratado todas las ocurrencias unicas del cromosoma hasta la posicion i - 1 }
    while i ⩽ n do
        ocurrencia_actual ← automata(c_i)
        en_curso^k ← desencola(ocurrencias_en_curso)
        if ocurrencia_actual · en_curso^k = en_curso^{k+1} then
            encolar(ocurrencias_en_curso, en_curso^{k+1})
        else
            tratamos_ocurrencia(en_curso^k)
            encola(ocurrencias_en_curso, ocurrencia_actual)
        end if
        i ← i + 1
    end while

    {Inv : Se han tratado todas las ocurrencias unicas del cromosoma hasta  }
    { la posicion n - |ocurrencias_en_curso| }
    while no_vacia(ocurrencias_en_curso) do
        en_curso^k ← desencola(ocurrencias_en_curso)
        tratamos_ocurrencia(en_curso^k)
    end while

end
```

Figura 4.6: Algoritmo usado para la búsqueda de patrones repetidos

patrón primario	"aat"
patrón complementario	"att"
cromosoma	"aattattaat"

Cuadro 4.1: datos de entrada

paso 1. a attattaat

1. *ocurrencia_actual* ← ⟨⟩ : el autómata no ha encontrado un estado final, y nos devuelve una secuencia vacía.
2. $en\_curso^k \leftarrow \langle\rangle$: por tanto k es 0.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las

ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $\langle\rangle \cdot \langle\rangle = [aat|att]^1$ y vemos que no se cumple.

4. Tratamos las ocurrencias en curso, que en este caso no se contabilizaran, puesto que es la cadena vacía.

5. Encolamos la ocurrencia actual:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle} \boxed{\langle\rangle} \boxed{\langle\rangle}$$

paso 2. a a ttattaat Este paso es igual que el paso 1.

paso 3. aa t tattaat

1. $ocurrencia\_actual \leftarrow aat$: el autómata ha encontrado un estado final, y nos devuelve el patrón.

2. $en\_curso^k \leftarrow \langle\rangle$: por tanto k es 0.

3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $aat \cdot \langle\rangle = [aat|att]^1$ y vemos que se cumple con aat^1.

4. Encolamos la nueva ocurrencia consecutiva en curso:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle} \boxed{\langle\rangle} \boxed{aat^1}$$

paso 4. aat t attaat

1. $ocurrencia\_actual \leftarrow att$: el autómata ha encontrado un estado final, y nos devuelve el patrón.

2. $en\_curso^k \leftarrow \langle\rangle$: por tanto k es 0.

3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $att \cdot \langle\rangle = [aat|att]^1$ y vemos que se cumple con att^1.

4. Encolamos la nueva ocurrencia consecutiva en curso:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle} \boxed{aat^1} \boxed{att^1}$$

paso 5. aatt [a] ttaat

1. $ocurrencia\_actual \leftarrow \langle\rangle$: el autómata no ha encontrado un estado final, y nos devuelve una secuencia vacía.
2. $en\_curso^k \leftarrow \langle\rangle$: por tanto k es 0.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $\langle\rangle \cdot \langle\rangle = [aat|att]^1$ y vemos que no se cumple.
4. Tratamos las ocurrencias en curso, que en este caso no se contabilizaran, puesto que es la cadena vacía.
5. Encolamos la ocurrencia actual:

$$ocurrencias\_en\_curso \quad \boxed{aat^1 \mid att^1 \mid \langle\rangle}$$

paso 6. aatta [t] taat

1. $ocurrencia\_actual \leftarrow \langle\rangle$: el autómata no ha encontrado un estado final, y nos devuelve una secuencia vacía.
2. $en\_curso^k \leftarrow aat^1$: por tanto k es 1.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $\langle\rangle \cdot aat^1 = aat^2$ y vemos que no se cumple.
4. Tratamos las ocurrencias en curso y se contabiliza una ocurrencia aat de orden 1.
5. Encolamos la ocurrencia actual:

$$ocurrencias\_en\_curso \quad \boxed{att^1 \mid \langle\rangle \mid \langle\rangle}$$

paso 7. aattat [t] aat

1. $ocurrencia\_actual \leftarrow att$: el autómata ha encontrado un estado final, y nos devuelve el patrón.
2. $en\_curso^k \leftarrow att^1$: por tanto k es 1.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $att \cdot att^1 = att^2$ y vemos que se cumple.
4. Encolamos la nueva ocurrencia consecutiva en curso:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle \mid \langle\rangle \mid att^2}$$

paso 8. aattatt [a] at

1. *ocurrencia_actual* $\leftarrow \langle\rangle$: el autómata no ha encontrado un estado final, y nos devuelve una secuencia vacía.
2. $en\_curso^k \leftarrow \langle\rangle$: por tanto k es 0.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $\langle\rangle \cdot \langle\rangle = [aat|att]^1$ y vemos que no se cumple.
4. Tratamos las ocurrencias en curso, que en este caso no se contabilizaran, puesto que es la cadena vacía.
5. Encolamos la ocurrencia actual:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle} \boxed{att^2} \boxed{\langle\rangle}$$

paso 9. aattatta [a] t

1. *ocurrencia_actual* $\leftarrow \langle\rangle$: el autómata no ha encontrado un estado final, y nos devuelve una secuencia vacía.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $\langle\rangle \cdot \langle\rangle = [aat|att]^1$ y vemos que no se cumple.
4. Tratamos las ocurrencias en curso, que en este caso no se contabilizaran, puesto que es la cadena vacía.
5. Encolamos la ocurrencia actual:

$$ocurrencias\_en\_curso \quad \boxed{att^2} \boxed{\langle\rangle} \boxed{\langle\rangle}$$

paso 10. aattattaa [t]

1. *ocurrencia_actual* $\leftarrow aat$: el autómata ha encontrado un estado final, y nos devuelve el patrón.
2. $en\_curso^k \leftarrow att^2$: por tanto k es 2.
3. Ahora comprovamos si la ocurrencia actual es una ocurrencia consecutiva de las ocurrencias consecutivas en curso. Nos preguntamos si $ocurrencia\_actual \cdot en\_curso^k = en\_curso^{k+1}$, por tanto, si $aat \cdot att^2 = att^3$ y vemos que no se cumple.
4. Tratamos las ocurrencias en curso, y contabilizamos una ocurrencia *att* de orden 2.
4. Encolamos la nueva ocurrencia consecutiva en curso:

$$ocurrencias\_en\_curso \quad \boxed{\langle\rangle} \boxed{\langle\rangle} \boxed{aat^1}$$

paso 11. Contabilizar las ocurrencias que quedan dentro de la cola. Vamos vaciando la cola hasta que queda vacia, y tratamos cada ocurrencia. En nuestro caso contabilizaremos sólo una ocurrencia *aat* de orden 1.

Resultados En total hemos contabilizado las siguientes ocurrencias únicas:

k	att^k	aat^k
1	0	2
2	1	0

4.4. Cálculo de las ocurrencias teóricas

Para calcular las ocurrencias teóricas se han utilizado dos métodos diferentes. Se han contrastado los resultados y finalmente se ha optado por uno de ellos.

método 1 El primer método que se pensó parte como base en el cálculo de las probabilidades de aparición de cada símbolo. Obteniendo estas probabilidades se procede a calcular la probabilidad de aparición de los dos patrones, multiplicando las probabilidades de cada símbolo de éstos. Para calcular el número de ocurrencias teóricas se multiplica la probabilidad de cada patrón por el número de grupos de longitud la del patrón (con lo que nos quedan las ocurrencias teóricas del patrón primario y del complementario) y se suman. La fórmula del cálculo es:

$$no_k = (\mathbb{P}[p^k] + \mathbb{P}[p_c^k]) * (|\mathcal{C}| - k * |p| + 1)$$

Como puede verse, el número de ocurrencias teóricas es básicamente el producto del patrón por la longitud del cromosoma. Como se busca el patrón primario y el complementario, la probabilidad del patrón es de hecho la suma de los dos probabilidades.

En este método, se suponen iguales la probabilidad de aparición de cada nucleótido, es decir, se supone equiprobabilidad. Por tanto, la probabilidad de aparición de cada patrón es:

$$\mathbb{P}[patron^k] = \frac{1}{4^{|patron|+k}}$$

método 2 En el segundo método se usan las cadenas de markov descritas en la sección 3.4. A medida que se lee el cromosoma se genera la matriz de transición y al finalizar la lectura se obtiene la probabilidad de aparición para cada una de las repeticiones de los patrones encontrados.

La fórmula seguida para encontrar la probabilidad del patrón de orden k es:

$$\mathbb{P}[p^k] = \mathbb{P}[p(1)] * \prod_{i=1}^{k*|p|-1} (\mathbb{P}[p(i+1)|p(i)])$$

$$\mathbb{P}[p(i+1)|p(i)]$$

Una vez calculadas dichas probabilidades se obtiene el número de ocurrencias de orden k, no_k, siguiendo la misma fórmula que el método 1.

Podemos ver en la tabla 4.2 una comparativa para el patrón "cg" y el cromosoma 21 del hombre.

orden	ocurrencias Método 1	ocurrencias Método 2	ocurrencias reales
1	2,840,280	739,444	739,444
2	118,701	8,233.72	19,756
3	4,960.75	91.6825	1,224
4	207.32	1.02089	184
5	8.66433	$1{,}14 * 10^{-2}$	64
6	$3{,}62 * 10^{-1}$	$1{,}27 * 10^{-4}$	28
7	$1{,}51 * 10^{-2}$	$1{,}41 * 10^{-06}$	10
8	$6{,}32 * 10^{-4}$	$1{,}57 * 10^{-08}$	2

Cuadro 4.2: Tabla comparativa de los dos métodos para calcular las ocurrencias teóricas.

El *método 1* asume que los símbolos del alfabeto son independientes y define el valor $a \in \Sigma$ con probabilidad la frecuencia de aparición de a en el cromosoma. En la práctica este modelo es demasiado simple como se dice en [RSM00] y es preferible trabajar con cadenas de markov de al menos orden 1. Uno de los primeros trabajos de cálculo de frecuencias de patrones de dna que usaron cadenas de markov fue [Cow91]. En su trabajo se preguntaba como saber que el número de dinucleótidos[4] era estadísticamente significativo. En el defiende claramente un modelo de cadenas de Markov ya que el modelo simple es inválido para representar cadenas de nucleótidos.

Por todos estos trabajos y por los que se han derivado de ellos, hemos elegido el método 2 para calcular el número esperado de apariciones de un determinado patrón.

Para comparar el número esperado de repeticiones con el real se ha elegido el cociente resultante de la división entre las ocurrencias reales y las teóricas [5] $\frac{|O^k|}{no_k}$. En los trabajos anunciados anteriormente se utilizaron fórmulas estadísticas mas complejas. No obstante la mayoría de trabajos que se han leído están orientados al descubrimiento de patrones biológicamente relevantes. En nuestro caso, según el criterio del co-director del proyecto, era mas relevante conocer este cociente, puesto que el objetivo no es el descubrimiento de nuevos patrones sino el análisis de algunos ya conocidos.

[4] un "dinucleótido" es un par de nucleótidos consecutivos, por tanto hay 16 posibles

[5] ver sección 3

Capítulo 5

Implementación del "Zoom": Gráfico de posiciones

Para hacer el "zoom" se ha realizado una inmersión de eficiencia en el algoritmo de búsqueda. No obstante éste tiene peculiaridades debido a los requisitos que se pedían, y no siempre que se requiere hacer un "zoom" se pide hacer un conteo de las repeticiones en la búsqueda del patrón.

Estas peculiaridades nos han llevado a la división de las funcionalidades en dos programas, los cuales se basan en los mismos componentes y librerías software. Los programas se llaman *Repatt* y *Repatt_zoom*. Podemos ver las diferencias en la tabla 5.1.

Repatt	*Repatt_zoom*
Realiza operaciones de conteo	No realiza las operaciones de conteo
Comprende siempre todo el rango del cromosoma	El rango lo define el usuario
Obtiene un "zoom" del cromosoma completo	Obtiene un zoom de la región definida

Cuadro 5.1: Diferencias funcionales entre los programas *Repatt* y *Repatt_zoom*

La decisión de separarlo en dos programas y no realizar uno único con diferentes opciones se ha hecho por motivos de eficiencia explicados en la sección 5.5

En los siguientes sub-apartados hablaremos de como se ha implementado el "zoom" independientemente de los dos programas enunciados, puesto que como se ha dicho, los componentes software son básicamente los mismos. Por último hablaremos de la división de los dos programas, por qué se ha hecho y las diferencias algorítmicas entre éstos.

5.1. Introducción

Como se ha explicado en el análisis de requerimientos en la sección 2.1.3, se ha de realizarse un módulo capaz de "navegar" a través del cromosoma mostrando las posiciones de cada ocurrencia, el orden a la que pertenecen y la dirección, además se ha de diferenciado entre las ocurrencias únicas y las sobrepuestas.

Podemos ver un ejemplo de éste gráfico en la figura 5.1.

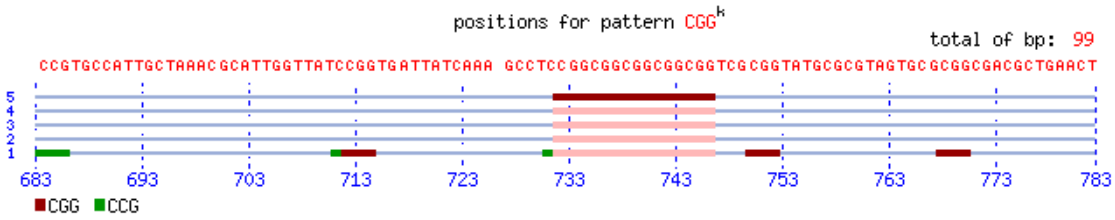


Figura 5.1: Ejemplo de "Zoom" para la bacteria *escherichia coli* buscando el patrón "CGG" en la región que comprende desde la base 3.452.683 a la 3.452.782

Como se puede observar en el ejemplo, para cada orden encontrado se muestra una línea horizontal de color azul, donde aparecen las ocurrencias del patrón base como líneas horizontales mas dobles de tonalidad roja y las complementarias de tonalidad verde.

Las líneas rojo oscuro y verde oscuro corresponden a las ocurrencias únicas y las de tonalidades claras a las acumuladas (que también llamaremos residuos o sobrepuestas).

En la parte superior aparecen los nucleótidos como se detallaba en el análisis de requerimientos, y en la parte inferior aparecen las posiciones (la parte de las centenas) a partir de la posición inicial.

Como puede observarse, las líneas del el patrón complementario sólo se alargan en el primer nucleótido de la repetición. Eso es así porque se solapan las repeticiones del patrón base y del complementario. En esa circunstancia se ha optado por mostrar sólo la línea de uno de ellos, en este caso la del patrón base.

5.2. Métodos para su implementación

Los métodos que se han estudiado para su implementación han sido dos. Antes de explicarlas tenemos que tener presente cómo va a utilizar el biólogo el programa.

Comportamiento que debe mostrar el componente de Zoom

1. Primeramente se debe mostrar el mapa de todo el cromosoma.
2. A partir del mapa global se harán "zooms" de las regiones que el biólogo crea mas interesantes.
3. A cada zoom se le puede volver a aplicar una selección, hasta llegar a una selección con longitud mínima de 100 bases.

Método A. Eficiencia temporal

Este método consiste en guardar en un fichero las posiciones donde han aparecido las repeticiones y su dirección. Posteriormente cuando se realice un "zoom" en una región

se trabaja con el fichero de posiciones debidamente indexado.

Con este método sólo recorremos una vez el cromosoma, pero el precio que conlleva es guardar una estructura con la información en memoria secundaria. El coste espacial es muy alto.

Método B. Eficiencia espacial

El método B es la opuesta a la anterior. Cada vez que se realiza un zoom recorremos la región que pide el usuario en el cromosoma.

Este método no es tan eficiente como la "A", pero no requiere un fichero que guarde la todas posiciones de los cromosomas.

Elección del Método

El método elegida a sido el "B", en busca de la eficiencia espacial. El problema que conlleva guardar un fichero con la información de las posiciones es que dicho fichero tiene un tamaño del mismo orden que el cromosoma.

Puesto que trabajaremos con cromosomas cuyos tamaños superan en general los 50 Megas, y tenemos que guardar las posiciones de todos los cromosomas con los que se estén trabajando en un mismo instante, la suma de esos tamaños hace que no sea posible aplicar el método que hace eficiente el coste temporal.

No obstante, como ya se ha visto, el coste temporal del algoritmo de búsqueda, en el cual nos basaremos para realizar el "Zoom", es lineal.

Una de las propiedades del comportamiento del componente de "Zoom" es que las regiones que elegirá el usuario habitualmente serán menores o iguales a la mitad del cromosoma. A partir de esa longitud en la mayoría de casos, el tiempo absoluto que se requiere para hacer un "Zoom" es de menos de un segundo que entendemos asumible.

5.3. Implementación del "Zoom"

Nuestro objetivo es realizar la imagen en un solo recorrido de la secuencia de entrada. La técnica que emplearemos para llegar al objetivo está supeditada a las propiedades siguientes:

- El número máximo de orden se conoce cuando finaliza el recorrido de la secuencia.
- El número de bases que componen el cromosoma se conoce cuando finaliza el recorrido. [1]

Las restricciones que debe cumplir nuestro algoritmo y están relacionadas con las propiedades descritas son las siguientes:

- La altura de la imagen depende del orden máximo de repeticiones encontradas.

[1] Podemos saber los caracteres totales del cromosoma a priori pero no el número total de bases puesto que el fichero se compone de otros caracteres además de los nucleótidos.

- Se muestra en la imagen el número de bases encontradas en la región que se ha buscado.
- Para cada orden se muestra en la imagen una línea horizontal que representa la región de la secuencia que se ha pedido.
- Estas líneas horizontales están ordenadas de forma descendente de mayor a menor, siendo la mayor la que ha de aparecer en la parte superior.
- No podemos guardar una estructura en memoria (primaria y/o secundaria) con la lista de posiciones encontradas.
- Sólo podemos recorrer una vez la secuencia de entrada.

Se debe encontrar una forma de realizar la imagen que cumpla las restricciones a pesar de las propiedades. El hecho de no guardar la lista de posiciones supone ir construyendo la imagen a medida que se lee el fichero de entrada. Debemos crear una imagen vacía y ir dibujando en ella a medida que se avanza en el cromosoma. Estas política conllevan algunos problemas prácticos al trabajar con la librería de gráficos que hemos utilizado:

a) Como la anchura es fija definimos el *ratio* $\frac{anchura(pixeles)}{numero\ de\ bases}$.

b) Cuando creamos la imagen se definen sus dimensiones : altura y anchura. *¿Qué altura definimos? ¿Cuantas bases hay en nuestra región?*

Solución al problema "a)"

En éste caso se ha optado por cambiar el denominador del ratio *número de bases* por otro valor conocido a priori del algoritmo. El ratio será $\frac{anchura(pixeles)}{numero\ de\ caracteres}$. Así pues, no mostraremos en la imagen tan solo las bases que componen nuestro alfabeto $\Sigma = \{a, c, g, t, N\}$ sino que mostremos cualquier símbolo que aparezca en el fichero de entrada. Eso supone añadir los siguientes:

- ' ' : Un espacio en blanco para cada retorno de carro.
- 'X' : Es carácter representa cualquier símbolo que no este dentro de nuestro alfabeto. En concreto lo usaremos para cada carácter que englobe una cadena de identificación fasta.

Podemos ver un ejemplo de estos caracteres en el figura 5.2. En el ejemplo podemos comprobar la aparición de cada uno de los caracteres que pueden aparecer.

Solución al problema "b)"

Ahora que sabemos que ratio tenemos podemos ir dibujando las líneas horizontales mientras recorremos el cromosoma, no obstante no podemos saber la altura de la imagen a priori.

Hay dos soluciones que podemos aplicar.

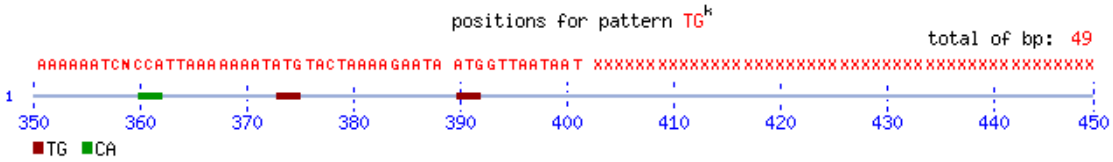


Figura 5.2: Ejemplo de "Zoom" para la ameba *Dicty.discoideum: cromosoma I* buscando el patrón "TG" en la región que comprende desde la base 350 a la 449

La primera y mas simple es definir una altura suficientemente grande como para dar cabida a cada uno de los ordenes que vayan apareciendo. No obstante sólo podría servir si el orden mínimo lo constituyera la línea superior, y no es el caso.

La segunda solución, y la que se ha aplicado consiste en definir una lista ordenada de imágenes. Cada imagen corresponde a una sola línea que representa un orden. La ordenación es de orden inferior a superior. Cuando acabamos de leer el cromosoma creamos la imagen resultado recorremos la lista en orden inverso y vamos haciendo una copia de cada imagen de la lista en la región de la imagen resultado que concierne.

¿Cuando añadir una ocurrencia a la imagen?

A partir del algoritmo de búsqueda presentado en la figura 4.6, vamos a introducir la operación de dibujar una ocurrencia única en la imagen. El algoritmo llamará a esta operación cada vez que encuentre una ocurrencia única, en la función de tratamiento.

Como se puede observar en los ejemplos de las imágenes, para cada ocurrencia única en el orden k se dibuja también la ocurrencia con un color mas claro en las líneas horizontales correspondientes a cada orden menor a k.

Por tanto el coste de añadir a la línea una ocurrencia encontrada es $O(k)$, siendo k el orden de la repetición de la ocurrencia a añadir. Este coste lo hemos reducido notablemente eliminando las líneas horizontales que hemos llamado líneas colapsadas.

Una línea de la imagen se colapsa cuando todos sus puntos corresponden a mas de una coincidencia. Las líneas colapsadas no aportan información al biólogo, puesto que al estar totalmente pintadas del color de alguna de las dos ocurrencias, no se puede apreciar la localización de los intervalos.

Para eliminar este tipo de líneas durante el proceso de inserción de ocurrencias (sin esperar a leer todo el cromosoma), hemos creado una restricción de ocurrencias máximas a dibujar en cada línea. Cuando dibujamos una nueva ocurrencia y llegamos a ese máximo, declaramos esa línea y todas las que corresponden a órdenes inferiores como colapsadas.

Si una línea está colapsada la borramos de la lista de líneas a representar en la imagen.

5.4. Coste del algoritmo de búsqueda con el zoom

5.4.1. Cálculo del coste asintótico

A continuación vamos a proceder al cálculo del coste que tiene el algoritmo de búsqueda añadiendo la creación de la imagen.

Presentaremos los siguientes conceptos [2] y sus definiciones que nos serán de utilidad para el cálculo:

- *Ocurrencia*:

$$o = (i = \langle l, r\rangle, s = \langle b_l, \ldots, b_r\rangle),\ \mathcal{C} = \alpha s \beta \quad |\alpha|, |\beta| \geqslant 0 \quad |s| > 1$$

  Cada ocurrencia es una coincidencia dentro del cromosoma con el patrón primario o secundario. Formalmente cada ocurrencia esta compuesta por el intervalo formado por la posición inicial y final donde ha aparecido la coincidencia dentro del cromosoma , y por la propia coincidencia. Por motivos de claridad nos referiremos a cada elemento del intervalo como o_l, o_r y a la subpalabra como o_s.

- *Conjunto de ocurrencias*:

$$O = \{o \,|\, o_s \in \mathcal{L}\}$$

  A este conjunto también lo llamaremos ***conjunto de ocurrencias acumuladas***. Recordemos que $\mathcal{L}$ es el lenguaje de búsqueda.

- *Conjunto de ocurrencias de orden k* :

$$O^k = \{o \,|\, o_s \in \mathcal{L}_k\}$$

  El conjunto de ocurrencias de orden k es el subconjunto de O cuyas subpalabras de cada ocurrencia coinciden con alguno de los dos patrones K veces concatenados.

- *Conjunto de ocurrencias únicas de orden k*: El conjunto de ocurrencias únicas de orden k es una subconjunto de las ocurrencias de orden k definido como sigue:

$$U^k = \{o \,|\, o \in O^k\ , (\langle i, j\rangle, w) \notin O^{k+1} \quad \forall i, j \quad i \leqslant o_l \leqslant o_r \leqslant j\}$$

  Es decir, las ocurrencias únicas o^k no son subpalabra de una ocurrencia de orden superior.

- *Coste de insertar una ocurrencia única en la imagen k*:

$$\forall i \in 1, \ldots, |\mathcal{C}| \quad h(i) \begin{cases} 0 & si \quad \forall k, o = (\langle i, j\rangle, s) \notin U^k \\ k & si \quad \exists k, o = (\langle i, j\rangle, s) \in U^k \end{cases}$$

  La función se define para cada una de las posiciones del cromosoma. Si en una posición no existe una ocurrencia única el coste de añadir es nulo (puesto que no

[2] algunos de ellos ya se presentaron en la sección 3. No obstante se han vuelto ha introducir por motivos de claridad

llamamos a la función de añadir una ocurrencia en la imagen). Cuando añadimos una ocurrencias única el coste debe ser el de añadir la dicha ocurrencia y todas sus ocurrencias acumuladas. Dado que el coste de añadir un punto a la imagen es constante, el coste de añadir una ocurrencia única es el que sale de recorrer el bucle de ir añadiendo sus ocurrencias acumuladas. Dicho bucle tiene coste igual al orden de la ocurrencia única que estamos añadiendo.

Dado que el coste de insertar una ocurrencia única depende del orden de ésta, y no lo conocemos con exactitud, trabajaremos con la esperanza estadística de su función. Así, tenemos que el coste asintótico es:

$$\mathbb{C} = \Theta(\sum_{i=1}^{|\mathcal{C}|}(1 + E[h(i)])) = \Theta(n + \sum_{i=1}^{|\mathcal{C}|} E[h(i)]) = \Theta(n + nE[h(i)])$$

A continuación procederemos al cálculo de la esperanza:

$$E[h(i)] = \sum_{k\geqslant 1}(k * \mathbb{P}[o \in U^k]) = \sum_{k\geqslant 1}(k * \mathbb{P}[o \in O^k] * \prod_{j=1}^{k-1} \mathbb{P}[o \notin O^j])$$

Como se puede observar en la fórmula descrita, la probabilidad de una ocurrencia única o de orden k es igual a la probabilidad de que o sea una ocurrencia acumulada de orden k i no lo sea para todo orden inferior de k.

Para facilitar el cálculo acotaremos a una cota superior el término correspondiente a la probabilidad de no ser una ocurrencia acumulada de orden inferior:

$$\prod_{j=1}^{k-1} \mathbb{P}[o \notin O^j] = \prod_{j=1}^{k-1} 1 - \mathbb{P}[o \in O^j] \leqslant 1 - \mathbb{P}[o \in O^1] \quad \forall k$$

Vemos que la demostración de la desigualdad es directa, pues se trata de un producto de probabilidades, y cualquier término multiplicado por una probabilidad da como resultado un término menor. Substituyendo en la fórmula de la esperanza tenemos:

$$E[h(i)] \leqslant \sum_{k\geqslant 1}(k * \mathbb{P}[o \in O^k] * (1 - \mathbb{P}[o \in O^1])) \leqslant (1 - \mathbb{P}[o \in O^1]) * \sum_{k\geqslant 1}(k * \mathbb{P}[o \in O^k])$$

Vemos que queda por resolver una serie del tipo :

$$\sum_{n\geqslant 1} n * x^n = \frac{x}{(1-x)^2}$$

La resolvemos y tenemos:

$$E[h(i)] \leqslant (1 - \mathbb{P}[o \in O^1]) * \frac{\mathbb{P}[o \in O^1]}{(1 - \mathbb{P}[o \in O^1])^2} \leqslant \frac{\mathbb{P}[o \in O^1]}{1 - \mathbb{P}[o \in O^1]}$$

Substituyendo en el coste por la cota superior encontrada tenemos:

$$\begin{aligned}
\mathbb{C} = \Theta(n(1 + E[h(i)])) &\simeq \\
\Theta(n(1 + \frac{\mathbb{P}[o \in O^1]}{1 - \mathbb{P}[o \in O^1]})) &\simeq \\
\Theta(n(\frac{1}{1 - \mathbb{P}[o \in O^1]})) &\simeq \\
\Theta(\mathbb{P}[o \notin O^1]^{-1} n) &\sim \\
\Theta(n)
\end{aligned}$$

Vamos a proceder a calcular el coste en varios casos concretos.

Primero definimos la probabilidad de que o pertenezca al conjunto de ocurrencias acumuladas de orden 1. Vamos a suponer que en el alfabeto del cromosoma se compone de las cuatro bases y existe equiprobabilidad de aparición. Sea l la longitud del patrón tenemos:

$$\mathbb{P}[o \in O^1] = 4^{-l}$$

Por tanto el coste asintótico del algoritmo de búsqueda añadiendo la creación de la imagen queda acotada por :

$$\mathbb{C} \leqslant \Theta(\frac{4^l}{4^l - 1} n)$$

y para $l \geqslant 2$ tenemos:

$$\Theta(\sum_{i=1}^{|\mathcal{C}|}(1 + E[h(i)])) \approx \Theta(1{,}06n)$$

5.4.2. Comparación del coste entre los dos algoritmos

Se han hecho varias ejecuciones del algoritmos sin imagen y añadiéndolo. Se ha usado un *pentium* a 200 Mhz. Las pruebas se han hecho buscando un patrón de longitud 1. Las ejecuciones se han efectuado recortando el tamaño de un mismo cromosoma de entrada y con el mismo patrón.

Podemos ver el resultado en el gráfico de la figura 5.3.

Como se puede apreciar, la diferencia de tiempo entre los dos algoritmos corresponde con el cálculo asintótico que se ha calculado para $l \geqslant 2$.

5.5. División de Repatt y Repatt_zoom

Como ya se ha explicado, una vez realizado el componente de zoom, se ha resuelto por la creación de dos programas a partir del algoritmo de búsqueda con imagen.

En uno de ellos, al que hemos llamado *Repatt*, se recorre siempre todo el cromosoma, se realiza el conteo de las ocurrencias y se realiza una imagen de éstas para todo el

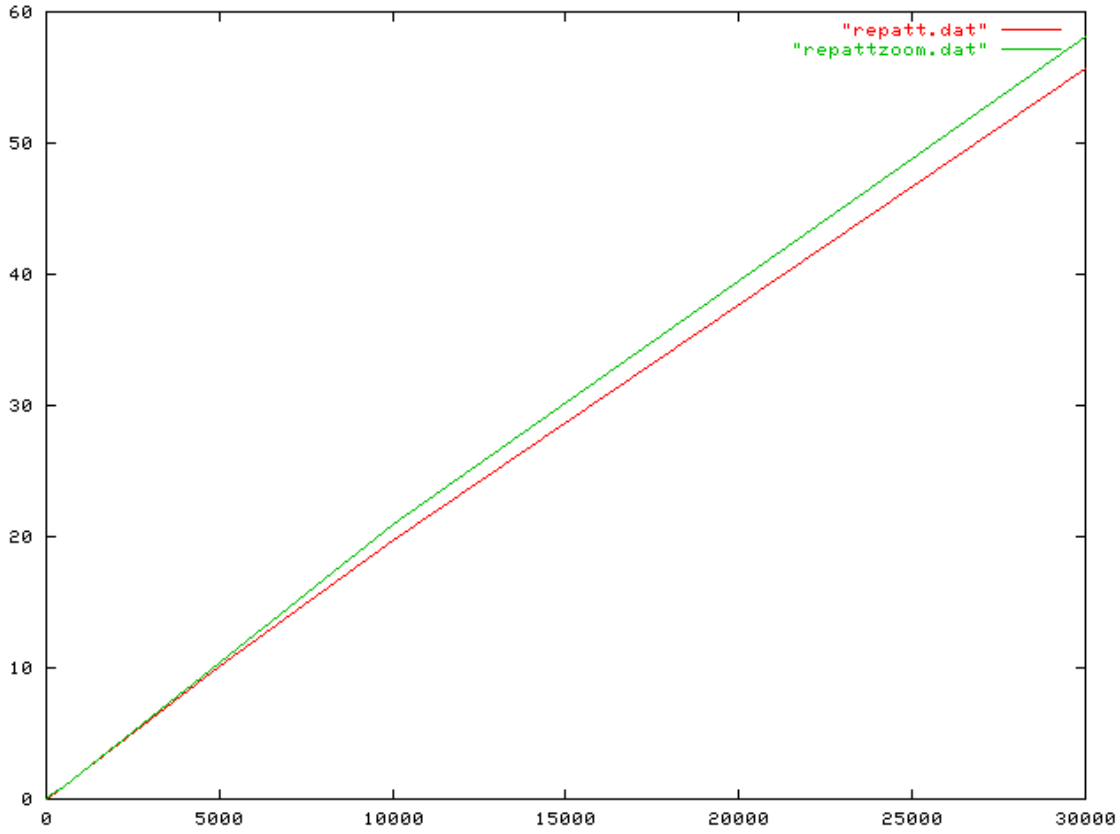


Figura 5.3: Gráfico del coste asintótico comparando la búsqueda con zoom y sin zoom

cromosoma. Por otra parte, el programa que hemos llamado *Repatt_zoom*, recorre la región que indica el usuario y tan solo devuelve la imagen de las ocurrencias en esa región.

Como se ha explicado en la sección 5.3, hemos elegido el método que consisten en recorrer la región del cromosoma cada vez que el usuario lo requiera. La principal característica que requiere el método escogido, es la velocidad en el cómputo del resultado. Es por eso que se han hecho esfuerzos para optimizarla.

Una de las vías de optimización ha sido la separación de un único programa en dos. Esta separación permite a los dos programa, y en especial al que sólo realiza imágenes (*Repatt_zoom*), prescindir de las sentencias condicionales situadas en el bucle principal.

Supongamos un programa que realizara las funciones de conteo y realización de imagen y tan solo una de ellas se hubiera de realizar. El bucle principal de este programa contendría sentencias condicionales que dirigirían la acción a una función o a la otra. Dado que el cuerpo del bucle se recorre para cada carácter del cromosoma, el coste en ciclos de *cpu* de dichas sentencias no es despreciable para cromosomas grandes.

Con la separación de las dos funcionalidades, eliminamos las sentencias condicionales. El coste asintótico queda igual, no obstante el coste real varia en unos segundos con cromosomas de gran tamaño.

Podemos ver un ejemplo de los tiempos para los tres programas distintos en la tabla 5.2 y su representación gráfica en la figura 5.4. Las ejecuciones se han realizado con la máquina *ATEAM* del departamento de LSI de la FIB. La misma máquina con la que corre la aplicación que da servicio a los usuarios del *MREPATT*.

tamaño (kb)	tiempo(seg.) único$_1$	tiempo (seg.) repatt	tiempo(seg.) único$_2$	tiempo (seg.) repatt_zoom
1	0.79	0.81	0.55	0.52
10	5.69	4.85	4.26	3.61
20	10.58	9.43	8.59	7.24
30	15.40	14.12	12.56	10.55
40	20.60	18.60	16.76	14.04
50	25.56	23.10	21.28	17.48

único$_1$: funciona como *repatt*, con conteo y realizando la imagen de las ocurrencias.
único$_2$: funciona como *repatt_zoom*, sin conteo y realizando la imagen de las ocurrencias.

Cuadro 5.2: Tabla de comparación de los tiempos entre un programa único y su separación de funcionalidades

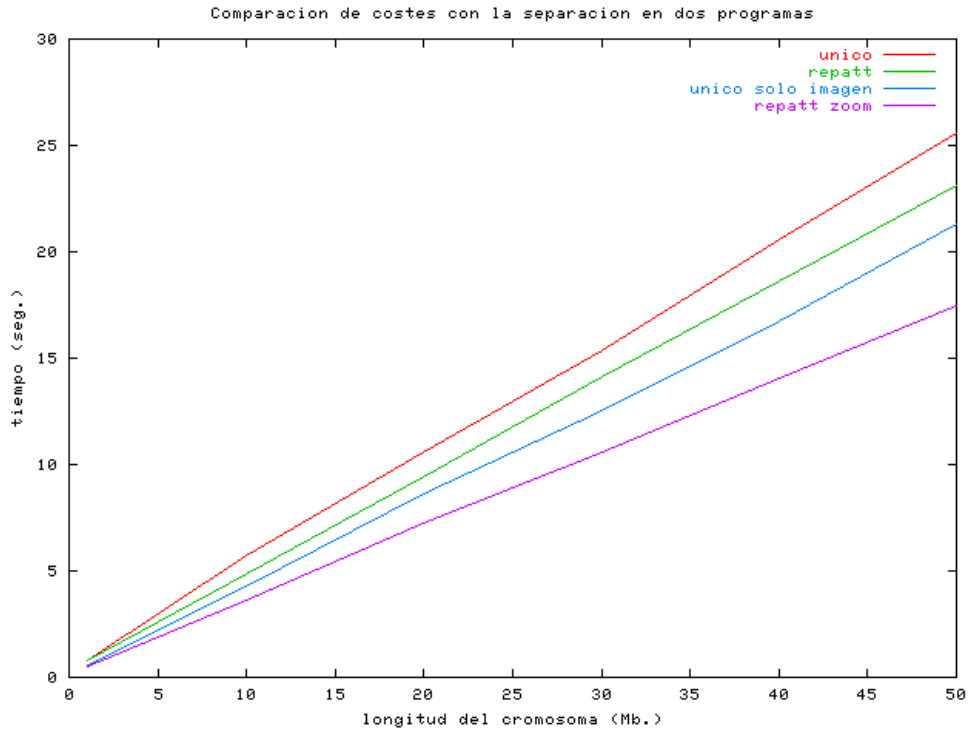


Figura 5.4: Gráfico de comparación de los tiempos entre un programa único y su separación de funcionalidades

En el ejemplo que se muestra, se han comparado los tiempos de un único programa, con flags para indicar qué funcionalidades debe realizar, y los dos programas especializados.

Como se puede ver, si nos centramos en la comparición entre *único* y *Repatt_zoom*, a partir de 15 Mb, la diferencia es mayor de un segundo. Esta diferencia de tiempo demuestra que la división de un único programa con dos funcionalidades en dos con especialización, es una buena vía para diminuir el *throughtput*.

Capítulo 6

Diseño e implementación de MREPATT

Nuestra aplicación es típicamente una *aplicación web*, explicadas en el capítulo **??**. El diseño que se ha realizado se basa en componentes de sistemas de información distribuidos. El patrón arquitectónico que se ha seguido se compone de tres capas, cuyo estilo atómico se representa en la figura 6.1.

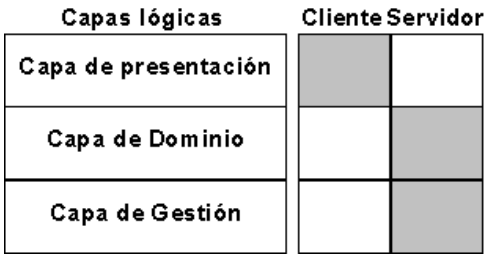


Figura 6.1: Estilo atómico utilizado para el diseño de *MREPATT*

Des del punto de vista de la implementación se presentaron las siguientes restricciones sobre los lenguajes a utilizar:

páginas activas de html	javascript, Applets de Java
aplicación web	perl, C, C++, php
gestión de datos	ficheros de texto, sql

En todas las páginas activas se ha utilizado el lenguaje *javascript*. Con este lenguaje se han implementado las validaciones de la entrada del programa, la comunicación entre ventanas, y se han escrito varios componentes como el menú de especies y el menú de resultados (estos componentes se explican detalladamente en el capítulo **??**). Para la realización de los componentes el autor ha preferido dicho lenguaje a utilizar los *applet* por los siguientes motivos:

- Los *applets* están orientados a incorporar aplicaciones dentro de un página *html*, y no era el caso.
- Los menús dinámicos en las página *html* permiten incorporar elementos de diseño gráfico que son difícilmente reproducidos con los *applets*.
- La mayoría de usuarios que navegan por internet están familiarizados con este tipo de componentes.

La aplicación web se ha implementado en *perl*, y como ya se ha dicho en capítulo anteriores, dicha aplicación ha utilizado el programa *Repatt*. Las ventajas que ha encontrado el autor en el lenguaje *perl* versus *C* o *C++* han sido:

- *Perl* permite trabajar a mas alto nivel en el tratamiento de ficheros. Como se verá en los siguientes capítulos, los resultados que devuelve la aplicación se obtienen básicamente mediante el tratamiento de ficheros de texto plano.
- *Perl* incorpora un gran número de paquetes que han facilitan tareas como la lectura de los parámetros del *CGI* o el envío de correo.
- el lenguaje *Perl* tiene instrucciones que permiten realizar directamente análisis sintácticos en ficheros de texto. Como se verá, se ha utilizado esta característica para la generación de las páginas *html*, y para realizar un analizador sintáctico que lee los ficheros índice en la capa de gestión.

En cuanto a utilizar *perl* o cualquier otro lenguaje de *script* similar como *php*, *ruby* o *tcl/tk*, el autor se ha decantado por el primero por ser el que mas conocía.

Para la gestión de datos se trabaja directamente con ficheros de texto plano. No hemos visto la necesidad de realizar una base de datos en un sistema gestor por los siguientes motivos:

- Hubiese aumentado mucho el coste y el tiempo de producción.
- El lenguaje *Perl*, con el que se ha implementado *MRepatt*, trabaja muy bien con ficheros de texto.
- Trabajar con ficheros de texto plano tiene la ventaja de que son muy fáciles de manipular, ya sea directamente o a través de otros programas. Por tanto, los datos de la aplicación son totalmente portables, ayudan al mantenimiento y también a la interoperabilidad entre dos o mas entidades de software.

En las secciones siguientes se explica el diseño, en líneas generales, de las tres capas de la arquitectura. En la sección referente a la capa de presentación se explican los diferentes modelos de implementación existentes para generar páginas en *aplicaciones web*. Dado que trabajaremos con *perl* y típicamente el *html* está empotradas en el código, se presenta un diseño alternativo para poder separar este acoplamiento y trabajar directamente con la página *html*.

En la sección que habla de la capa de domino, se explica la estructura en módulos de la aplicación. Cada uno de éstos, cuyo diseño e implementación se detallan en los capítulos siguientes, han de implementar las siguientes funcionalidades del sistema:

- Entrada de los cromosomas y patrones.
- Gestión de los genomas que añaden los usuarios.
- Cálculo de los resultados de la búsqueda para cada cromosoma.
- Cálculos y visualización de los resultados para un grupo de cromosomas.

Por último, en la sección referente a la capa de gestión se explica la estructura de directorios y ficheros que utiliza la aplicación y se presentan los componentes que se han implementado para el acceso a los datos permanentes.

6.1. Diseño e implementación de la capa de presentación

Cuando se trabaja con componentes CGI que escriben directamente la página *html*, como *perl*, *C* o *C++* , nos encontramos ante el inconveniente de que la página está empotrada [1] dentro del código del lenguaje. Esto dificulta la separación entre la capa de presentación y la de dominio. Cuando se requiere modificar la página *html* que se presenta como resultado, se debe hacer directamente en el código programado y si queremos diseñar la página o modificar un componente visual como el color, un nuevo botón, etc,... no se pueden hacer servir herramientas de diseño gráfico sobre páginas *html*. Por tanto, los diseños gráficos y el mantenimiento de la capa de presentación se complican cuando usamos este tipo de lenguajes para implementar aplicaciones web. En la figura 6.2 se puede ver un ejemplo de código *CGI* hecho en *perl* y que escribe una página *html*.

Como se puede apreciar, la página *html* esta *incrustada* dentro del programa *perl*, en las sentencias *print*. El mismo tipo de código nos encontramos en aplicaciones escritos en cualquiera de los lenguajes comentados anteriormente, con la dificultad añadida de que en la mayoría de casos se deberá compilar cada vez que se precise modificar la página.

A diferencia de éste, existe otro modelo que soluciona el problema, separando la parte de presentación y la de dominio. En este caso nos encontramos ante un modelo opuesto al anterior en el sentido que es el lenguaje de implementación el que está empotrado en la página *html*. Algunas de las arquitecturas y lenguajes que permiten trabajar con esta fórmula son la arquitectura *J2EE* de *Sun* con los *jsp*, la arquitectura *ASP* de *Microsoft* o el lenguaje *PHP* de *Apache Software Foundation*. Podemos ver un ejemplo de código para crear un página *html* con *jsp* en la figura 6.3.

En el ejemplo se puede apreciar que el código es el de una página *html*, con algunos *tags* especiales que indican las regiones dónde hay instrucciones de *jsp*. Éste modelo facilita el diseño y mantenimiento de la capa de presentación, ya que ha diferencia del

[1] se utilizan los términos *código empotrado* o *código incrustado* para designar lo que en inglés se conoce como *embedded code*

```
#Copia del contenido del fichero introducido en el formulario, en el fichero temp
open(INPUTFD, ">$TRANSPOFILE")||die ("¡No se puede abrir el fichero!");
print INPUTFD "$cgi_data('upfile')";
close(INPUTFD);


#Ejecutamos el programa Trans, que hace la busqueda de transposones con TIR conoc
system(`$TRANSPOHOME/transpo.sh $TRANSPOFILE $TIR $ERR $MINBODY $MAXBODY $TARGET

# Mostramos la salida al cliente
  print "<h2><code>TRANSPO </code> RESULTS</h2>";
  print "<P>";
  print "The list of transposons found is printed in <i>transpo</i> format and
in FASTA format.";
  print "<P>";
  print "The following list of transposons is determined by:";
  print "<pre><tt>\n";
  print "TIR: $TIR  ";
  print "Errors: $ERR   ";
  print "Minimum length: $MINBODY   ";
  print "Maximum length: $MAXBODY   ";
  print "</pre></tt>\n";

  print"<P>";
```

Figura 6.2: Ejemplo de diseño de pantallas en Perl

```
<HTML>
<%@ page language="java" imports="java.util.*" %>

<H1>Welcome</H1>

<P>Today is </P>
<jsp:useBean id="clock" class="jspCalendar" />
<UL>
<LI>Day: <%= clock.getDayOfMonth() %>
<LI>Year: <%= clock.getYear() %>
</UL>
<%-- Check for AM or PM --%>
<%! int time = Calendar.getInstance().get(Calendar.AM_PM); %>
<%
if (time == Calendar.AM) {
%>
Good Morning
<%
}
else {
%>
Good Afternoon
<%
}
%>
<%@ include file="copyright.html" %>
</HTML>
```

Figura 6.3: Ejemplo de diseño de pantallas en JSP

anterior, se puede trabajar directamente con la página *html*. Esto permite realizar maquetas y prototipos de las pantallas de una forma rápida, generalmente con la ayuda de una herramienta de diseño gráfico. Con el modelo anterior, para realizar un prototipo se suele generar la página, encapsularla en el código *perl* (o el lenguaje de implementación) y posteriormente compilarlo.

Dados estos dos modelos, hemos preferido trabajar con el segundo por todas las ventajas que conlleva. No obstante, dado que hemos implementado nuestra aplicación con *perl*, hemos tenido que realizado un diseño con las adaptaciones correspondientes para llevarlo a la práctica. Dicho diseño se basa en lo que hemos llamado *macros*. Las macros son archivos *html* que contienen *tags* especiales que indican regiones con variables, o estructuras repetitivas. Cada pantalla que se presenta al usuario es una instancia de una macro. Cuando se pide una página, hay un componente *perl* que lee la macro y efectúa un análisis sintáctico para instanciar el contenido de los *tags* con los valores concretos.Podemos ver un ejemplo de una macro en la figura 6.4.

```
<table border="0" width="550" cellspacing="0" cellpadding="0" bgcolor="#210592"><tr><td>
    <table border="0" width="100%" cellspacing="1" cellpadding="2">
    <tr bgcolor="#bec5ff">
      <td class="titleitem" width="225" align="center" valign="middle"> Specie </td>
      <td class="titleitem" width="225" align="center" valign="middle"> Chromosome </td>
      <td class="titleitem" width="100" align="center" valign="middle"> Size </td>
    </tr>
    <!--%BEGIN_SPECIE-->
    <tr bgcolor="white">
      <td rowspan="%NODES" align="left"  valign="top" class="item"> %SPECIE </td>
      <td class="item"> %CHROMOSOME </td>
      <td class="item" align="right"> %SIZE Mb </td>
    </tr>
    <!--%BEGIN_NODE-->
    <tr bgcolor="white">
      <td class="item">  %CHROMOSOME </td>
      <td class="item" align="right"> %SIZE Mb </td>
    </tr>
    <!--%END_NODE-->
    <!--%END_SPECIE-->
    </table>
```

Figura 6.4: Ejemplo de diseño de pantallas con macros en *MREPATT*

Como se puede apreciar, las variables que deben ser instanciadas son aquellas que empiezan con el carácter `%` (por ejemplo `%NODES`) y las regiones repetitivas están delimitadas por los tags que corresponden a comentarios de código *html* que comienzan con `%BEGIN_` y `%END_`.

Con este modelo separamos completamente la capa de presentación de la capa de dominio, y nos permite realizar diseño de páginas web con herramientas para este propósito. Si deseamos cambiar cualquier cosa del diseño, no hace falta buscar el código *PERL* que hace la instanciación.

Como ya se ha dicho en la sección anterior, y se detalla en la sección siguiente, la aplicación se ha dividido en módulos con el fin de tratar cada funcionalidad del sistema por separado. Dentro de cada módulo existe un componente que analiza sintácticamente las *macros* de su funcionalidad y genera las páginas *html* que le pertocan. En los capítulos siguientes se detallan las operaciones de cada una de las pantallas y los componentes javascript que se han implementado en la capa de presentación.

6.2. Diseño e implementación de la capa de dominio

Como ya se ha dicho, se ha utilizado para la aplicación un patrón arquitectónico de tres capas. La capa de dominio se ha estructurado en módulos y se ha seguido un patrón de diseño *fachada*, es decir, hay un único componente que recibe las todas las peticiones de los clientes y las reenvia al módulo que las trata. Un modelo alternativo sería que la petición llegase directamente al módulo, no obstante se ha optado por el diseño *fachada* por los siguientes motivos:

- Este patrón de diseño proporciona una interficie unificada para el conjunto de módulos, permitiendo un uso mas sezillo de éstos. Todas las llamadas a la aplicación siguen el mismo formato, cambiando unicamente los paramétros.

- Las validaciones de los parámetros de entrada se pueden efectuar desde un único componente. Esto mejora la seguridad del sistema a todos los niveles, en especial a nivel de internet, pues sólo hay una *puerta* de aceso a la aplicación.

- Este tipo de diseño facilita el mantenimiento de la aplicación y la extensibilidad, es decir, si se requiere una nueva funcionalidad y se crea un nuevo módulo entonces las peticiones a la aplicación no deben cambiarse, sino que se modifican los parámetros de éstas.

Como se puede ver en la figura 6.5, la aplicación se compone de cinco módulos principales y un módulo de enlace entre la capa de dominio y la de gestión. El módulo de enlace, al que hemos llamado *RepattUtil*, se encarga de las siguientes tareas:

- Facilitar la ruta de los archivos que cualquier módulo pueda necesitar.

- Controlar la concurrencia de los accesos de los ficheros.

- Contiene las clases que leen los ficheros índice para acceder a los datos.

Cada uno de estos módulos se encarga de servir las páginas *html* a los clientes de las peticiones que implementa. Para ello necesita acceder al fichero de *macros* explicado en la sección anterior. Por tanto, todos los módulos necesitan accedir almenos una vez al módulo *RepattUtil* para conocer la ruta del fichero *macro*.

La *fachadada* de nuestro modelo la implementa una sola clase llamada *Repatt.pl* y esta clase nos ayuda a resolver el problema que comporta trabajar con clientes sin estado de las aplicaciones basadas en el *Web Server*, dado que es el responsable de saber el estado a partir de los parámetros que recibe de la página *html*. Dicho problema se puede resumir de la siguiente manera:

> Cada vez que un cliente realiza un petición al servidor, éste le sirve un nueva página *html*. Si la página servida se carga en la misma ventana desde donde se hizo la petición, toda la información de la página que hizo la petición se pierde. Por tanto, se pieden la petición qe se hizo y cualquier otro elemento

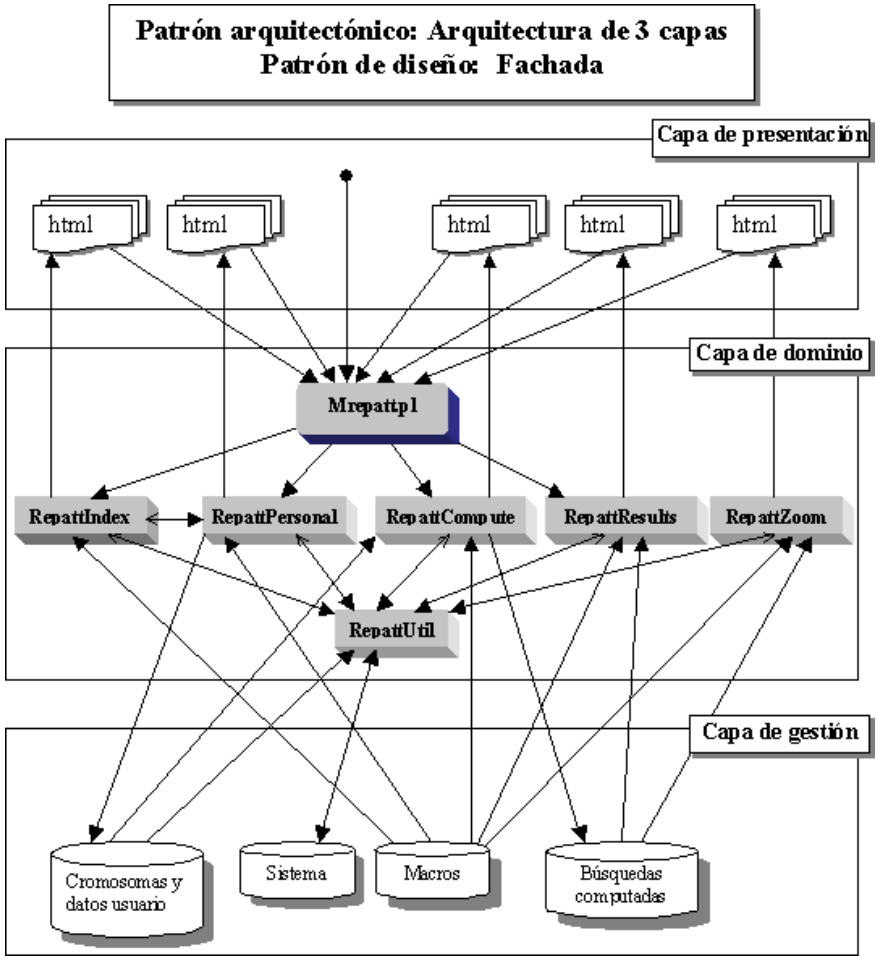


Figura 6.5: Módulos de *MREPATT*

que introdujo el usuario. El problema se hace efectivo cuando la nueva página (o descendientes de ésta) requiere información que insertó el usuario en alguna de las página que ya fueron substituidas por las nuevas.

Para solucionar este problema existen tres vías no excluyentes, que son guardar el estado de la aplicación en el lado del cliente mediante *cookies*[2], guardar el estado en el lado del servidor y , des del lado del cliente, hacer que la página de respuesta del servidor se aloje en una ventana diferente que la de petición. Típicamente se hacen servir las tres, no obstante, la forma de tratar el estado en el servidor varía mucho en función de la arquitectura del servidor de aplicaciones. Nuestra estrategía se base en dos ideas:

- Rescordar el estado a través de los parámetros que se envían con las peticiones y

[2]las cookies son registros de información que los navegadores pueden almacenar en el disco duro de los clientes sin perjudicar su privacidad.

respuestas del cliente a servidor, en otras palabras, en las páginas que se sirven se va acumulando la información para las próximas peticiones. Esta solución sólo sirve cuando la información que se acumula no es excesiva, como era nuestro caso. La implementación de esta solución se basa en el componente *fachada*, ya que es este el que reconoce el estado de la aplicación para cada cliente, y envia las peticiones al módulo que le corresponde.

- Dar de alta una sesión de usuario y escribir el estado en el disco duro. Cuando el usuario finaliza se da de baja la sesión y se borran los ficheros creados. Se detalla la implementación de esta solución en el capítulo **??** del módulo *RepattResults*, ya que se ha hecho servir únicamente para dicho módulo.

El módulo *RepattIndex* escribe la página de inicio de la aplicación. La principal característica de éste es el tratamiento de los parámetros de entrada de nuestra aplicación, en especial el árbol filo-genético con el que tiene que trabajar el usuario para elegir los cromosomas. Dicho árbol se ha implementado en la capa de presentación con un componente javascript. Dado que, como se explicará en los siguientes capítulos, dicho árbol incorpora los cromosomas que añadan los usuarios, el módulo *RepattIndex* se comunica con *RepattPersonal* para saber qué cromosomas debe incorporar.

El módulo *RepattPersonal* se encarga de todos los aspectos referentes a la gestión y consulta de los ficheros personales de los usuarios. Cada usuario dispone de una cuenta pública para poder insertar o eliminar los cromosomas que estime oportunos. Se ha implementado una interficie poder realizar dicha tarea, para posteriormente, realizar las consultas de éstos cromosomas mediante el módulo *RepattIndex*.

El cómputo y análisis de las repeticiones de un patrón en un grupo de cromosomas, se basa en la recopilación de los resultados individuales de las búsquedas de dicho patron para cada uno de los cromosomas que pertenecen a este grupo. Para no tener que calcular en cada consulta los resultados individuales, éstos se guardan en la base de datos, y de esta forma sólo hay que computarlos la primera vez que aparecen en una petición. El módulo que realiza los cálculos individuales y los guarda a disco es *RepattCompute*. Este módulo hace las llamadas pertinentes al programa *Repatt*, monitoriza el progreso de los cálculos y envía un correo electrónico al usuario cuando éstos finalizan. El módulo *RepattResults* recoge los resultados obtenidos, y a partir de éstos obtiene el cómputo de las repeticiones para un grupo de cromosomas. Este módulo se encarga de generar la interficie para su visualización, y mantener el control de una sesión para eliminar los ficheros temporales a su finalización.

Por último, el módulo *RepattZoom* se encarga de realizar las llamadas pertinentes y devolver las imágenes creadas por el programa *Repatt_Zoom*. Presenta la interficie gráfica para que el usuario pueda navegar por la imagen y elija las regiones que crea interesantes. En las siguientes secciones se describirán detalladamente cada uno de éstos módulos, sus implementación y sus funcionalidades.

6.3. Gestión de datos

La aplicación *MRepatt* trabaja directamente con ficheros de texto plano. No hemos visto la necesidad de realizar una base de datos en un sistema gestor por los siguientes motivos:

- Hubiese aumentado mucho el coste/tiempo de producción.
- El lenguaje *Perl*, con el que se ha implementado *MRepatt*, trabaja muy bien con ficheros de texto.
- Trabajar con ficheros de texto plano tiene la ventaja (creemos que es una ventaja) de que son muy fáciles de manipular, ya sea directamente o a través de otros programas. Por tanto, los datos de la aplicación son totalmente portables, ayudan al mantenimiento y también a la interoperabilidad entre dos o mas entidades de software.

Se ha estructurado y organizado la información mediante estructuras de directorios del sistema operativo. Dicha información se puede dividir en dos tipos : la información permanente y la temporal. La información que necesita guardar/manejar la aplicación de forma permanente es:

- Información referente a los cromosomas.
- Información referente a los resultados ya calculados.
- Ficheros de las Macros para generar la salida.

Durante una sesión de un usuario también se guardan en el servidor las páginas de resultados que se van mostrando y se eliminan en cuanto finaliza dicha sesión. El motivo por el que se guardan es que dichas páginas se generan bajo demanda la primera vez que las pide el usuario. Si el usuario vuelve a pedir una página, se muestra la que hemos guardado para así no volver a generarla otra vez. En la sección **??** se explica mas detalladamente qué es una sesión y su funcionamiento.

6.3.1. Gestión de los cromosomas

Los cromosomas que manejara la aplicación son ficheros de tipo *FASTA* y se encuentran en un directorio compartido por otras aplicaciones, dentro del mismo servidor donde se ejecuta *MRepatt*. Todos los cromosomas de una especie se encuentra en un subdirectorio dentro del que se ha mencionado. Por tanto lo que se necesita es un fichero índice que indique exactamente en que directorio se encuentra un cromosoma. Dado que el usuario trabaja con el árbol filo-genético de las especies (ver sección 7.4 del manual) el fichero índice debe tener contener dicha estructura.

Para crear dicho fichero se ha diseñado un pequeño lenguaje que hemos llamado *"MapTree"*. La Formalización es la siguiente:

maptree → main root arbol
main → [**Main** (**base_path** Path) (**root** Etiqueta)]
root → [**root** ramas]
arbol → nodo | arbol nodo
nodo → [Etiqueta ramas]
ramas → nodo_padre | nodo_hoja | ramas nodo_padre | ramas nodo_hoja
nodo_padre → (IdNodo Etiqueta)
node_hoja → (IdNodo Etiqueta Path)

Cuando se insertan o eliminan cromosomas, se debe editar el fichero índice para modificarlo. A continuación se muestra un fragmento de ejemplo del fichero índice:

```
[Main
  (base_path "/homes/aux-users26/users/alggen/genomes/")
  (root "Species")
]

[root
  (eukaryota "eukaryota")
  (virus "viruses")
  (bacteria "bacteria")
]

[eukaryota
  (animals "animals")
  (plants "plants")
  (fungus "fungus")
  (parasite "parasite")
  (ameba "ameba")
]

[virus
  (alcelaphineHV "alcelaphineherpesvirus" "virus/alcelaphineherpesvirus")
  (atelineHV "atelineherpesvirus" "virus/atelineherpesvirus")
  (bovineHV "bovineherpesvirus" "virus/bovineherpesvirus")
  (cercopithesineHV "cercopithesineherpes" "virus/cercopithesineherpesvirus")
  (equineHV "equineherpesvirus" "virus/equineherpesvirus")
]
```

Además de los cromosomas que *ALGGEN* tiene guardados en el servidor, el usuario también puede trabajar con los suyos. Para que un usuario pueda comparar sus cromosomas con los de *ALGGEN* se ha pensado que la mejor forma es introducir una especie en el árbol, de donde cuelgan todos los cromosomas que éste vaya añadiendo. Para cada usuario se guarda un fichero únicamente con su *rama*. A continuación se muestra un ejemplo:

```
  (i103987966914143 "test1" "user/rroset@lsi.upc.es/103987966914143")
  (i103987966914144 "test2" "user/rroset@lsi.upc.es/103987966914144")
```

Cada vez que un usuario entra dentro de la aplicación se crea un nuevo fichero índice concatenando el índice general y el particular del usuario. De esta forma, si el fichero general se cambia, el fichero particular del usuario también. A continuación se detallan

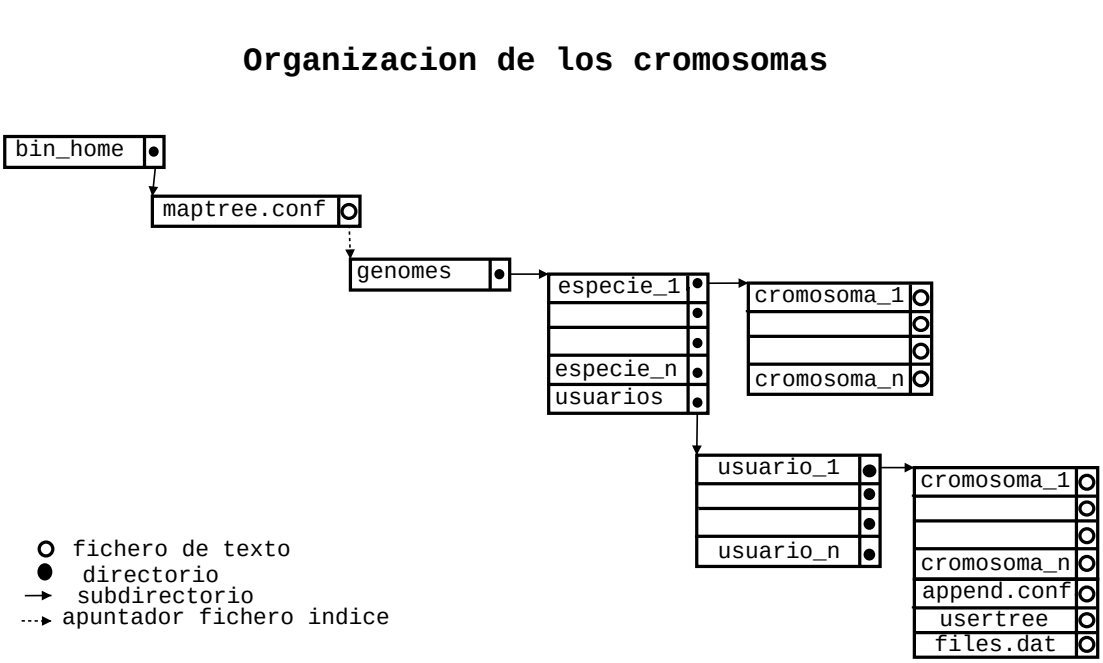


Figura 6.6: Organización de los cromosomas

los directorios y ficheros que aparecen en la figura 6.6, que describe la organización de los directorios para guardar la información referente a los cromosomas:

- Directorio "bin_home": Es el directorio base de la aplicación, dónde se ubica el programa principal.
- Fichero "maptree.conf": Es el fichero índice explicado anteriormente, que guarda en una estructura de árbol los caminos (*path*) hacia los cromosomas.
- Directorio "genomes": Directorio base dónde se guardan los cromosomas. Es un directorio compartido por mas aplicaciones que también utilizan cromosomas. Esta compuesto de subdirectorios para cada una de las especies que tenemos guardadas. Como se ha dicho, hay un caso especial, la especie *"user"*, que guarda la información referente a cada usuario.
- Directorio "especie$_i$": Directorio que guarda los ficheros pertenecientes a dicha especie.
- Directorio "usuarios": En este directorio guardamos la información de los usuarios. Dentro tendremos un subdirectorio para cada usuario que haya añadido algún cromosoma.
- Fichero "cromosoma$_i$": Son los ficheros en formato *FASTA* con la secuencia de *ADN* de un cromosoma.

- Directorio "usuario$_i$": En este directorio se guarda toda la información referente a un usuario. Dentro tenemos los ficheros de los cromosomas que ha añadido, y además la información necesaria para gestionarlos.

- Fichero "append.conf": En este fichero se guarda la extensión necesaria del fichero "maptree.conf" para poder generar el fichero índice personalizado de cada usuario.

- Fichero "usertree": Este es el fichero índice personalizado del usuario. Es el mismo que "maptree.conf" con las ramas de los cromosomas de cada usuario añadidas al árbol. Éste fichero se genera cada vez que el usuario entra en la aplicación.

- Fichero "files.dat": Información asociada a la gestión y mantenimiento de los cromosomas del usuario. En este fichero guardamos el tamaña de los cromosomas, el identificador asociado a cada cromosoma y la fecha en que fue añadido.

Cada cromosoma tiene un identificador único global, es decir, dados dos cromosomas cualquiera de dos especies distintas, dichos cromosomas tendrán identificadores diferentes. Se accede al fichero "files.dat" en exclusión mutua cada vez que se modifiquen los ficheros de un usuario, con el fin de evitar efectos laterales al estar trabajando en un entorno concurrente y multi-usuario. Los cromosomas que añaden los usuarios reciben su identificador de forma automática y no es posible que dos cromosomas añadidos en el mismo instante de tiempo tengan el mismo identificador.

6.3.2. Gestión de los resultados de Repatt

Con *Repatt* obtenemos los resultados de la búsqueda de un patrón en un sólo cromosoma. *MRepatt* utiliza los resultados de cada uno de los cromosomas seleccionados por un usuario dentro de un grupo para computar los resultados referentes a ese grupo. Cada vez que el usuario hace una consulta *MRepatt* se ejecuta para cada cromosoma y patrón *Repatt*, posteriormente se recogen los resultados individuales y se computa el resultado para el grupo. Para acelerar este proceso, guardaremos los resultados individuales de cada cromosoma y patrón que nos devuelve *Repatt*.

A continuación se detallan los directorios y ficheros que aparecen en la figura 6.7, que describe la organización de los directorios para guardar la información referente a los resultados:

- Directorio "data": Es el directorio base dónde se ubican los resultados por cada patrón y cromosoma.

- Directorio "cromosoma$_i$": Es el directorio base dónde se ubican los resultados por cada patrón. Cada cromosoma tiene un identificador único, que es el que aparece en el fichero índice con estructura de árbol. Cada uno de los directorios "cromosoma" recibe como nombre este identificador.

- Directorio "patron$_i$": Es el directorio base dónde se ubican los resultados, para un cromosoma i un patrón. El nombre del directorio es directamente el nombre

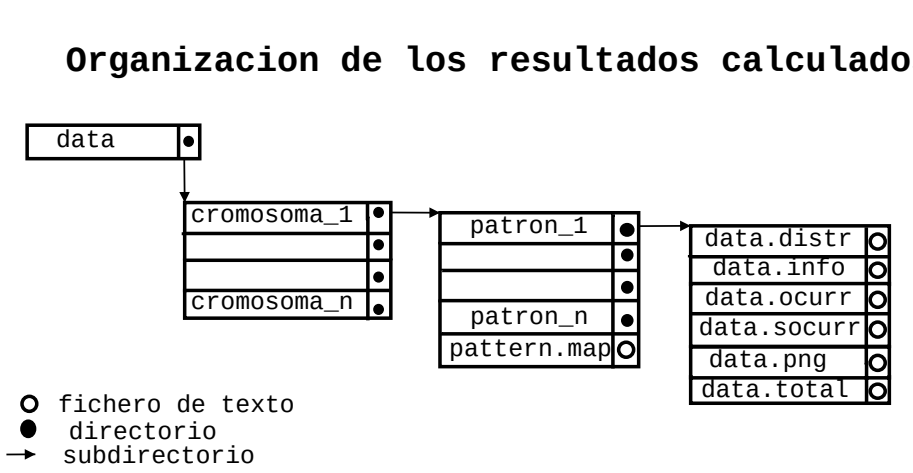


Figura 6.7: Organización de los resultados

del patrón que se ha buscada para un identificador de cromosoma. Dentro de este directorio aparecen los ficheros que ha escrito *Repatt*.

- Fichero "pattern.map": Este fichero tiene dos funciones: guardar la lista de patrones que tenemos calculados, y controlar la concurrencia de las ejecuciones a *Repatt*.
- Fichero "data.distr": Es el fichero que nos devuelve *Repatt* con la lista de ocurrencias acumuladas teóricas.
- Fichero "data.info": Es el fichero que nos devuelve *Repatt* con información referente a la imagen del mapa total de las ocurrencias. En este fichero se guarda por ejemplo la altura y anchura de la imagen, y así poder determinar el marco de la imagen en la página *html* que se devuelve.
- Fichero "data.ocurr": Es el fichero que devuelve *Repatt* con la lista de ocurrencias acumuladas.
- Fichero "data.socurr": Es el fichero que devuelve *Repatt* con la lista de ocurrencias únicas.
- Fichero "data.png": Es la imagen que devuelve *Repatt* con el mapa para todo el cromosoma de las ocurrencias, el *zoom*.
- Fichero "data.total": Es el fichero que devuelve *Repatt* con el número total de cada una de las bases, y con la tabla de *Markov* de primer orden.

Cada vez que la aplicación necesita acceder a los resultados, ya sea para añadir un nuevo cómputo o leerlos, se accede a través del fichero *"pattern.map"*. Este fichero se abre en exclusión mutua, para controlar que no hayan dos procesos que calculen el mismo resultado. Se verá con mas detalle en el capítulo **??**.

6.3.3. Componentes Perl para acceder a la información

En la figura 6.8 aparecen los componentes que enlazan la capa de dominio con la de gestión. A continuación se detallan sus funcionalidades:

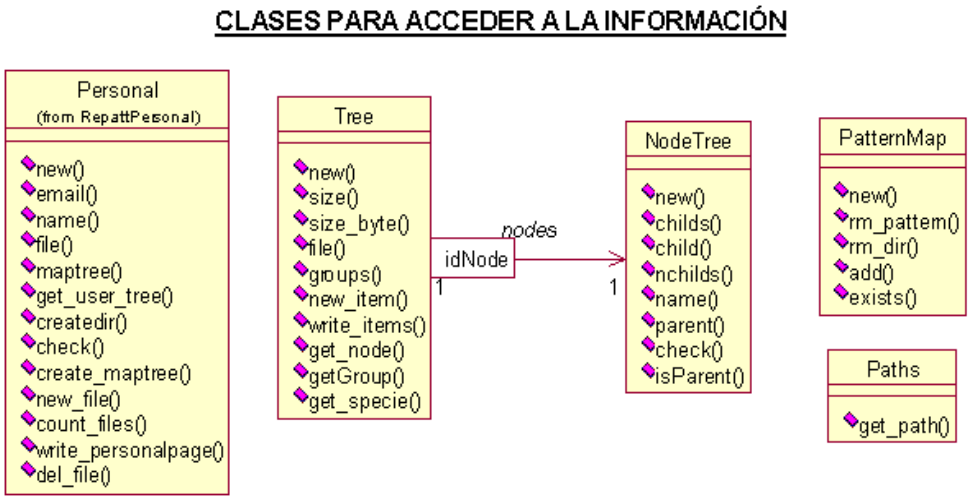


Figura 6.8: Componentes *Perl* para acceder a la información

- Fichero *Paths*: Hemos utilizado *Path* como fichero de configuración de la aplicación. En dicho fichero hemos introducido todos los *paths* que necesita *MRepatt* para funcionar. Así, este fichero guarda la dirección de las macros, de los programas a los que llama *MRepatt*, direcciones html que necesita, etc.

- Clase *Personal*: Esta clase es la encargada de crear el fichero *"maptree.conf"* de un usuario, cada vez que entre en la aplicación. También se usa para acceder a la información de los ficheros de un usuario.

- Clase *Tree*: En esta clase hemos programado el analizador sintáctico del lenguaje *maptree*. La clase se encarga de analizar el fichero y dar información sobre éste.

- Clase *PatternMap*: Clase para acceder en exclusión mutua a los ficheros índice de los patrones.

6.4. Implementación de MRepatt

Para explicar la implemtación de *MRepatt* empezaremos explicando el flujo de las pantallas, para cada una de las pantallas se presentarán los diferentes casos de uso que se han encontrado, y para cada uno de éstos se detallará su diagrama de secuencia.

Debido a la naturaleza de las pantallas sobre un navegador de páginas web, se ha desarrollado un diagrama orientado a este propósito. Los dos elementos que aparecen en el diseño de las pantallas web son:

- Ventanas.
- Frame.

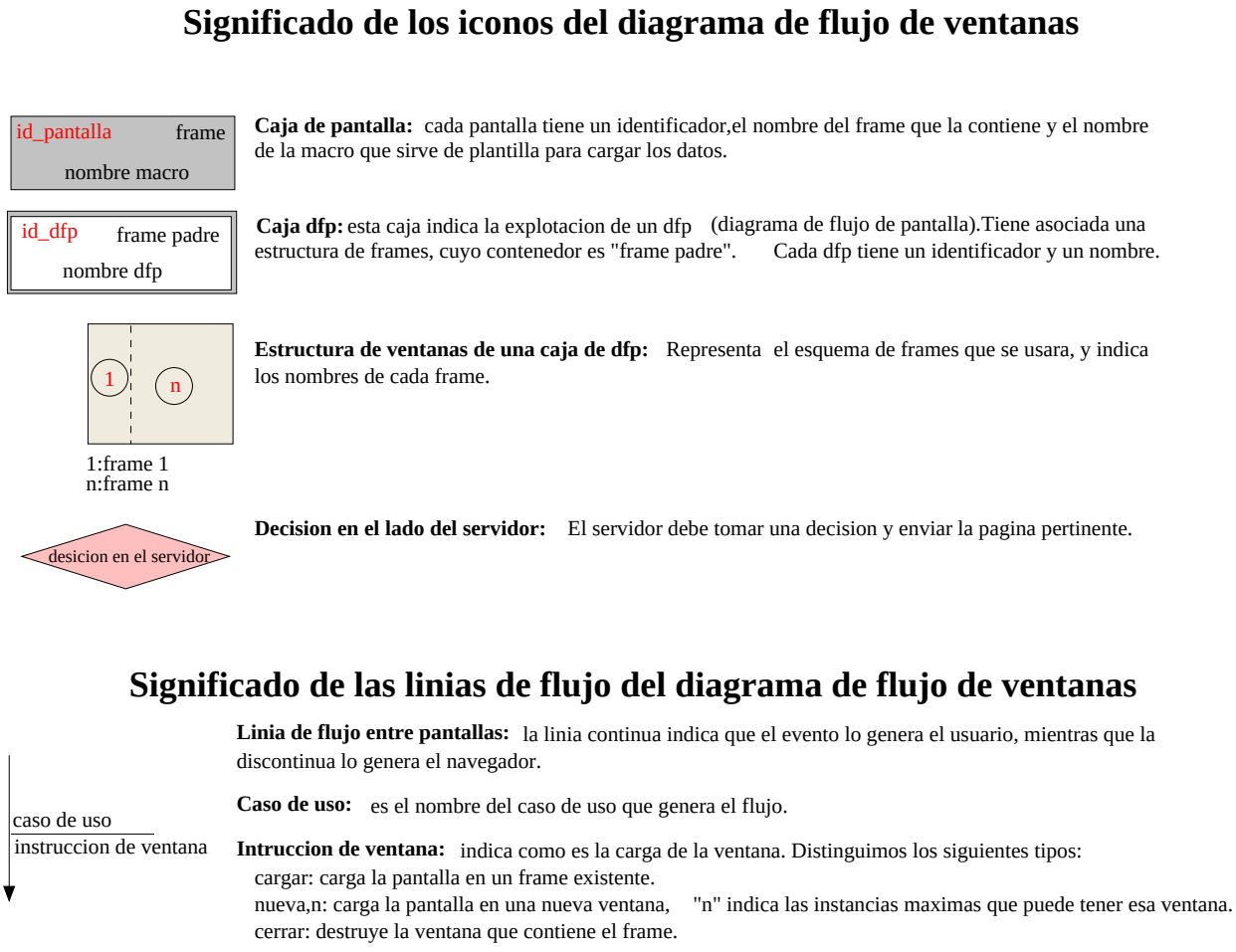


Figura 6.9: Elementos que aparecen en el diagrama de flujo de pantallas

En el diagrama que se expone, se tratan los frames y las ventanas como un mismo elemento, y se ha elegido frame para identificarlo. Dado que un frame puede contener un conjunto de frames, se ha analizado cada contenedor con un diagrama distinto. Con el diagrama que se ha desarrollado se modeliza tanto el comportamiento de las ventanas como el de los frames. Para cada petición al servidor, se indica el nombre de la pantalla que se carga y el frame donde se hace. También se ha indicado el tipo de carga que se

realiza para cada pantalla, y se distingue entre una carga sobre el mismo frame, el cierre de la ventana contenedora del frame, o la creación de una nueva ventana. Podemos ver en la figura 6.9 los elementos que aparecen en el diagrama.

El primer diagrama de flujo de pantallas que se presenta es el prinicipal, cuyo contenedor es el frame que hemos llamado "search". Podemos ver este diagrama en la figura 6.10.

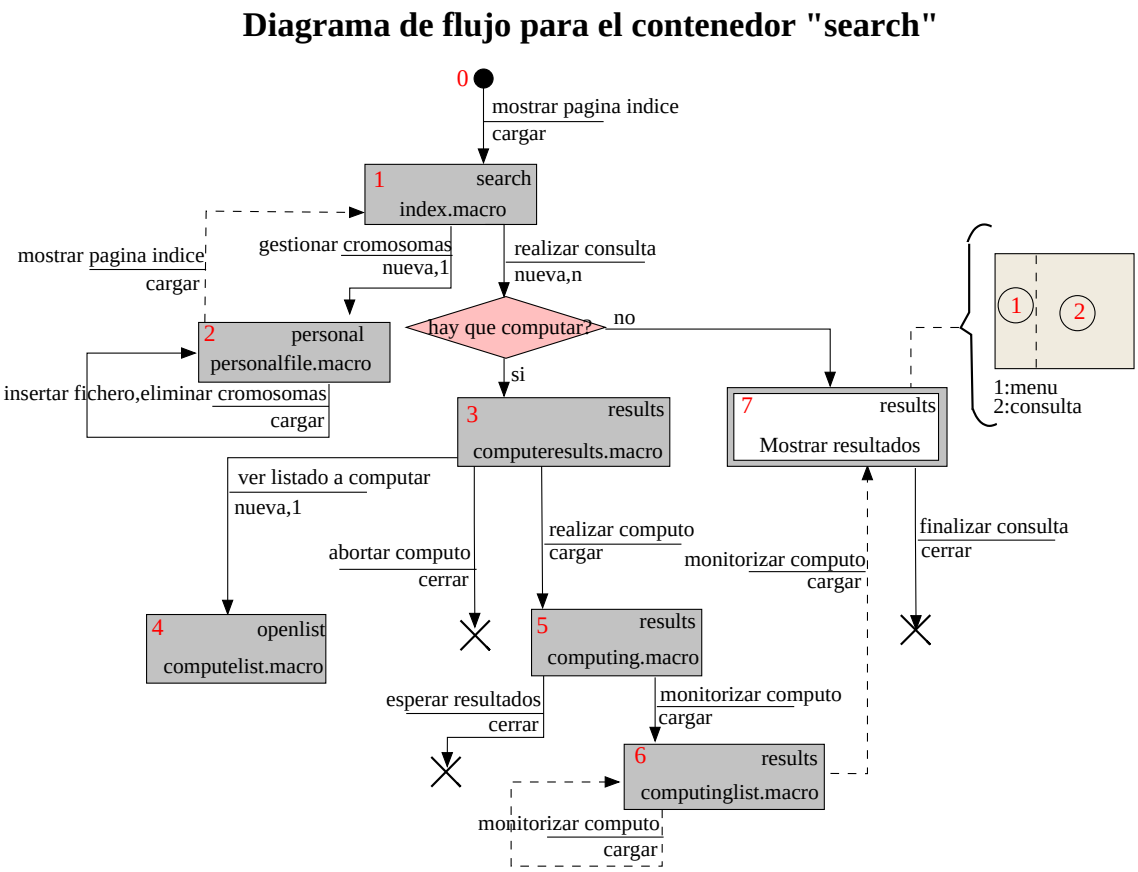


Figura 6.10: Elementos que aparecen en el diagrama de flujo de pantallas

El primer diagrama hace una explotación de otro, para modelizar el flujo de pantallas de los resultados. Los resultados aparecen en una nueva ventana con dos *frames*, cuyo contenedor es el *frame* "results". Podemos ver este segundo diagrama en la figura 6.11.

En las siguientes secciones se describen los casos de uso para cada pantalla, y se analiza las acciones en el servidor para cada caso.

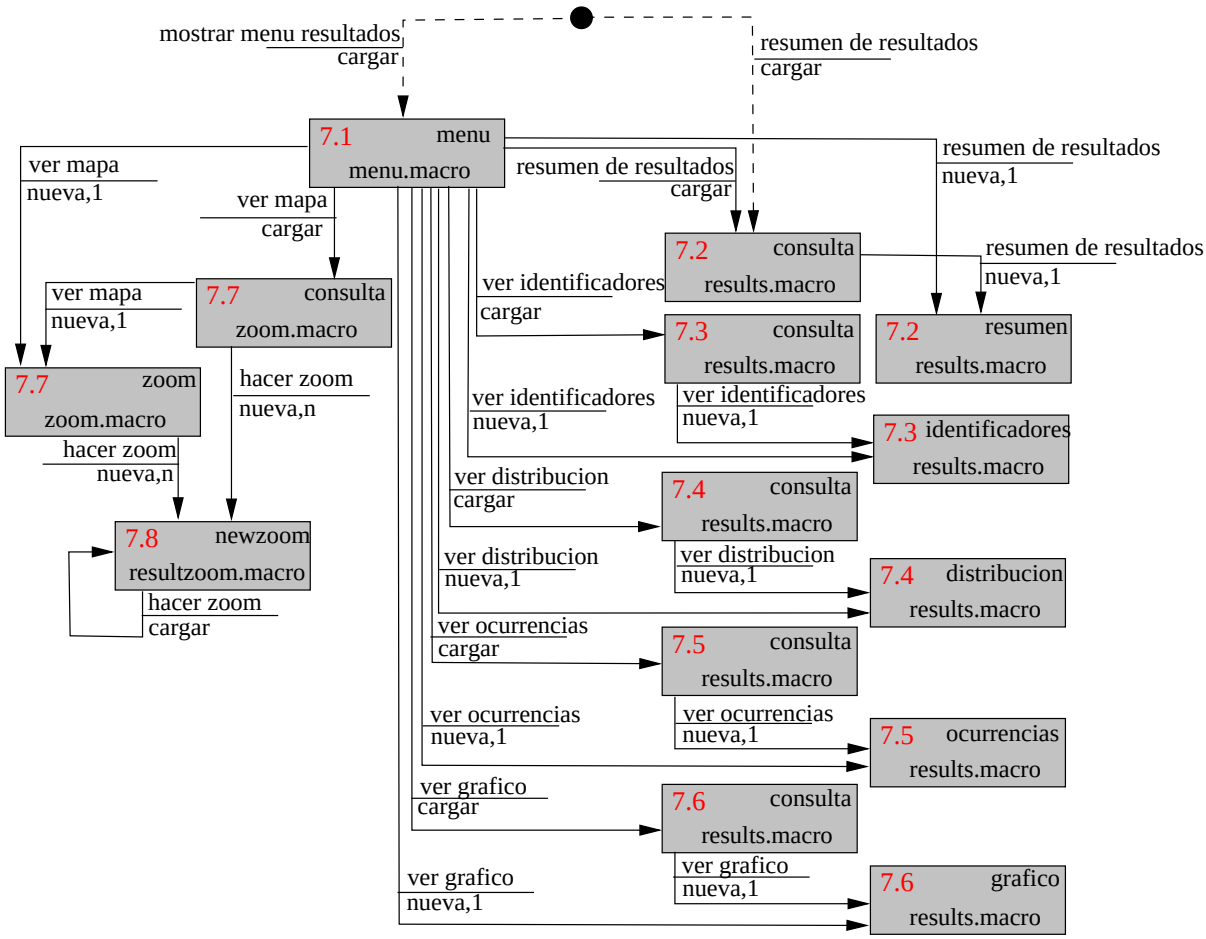


Figura 6.11: Elementos que aparecen en el diagrama de flujo de pantallas para los resultados

6.5. Pantalla 0: página de inicio

La pantalla 0 corresponde al inicio de sesión de la aplicación *MRepatt*. Dicha pantalla le pide al servidor la página índice de la aplicación, que la genera el módulo índice.

El módulo Indice sólo trata un caso de uso: la petición por parte del navegador de la página índice , como se muestra en la figura 6.12. El objetivo de este módulo es es conseguir que el usuario tenga en la página principal un componente visual que corresponda a una estructura de árbol. Dentro de este árbol tenga total libertad para elegir los grupos a comparar y dentro de cada grupo los cromosomas que crea oportunos. A este componente le hemos llamado *"menú de especies"*, y como se ha explicado, se

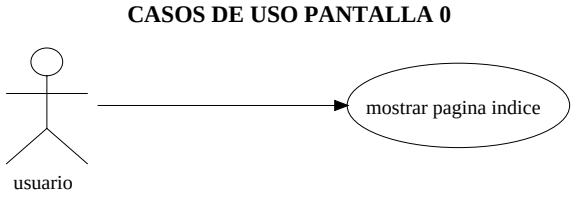


Figura 6.12: Diagrama de casos de uso de la pantalla 0

ha utilizado javascript. Se ha implementado dicho componente para que sea capaz de funcionar en cualquier navegador con el plugin de javascript instalado.

6.5.1. Caso de uso: mostrar página índice

Podemos ver el diagrama de secuencia en la figura 6.13. La secuencia de acciones para el caso expuesto y obtener la página principal es la siguiente:

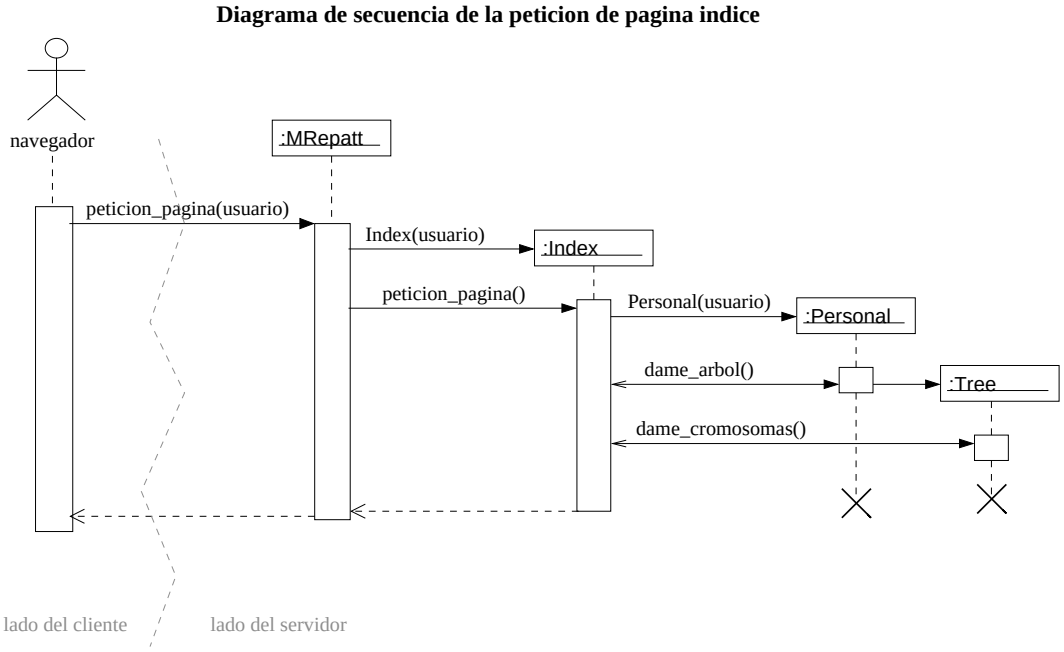


Figura 6.13: Diagrama de secuencia de mostrar página índice

1. Se hace la petición al servidor de la página índice. La petición lleva como parámetro el módulo al que va dirigido y opcionalmente el identificador de usuario (su email).

2. La clase *MRepatt* lee la petición y la dirige al módulo *RepattIndex*.

3. La clase *Index* le pide a la clase *Personal* el árbol filogenético del usuario. Esta clase genera el código del árbol que debe leer la clase *Tree*. Si en los parámetros aparece el identificador de usuario, la clase *Personal* añade los cromosomas de este usuario en el código, en otro caso, genera el código por defecto del árbol, únicamente con los cromosomas de *ALGGEN*. Una vez que ha generado el código, se llama a la clase *Tree*, que lo interpreta, y carga el árbol filogenético.

4. La clase *Index* le pide a la clase *Tree* los cromosomas para escribirlos en la página que devuelve al usuario.

6.6. Pantalla 1: página índice

La página principal permitirá al usuario elegir los cromosomas que tenemos guardados en la base de datos y los patrones para realizar la búsqueda. Los objetivos para esta pantalla son los siguientes:

- Crear la página *html* a partir de la lista de cromosomas de la base de datos.
- Incorporar en la página *html* un elemento visual que permita al usuario, elegir grupos de cromosomas para realizar la búsqueda. Dicho elemento debe ser intuitivo y de fácil uso.
- Permitir al usuario trabajar indistintamente con los cromosomas que éste ha incorporado en la base de datos y los propios de *ALGGEN*.

Podemos ver una representación gráfica de la página índice que se presenta en la figura 6.14. Se puede apreciar el "menú de especies" como el componente rotulado con un 1. Hay explicación detallada del funcionamiento de este componente en la página 88 del manual, en la sección 7.4.

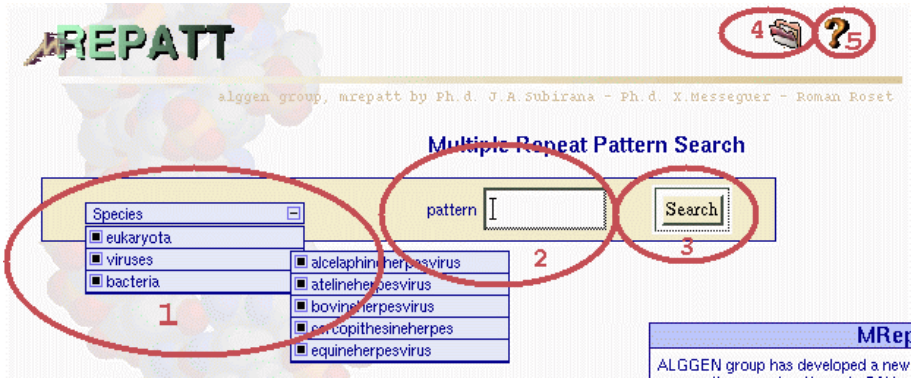


Figura 6.14: Pantalla de la página principal

Desde la página índice, el usuario puede realizar una consulta o modificar los cromosomas de su base de datos. Podemos ver los casos de uso en la figura 6.15.

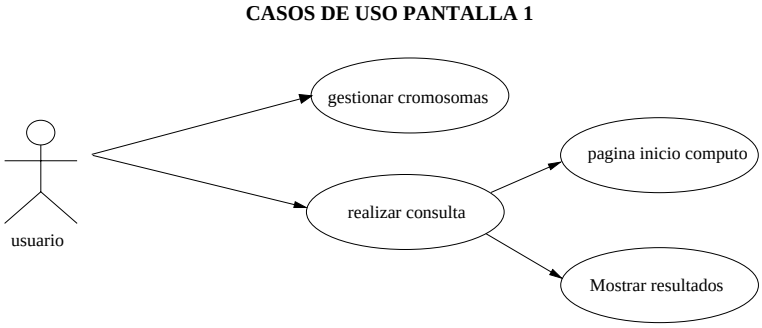


Figura 6.15: Diagrama de casos de uso de la pantalla 1

6.6.1. Caso de uso: gestionar cromosomas

El caso de uso "gestionar cromosomas" lo implementa el módulo Personal y con esta petición se obtiene la página de gestión de ficheros personales. Podemos ver el diagrama de secuencia en la figura 6.20. La secuencia de acciones para obtener dicha la página es:

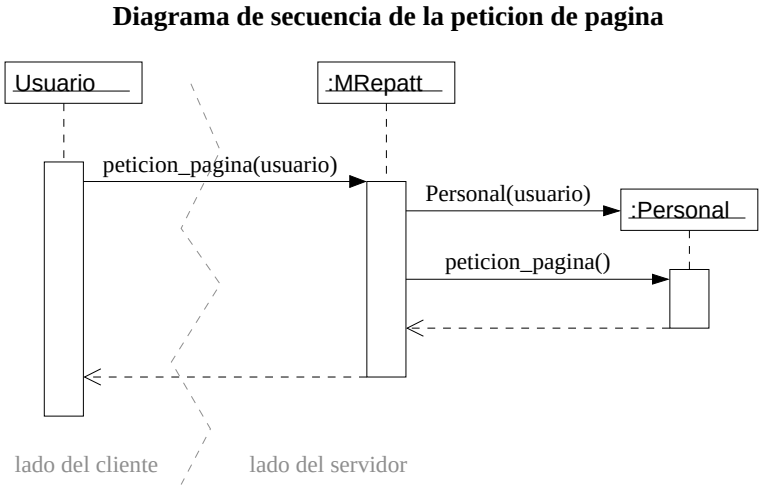


Figura 6.16: Diagrama de secuencia de gestionar cromosomas

1. Se hace la petición al servidor de la página de gestión de ficheros personales. La petición lleva como parámetro el módulo al que va dirigido y el identificador de usuario (su email).
2. La clase *MRepatt* lee la petición y la dirige al módulo *RepattPersonal*.
3. La clase *Personal* recibe la petición, lee el fichero "files.dat" del usuario (explicado la sección 6.3.1) para obtener los ficheros actuales del usuario, y escribe la página resultado.

6.6.2. Caso de uso: realizar consulta

[Falta Realizar Consulta]

6.7. Pantalla 2: página de gestión de ficheros personales

La pantalla de gestión de ficheros la implementa el módulo Personal. Ss debe presentar una interficie capaz de dar de alta y baja ficheros en la base de datos. Posteriormente, se trabajará con estos ficheros en la pantalla índice (la pantalla 1). Despues de esta petición el usuario recibe una página como la que aparece en la figura 6.21. Encontraremos la explicaciones pertinentes a su uso en la sección 7.6 del manual.

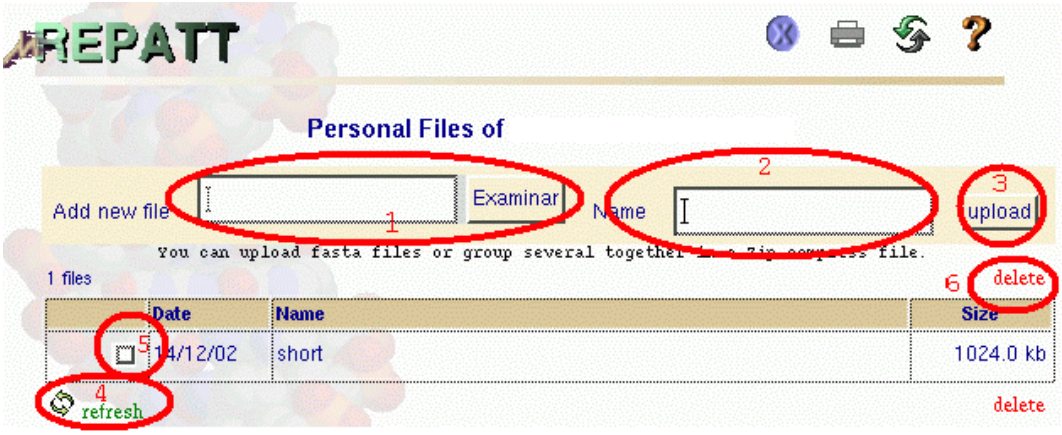


Figura 6.17: Pantalla de la página del módulo personal

Podemos ver los tres casos de uso en el diagrama de la figura 6.19. Los objetivos para esta pantalla son los siguientes:

- Crear una página *html* que gestione los archivos de un usuario.
- Permitir al usuario enviar archivos de cromosomas al servidor.
- Permitir al usuario eliminar los archivos que crea oportunos.

6.8. MRepatt: Módulo Personal

La finalidad del módulo *Personal* es la gestión de los cromosomas personales de cada usuario. Ss debe presentar una interficie capaz de dar de alta y baja ficheros en la base de datos. Posteriormente, se trabajará con estos ficheros en el módulo explicado en el apartado anterior. Los objetivos para este módulo son los siguientes:

- Crear una página *html* que gestione los archivos de un usuario.

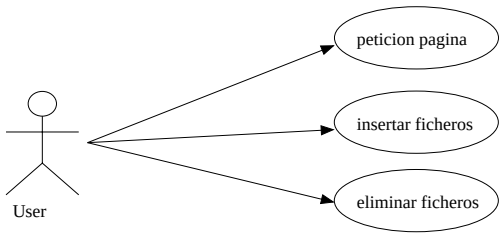


Figura 6.18: Diagrama de casos de uso del módulo Personal

- Permitir al usuario enviar archivos de cromosomas al servidor.
- Permitir al usuario eliminar los archivos que crea oportunos.

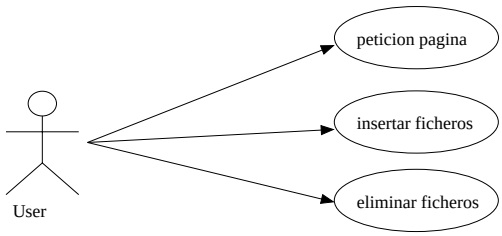


Figura 6.19: Diagrama de casos de uso del módulo Personal

Podemos ver los tres casos de uso en el diagrama de la figura 6.19. La secuencia de acciones para obtener la página de gestión de ficheros personales es:

1. Se hace la petición al servidor de la página de gestión de ficheros personales. La petición lleva como parámetro el módulo al que va dirigido y el identificador de usuario (su email).
2. La clase *MRepatt* lee la petición y la dirige al módulo *RepattPersonal*.
3. La clase *Personal* recibe la petición, lee el fichero "files.dat" del usuario (explicado la sección 6.3.1) para obtener los ficheros actuales del usuario, y escribe la página resultado.

Podemos ver el diagrama de secuencia en la figura 6.20. Despues de esta petición el usuario recibe una página como la que aparece en la figura 6.21. Encontraremos la explicaciones pertinentes a su uso en la sección 7.6 del manual. Para insertar uno o varios cromosomas el usuario tiene dos vias:

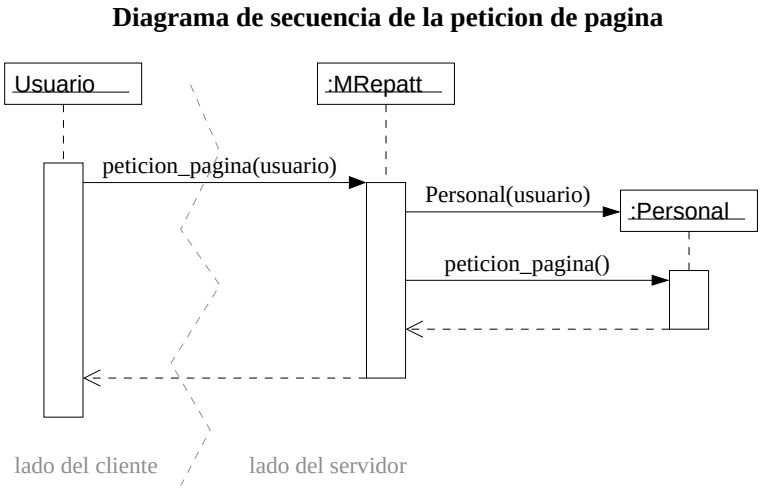


Figura 6.20: Diagrama de secuencia de la petición de página del módulo Personal

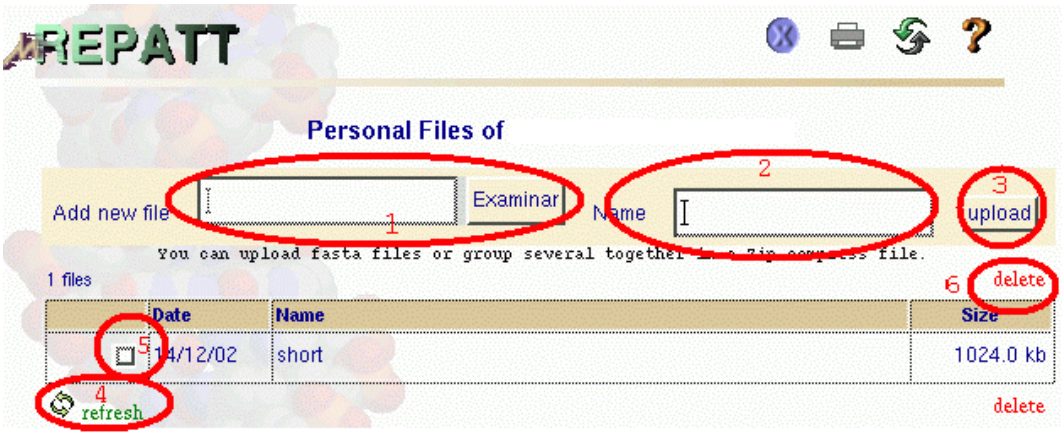


Figura 6.21: Pantalla de la página del módulo personal

- Enviar al servidor un fichero en formato *Fasta*.
- Comprimir en un fichero en formato *zip* varios ficheros en formato *Fasta* y enviarlos al servidor.

La secuencia de acciones para insertar un fichero con uno o varios cromosomas[3] es la siguiente:

- Se envia la petición de inserción del fichero al servidor. La petición tiene como parámetro el usuario y el fichero que se envia.

[3] entiéndase cromosoma como cada fichero *Fasta*

- La clase *MRepatt* lee la petición y la dirige al módulo *RepattPersonal*.
- La classe *Personal* recibe la petición, recoge el fichero, lo descomprime y para cada uno de los cromosomas (si el fichero no está comprimido, solo efectua la operación para el cromosoma que se ha enviado) realiza las siguientes acciones:
  - Asigna un identificador nuevo al cromosoma.
  - Escribe el fichero en la base de datos.
  - Modifica los ficheros "files.dat" y "append.conf", explicados en la sección 6.3.1.
- La clase *MRepatt* pide la página de gestión de ficheros personales. Dicha página ahora contendrá los nuevos cromosomas introducidos. La página que se envia también contiene código en *javascript* para que se ejecute cuando se cargue la página en el cliente. Dicho código obliga a refrescar la página de inicio que contiene el "menu de especies", ya que se deben cargar en éste los nuevos cromosomas introducidos por el usuario.

Podemos ver el diagrama de secuencia en la figura 6.22.

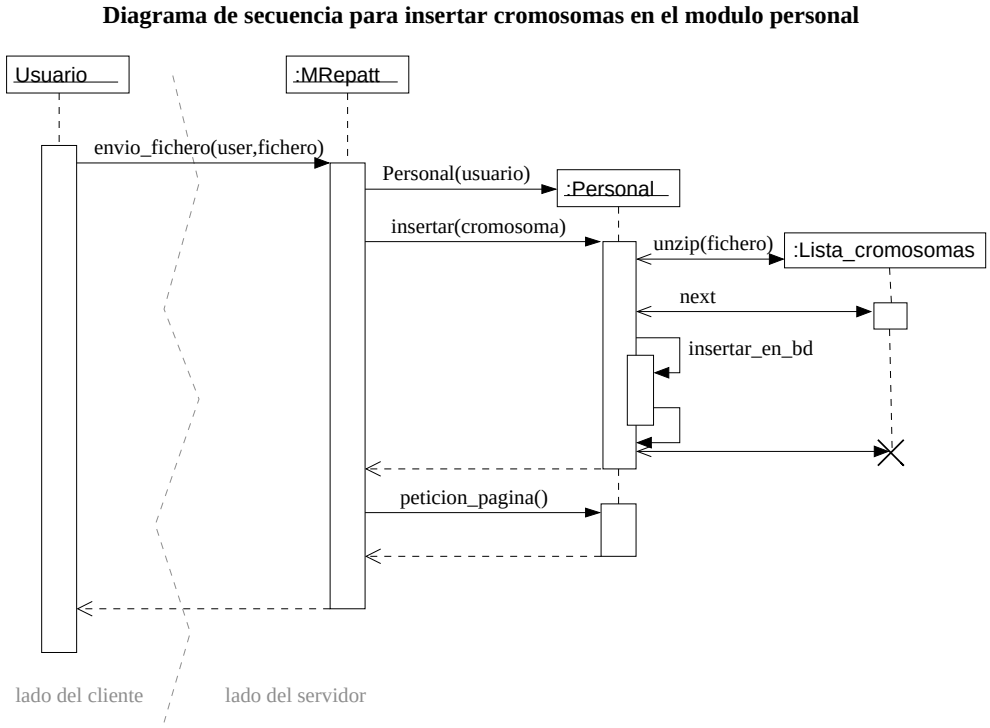


Figura 6.22: Diagrama de secuencia de la petición de inserción del módulo Personal

Cuando el usuario quiera eliminar alguno de los cromosomas que introdujo, deberá seguir los siguientes pasos:

1. Seleccionar todos los ficheros que se quieren eliminar. Esta operación se efectua unicamente en el lado del cliente, y genera una lista con los identificadores de los ficheros seleccionados.

2. Se envia al servidor la lista de los identificadores de los ficheros que se quieren borrar.

Una vez se han seleccionado los ficheros a eliminar y el navegador ha confeccionado la lista de identificadores, se procesa la petición al servidor. La secuencia de acciones en el servidor es la siguiente:

- Se envia la petición de eliminación de los fichero al servidor. La petición tiene como parámetro el usuario y la lista de identificadores de los cromosomas a eliminar.
- La clase *MRepatt* lee la petición y la dirige al módulo *RepattPersonal*.
- La classe *Personal* recibe la petición, recoge la lista y para cada uno de los cromosomas a eliminar realiza las siguientes operaciones:
  - Llama a la clase PatternMap y elimina los resultados de los cromosomas de la base de datos de resultados.
  - Modifica el fichero "files.dat" y "append.conf", explicados en la sección 6.3.1.
- La clase *MRepatt* pide la página de gestión de ficheros personales. Dicha página ahora contendrá los nuevos cromosomas introducidos. La página que se envia también contiene código en *javascript* para que se ejecute cuando se cargue en el cliente. Dicho código obliga a refrescar la página de inicio que contiene el "menu de especies", ya que se deben eliminar los cromosomas de dicho menú.

Cuando se envia la petición al servidor aparece una página de espera mientras el la página principal se refresca para que aparezca el "menú de especies" sin los cromosomas que ha eliminado el usuario. Podemos ver el diagrama de secuencia en la figura 6.23.

6.9. MRepatt: Módulo de computación

MRepatt guarda en la base de datos todos los resultados que se han ido computando, con la finalidad de mostrar dichos resultados sin tener que volver a procesarlos, si se vuelven a consultar. El módulo de computación se encarga de realizar este cómputo la primera vez que se realiza una consulta. Los objetivos para este módulo son los siguientes:

- Realizar el cómputo de la consulta que ha pedido el usuario.
- Mostrar información al usuario de los cromosomas y patrones que se deben procesar, antes de iniciar el cómputo.
- Mostrar una página de monitorización de los procesos.

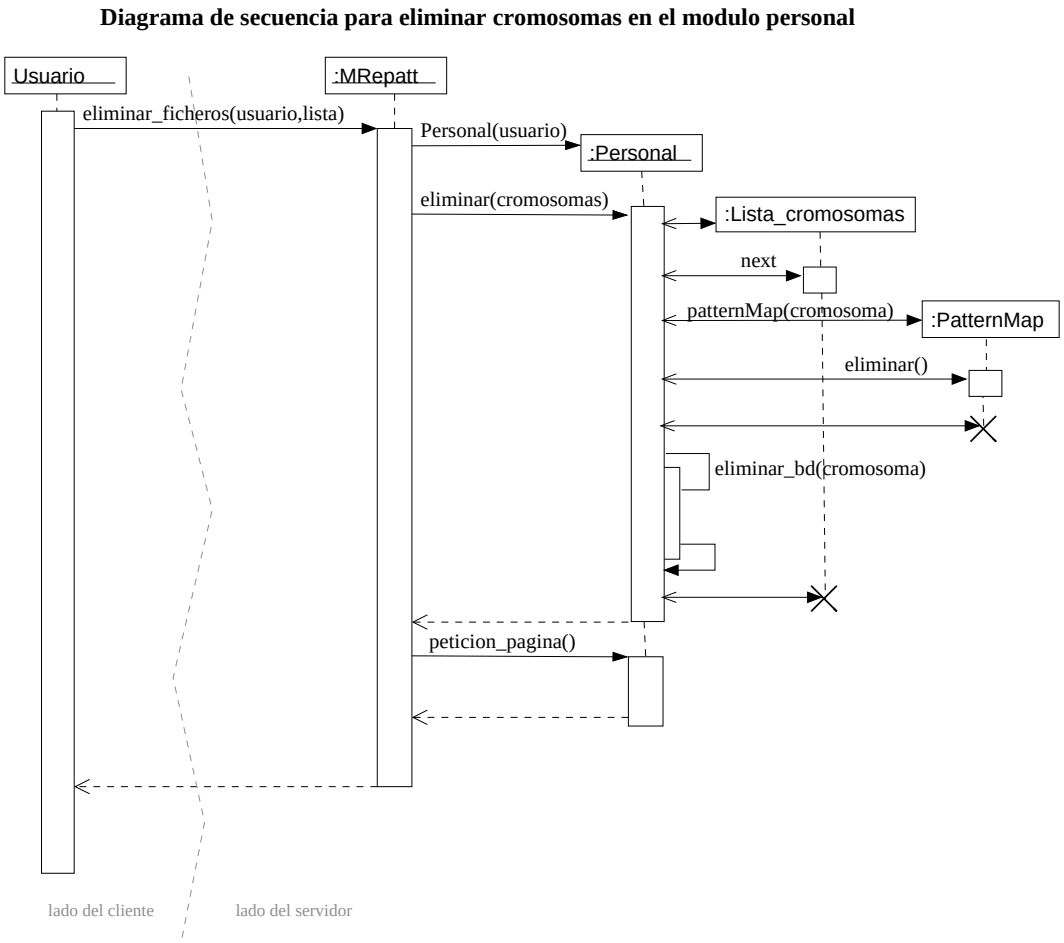


Figura 6.23: Diagrama de secuencia de la petición de eliminación del módulo Personal

- Enviar un correo al usuario cuando el cómputo finaliza.

Para facilitar la descripción de este módulo, se explicará cada uno de los pasos cronológicos que sigue el proceso del cómputo. En los siguientes pasos, se ha utilizado la siguiente nomenclatura:

- Par a búscar: Cada par $\langle$ *cromosoma, patrón* $\rangle$ que forma parte de una consulta.
- Conjunto a computar: Conjunto de todos los pares de búsqueda cuyos resultados no se han encontrado en la base de datos.
- Consulta: Conjunto de los pares a buscar.

La computación se basa en recoger los pares del conjunto a computar y hacer la llamada al programa *Repatt* con ese par como parámetro. Como ya se ha explicado

en anteriores capítulos, el programa *Repatt* realiza la búsqueda de las repeticiones de un patrón en un solo cromosoma. Dado que el usuario realiza la consulta para obtener los resultados totales de un grupo de cromosomas, el primer paso, para el módulo de computación es separar la consulta en pares a buscar. El módulo de computación no calcula los resultados totales ni los visualiza, sino que se ha dejado esta funcionalidad para el módulo de Resultados que se explicará en la siguiente sección.

6.9.1. Detección del conjunto a computar de la consulta

El usuario realiza la consulta en la página índice. Desde esta página el servidor recibe la petición y cuando lo hace, se busca en la base de datos, si los resultados para cada par a buscar ya han sido procesados. En este apartado se estudia el caso de uso

Parte III

Apéndice

Capítulo 7

Manual de usuario de MREPATT

frase

autor, origen, 1781

En las siguientes secciones se explica detalladamente cómo funcionan cada una de las diferentes pantallas del programa *MREPATT*.

7.1. Página principal

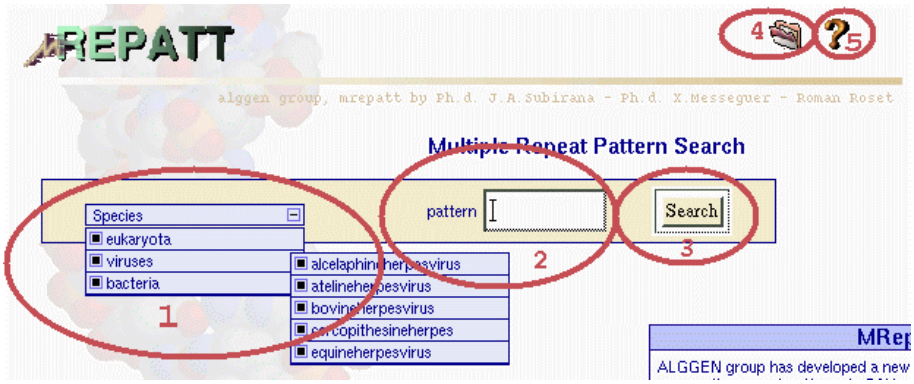


Figura 7.1: Pantalla de la página principal

Las acciones que se pueden realizar en la página principal son:

- Realizar una búsqueda.
- Abrir una ventana de archivos personales.

Para ver todos los iconos que aparecen en la aplicación y su significado lea la sección 7.2.

7.1.1. Realizar una búsqueda

Los pasos a seguir para realizar una búsqueda son los siguientes:

1. Escoger los cromosomas o especies a comparar en el "menú de especie", el *componente 1*. para ver mas información sobre el funcionamiento del "menú de especies" lea la sección 7.4.

2. Introducir uno o varios patrones en el *componente 2*. Para ver mas información sobre el uso de patrones, patrones válidos y el tipo de búsqueda que se realiza (búsqueda del patrón introducido y su complementario) lea la sección 7.3.

3. Pulsar el botón "Search" en el *componente 3*.

7.1.2. Abrir una ventana de archivos personales

Esta funcionalidad permite incorporar sus ficheros para poder realizar una búsqueda.

Pulsando el botón correspondiente al *componente 4*, se abrirá una ventana preguntando su correo electrónico, y después de haberlo introducido aparecerá la ventana de "Gestión de ficheros personales".

En ella usted podrá gestionar sus propios ficheros, es decir, introducir ficheros nuevos y eliminarlos cuando haya finalizado su análisis.

Los ficheros que introduzca en la ventana de "Gestión de ficheros personales" irán apareciendo en el *componente 1*, y usted podrá trabajar conjuntamente con los de nuestra base de datos.

7.2. Iconos usados en MREPATT

A continuación se detalla la lista de iconos y su descripción:

7.2.1. Iconos Generales

Ayuda
Cuando pulse sobre este icono se abrirá una ventana con la ayuda en línea relacionada con la página en curso.

Nueva ventana
Este icono aparece en frames. Cuando pulse sobre este icono se abrirá una ventana nueva con la misma información que el frame en curso.

Imprimir
Imprime la ventana en curso. Cuando pulse este botón se abre el dialogo de la impresión.

Cerrar ventana
Cuando pulse este botón se cierra la ventana en curso.

Nueva gestión personal de archivos
Cuando pulse este botón aparecerá una nueva ventana que le preguntará su e-mail, para poder cargar la ventana de archivos personales.

Abrir la ventana de gestión personal de archivos
Si usted anteriormente ya introdujo su e-mail, cuando pulse este botón se le abrirá la ventana con sus archivos personales.

7.2.2. Iconos en la ventana de Gestión personal de archivos

Refrescar información
Este botón aparece en la ventana de gestión personal. Cuando haya realizado cambios en los ficheros (marcar para borrar o introducción de ficheros), pulse este botón para guardar los cambios.

7.2.3. Iconos en la ventana de Zoom

Desplazamiento largo a la izquierda
Hace que se muestre la región inmediatamente a la izquierda de la actual y con el mismo tamaño que la actual.

Desplazamiento corto a la izquierda
Desplaza la región mostrada a la derecha, haciendo que una parte de la región actual se siga viendo. El número de bases desplazadas depende de la longitud actual de la región. Para regiones de 100 bases, el desplazamiento es siempre de una base.

Desplazamiento corto a la derecha
Desplaza la región mostrada a la izquierda, haciendo que una parte de la región actual se siga viendo. El número de bases desplazadas depende de la longitud actual de la región. Para regiones de 100 bases, el desplazamiento es siempre de una base.

Desplazamiento largo a la izquierda
Hace que se muestre la región inmediatamente a la derecha de la actual y con el mismo tamaño que la actual.

Zoom Out
Muestra una región de longitud mayor que la región actual. En la nueva imagen, la región actual será una de las 10 regiones que la dividen.

7.3. Patrones

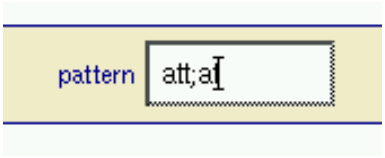


Figura 7.2: Pantalla de la región de entrada del patrón

MRepatt permite la introducción de uno o varios patrones. Cada uno de los patrones introducidos produce búsquedas diferentes y sus resultados se presentan en la misma página de resultados con tablas comparativas.

En esta sección se explica:

- Tipos de patrones permitidos.
- Tipo de búsqueda que se realiza.
- Cómo introducir patrones.

7.3.1. Tipos de patrones permitidos

Los patrones permitidos son aquellos formados íntegramente por las cuatro bases: A,C,G,T. Se pueden introducir en mayúsculas o minúsculas indistintamente.

En la versión presente no se permiten símbolos comodín como puedan ser los de la "IUB". No obstante es posible que en versiones futuras se contemple.

Aquellos patrones introducidos que sean repeticiones de un subpatrón (por ejemplo "atat" o "aa") no son válidos y se procederá la búsqueda cambiando ese patrón por su subpatrón, previo aviso al usuario. Por ejemplo, si introducimos "cgcg", MREPATT buscará el patrón "cg".

7.3.2. Tipo de búsqueda que se realiza

MRepatt busca conjuntamente las ocurrencias del patrón introducido y de su complementario.

El número de ocurrencias devuelto para cada repetición es la suma de las ocurrencias de los dos patrones. Cuando el complementario (o una permutación circular de éste) sea igual que el patrón introducido, MRepatt sólo busca el patrón introducido y multiplica por dos el número de ocurrencias. Eso pasará por ejemplo con patrones como "at" o "atcg"(cuyo complementario "cgat" es una permutación de "atcg").

Por ejemplo, si el cromosoma fuera "aattcgcgtaaa", y el patrón introducido fuera "a", el número de ocurrencias acumuladas para cada repetición sería el siguiente:

- Repetición 1 ("a"+"t"): 8
- Repetición 2 ("aa" + "tt"): 4
- Repetición 3 ("aaa" + "ttt"): 1

Mientras que el número de ocurrencias únicas seria:

- Repetición 1 ("a"+"t"): 1
- Repetición 2 ("aa" + "tt"): 2
- Repetición 3 ("aaa" + "ttt"): 1

7.3.3. Cómo introducir patrones

Para introducir un patrón lo escribimos en el componente de texto "pattern". Por ejemplo para buscar el patrón "AG" (y su complementario) introducimos "ag" o "ct" en el componente de texto "pattern."

Para introducir más de un patrón, los separamos con ";". Si queremos comparar los patrones "ag" y "att" tenemos las cuatro posibilidades (sin contar las variaciones en el orden de introducción de los patrones):

1. pattern: ag;att

2. pattern: ag;aat

3. pattern: ct;att

4. pattern: ct;aat

7.4. Menú de especies

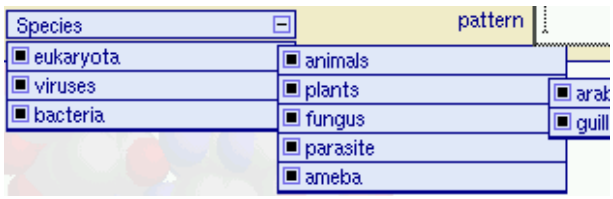


Figura 7.3: Pantalla del menú de especies

El menú de especies permite escoger uno o varios grupos de cromosomas para comparar las ocurrencias de los patrones repetidos.

Con el menú de especies podremos:

- Agrupar cromosomas de varias especies para analizarlos como si de un solo cromosoma se tratase.

- Escoger varios grupos para compararlos.

7.4.1. Agrupar cromosomas

La agrupación de cromosomas permite realizar la búsqueda en un conjunto de cromosomas y darnos los resultados para ese conjunto.

MRepatt interpreta este conjunto como un solo fichero resultante de la concatenación de cada uno de los cromosomas seleccionados.

Los conjuntos que podemos formar vienen dados por la estructura de árbol del menú. Para agrupar dos especies tenemos que pinchar siempre el primer padre que tienen en común. Por ejemplo, para agrupar cromosomas de "homo sapiens" y de "drosophila" tenemos que pinchar el grupo "animals". Para agrupar cromosomas del grupo "ameba" y de "homo sapiens" debemos pinchar el grupo "eukaryota".

Cuando seleccionamos un grupo todas sus ramas pasan a estar seleccionadas. Podemos eliminar ramas o cromosomas de un grupo seleccionado pinchando sobre los subgrupos o cromosomas que queramos eliminar del grupo. Cuando pinchemos un subgrupo seleccionada todos los cromosomas y las ramas que cuelguen de éste quedaran deseleccionadas. Por ejemplo, para agrupar los cromosomas 21 y 22 de "homo sapiens", pinchamos en "homo sapiens" y posteriormente pinchamos en el cromosoma 20 para

deseleccionarlo. Si queremos agrupar todos los cromosomas disponibles de "homo sapiens" y "mus musculus", pinchamos en "animals" y posteriormente pinchamos todas las especies de "animals" menos "homo sapiens" y "mus.musculus" para deseleccionarlas.

Para seleccionar un grupo tenemos que seguir estos pasos:

1. Pinchar en el muen el grupo que contenga todos los cromosomas que queramos agrupar. Con esta acción se seleccionan todos las ramas que cuelgan del grupo seleccionado. En el muen se podrán ver los grupos y sus sub-grupos seleccionados con una marca roja a la izquierda.

2. Pinchar sobre las ramas o cromosomas que cuelgan del grupo seleccionado para deseleccionarlas.

7.4.2. Escoger varios grupos

Todos aquellos grupos seleccionados que no tengan ningún padre común seleccionado, serán grupos diferentes. MRepatt analizará cada grupo diferente por separado, y presentará tablas y gráficas comparativas para esos grupos.

Para comparar grupos sólo tenemos que pinchar esos grupos en el menú. Por ejemplo, para comparar los grupos "eukariota" y "virus", pinchamos en el menú el grupo "eukariota" y "virus". Podemos eliminar especies o cromosomas dentro cada grupo seleccionado, como se ha explicado en el apartado anterior. Por ejemplo, para comparar los cromosomas I y II de "S.cerevisiae" con los cromosomas 21 de "homo sapiens" podemos pinchar directamente el cromosomas 21 de "homo sapiens" y seleccionar conjuntamente los dos cromosomas de "S.cerevisiae" pinchando en la especies "S.cerevisiae" del menú, y entonces volver pinchar sobre todos los cromosomas que cuelgan de "S.cerevisiae" menos el I y el II para deseleccionarlos.

7.5. Página de E-mail

7.5.1. Componentes de la página

La introducción de su e-mail es un requisito para poder trabajar con sus ficheros personales. El email que nos facilite servirá únicamente para:

- Identificar sus ficheros.
- Enviarle un e-mail de aviso para informarle cuando haya finalizado una búsqueda.

Le garantizamos que no se hará un uso fraudulento de su dirección, ni se proporcionará está información a terceras personas. Tampoco se le enviará publicidad ni ninguna otra información que no sea la de avisarle sobre la finalización de una búsqueda en MRepatt.

1. Editor para introducir su email. Un ejemplo de valor corrector para este campo es: "mrepat.example@yahoo.com".

Figura 7.4: Pantalla de la página de e-mail para entrar en el módulo personal

2. Pulse este botón cuando haya introducido su email en el *componente 1* para pasar a la página de "Gestión de ficheros personales".

3. Pulse este botón para cerrar la ventana y cancelar la operación.

4. Botón para mostrar esta ayuda.

7.5.2. Cookies

Si usted tiene activada en el navegador la opción de grabar cookies, esta página graba una con su dirección de email. El objetivo de esta cookie es iniciar las próximas sesiones de *MRepatt* con sus ficheros personales cargados en el "menú de especies" de la página inicial. Para ver mas información sobre el funcionamiento del "menú de especies" lea la sección 7.4.

Para ver todos los iconos que aparecen en la aplicación y su significado lea la sección 7.2.

7.6. Gestión de ficheros personales

En la página de gestión de ficheros personales usted podrá:

- Insertar ficheros.
- Eliminar ficheros insertados.

7.6.1. Formato de los archivos a enviar

Los ficheros que se inserten han de ser de tipo *"FASTA"*. Los ficheros *"FASTA"* están formados por una etiqueta y la secuencia. No obstante, la parte de la etiqueta es opcional. Ejemplos de formato fasta son:

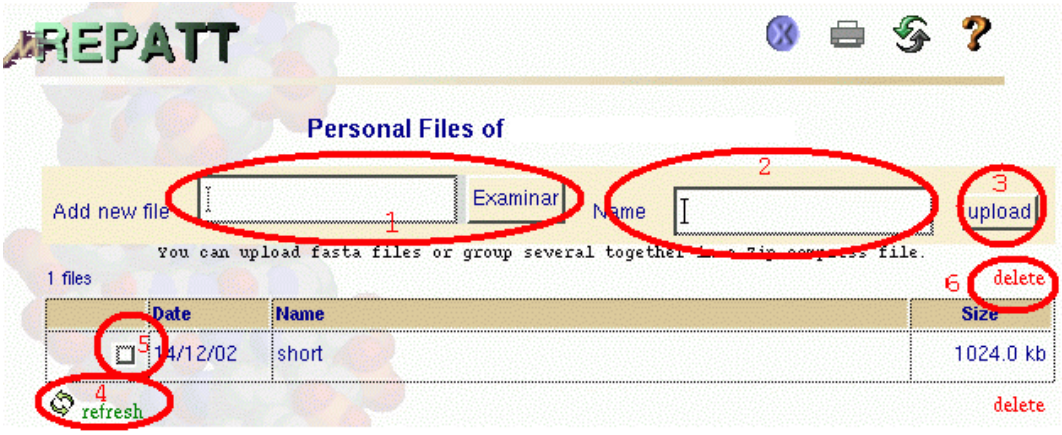


Figura 7.5: Pantalla de la página del módulo personal

- Ejemplo con etiquetas:

```
>sequence#1
ACGTGACTAGCTACAGNTGCTACANTCGATCAGTAGNGT
CGGCCGCGCGTAGCTAGCAATACGACNNTACGTAGCctg
atagtcagctagcngctagtcaGCTAGCATGACG
>sequence#2
gtacgatcgacgcgcgcgcgcatatatatatatattctc
tctcctcggtcagtcagctacgtatatatatatatatnn
atagtcagctagcngctagtcaGCTAGCATGACG
```

- Ejemplo sin etiquetas:

```
ACGTGACTAGCTACAGNTGCTACANTCGATCAGTAGNGT
CGGCCGCGCGTAGCTAGCAATACGACNNTACGTAGCctg
atagtcagctagcngctagtcaGCTAGCATGACG

gtacgatcgacgcgcgcgcgcatatatatatatattctct
ctcctcggtcagtcagctacgtatatatatatatatnn
atagtcagctagcngctagtcaGCTAGCATGACG
```

Hay dos formatos posibles para insertar los ficheros: comprimidos o sin comprimir. Cuando se envía un archivo comprimido, se pueden enviar mas de un fichero dentro del comprimidos. No obstante, todos los ficheros dentro del paquete comprimido deben ser de tipos *"FASTA"*.

7.6.2. Insertar ficheros no comprimidos

Los pasos a seguir para enviar directamente un fichero "FASTA" son los siguientes:

1. Pulse el botón "Examinar" del *componente 1* y elija el fichero que quiera utilizar.

2. Introduzca un nombre identificativo para ese fichero en el *componente 2*.

3. Pulse el botón "upload" del *componente 3*. La operación "upload" puede durar varios minutos , en función del tamaño del fichero que quiera insertar. Durante esta operación, el puntero del ratón cambiará forma (habitualmente por un reloj).

Vuelva al paso 1 si quiere insertar mas ficheros. Cuando haya finalizado la inserción de todos los ficheros que crea oportunos lea la sección "Refrescar el Menú de especies" de esta misma página de ayuda.

7.6.3. Insertar ficheros comprimidos

Los ficheros una vez comprimidos se insertan de la misma forma que los "FASTA". El formato de compresión es en "zip". Algunos de los programas que comprimen en zip son:

- Linux/Unix: zip.
- Windows: winzip

Para comprimir siga estos pasos:

1. Cree una carpeta nueva. Por ejemplo: "mrepatt_myfiles".

2. Copie en esa carpeta los ficheros en formato *FASTA* que quiere insertar.

3. Comprima la carpeta en un solo fichero. Por ejemplo "mrepatt_myfiles.zip".

A continuación siga los pasos del apartado "Insertar ficheros no comprimidos", y elija el fichero "mrepatt_myfiles.zip" de nuestro ejemplo, en el paso 1.

7.6.4. Refrescar el Menú de especies

Una vez haya finalizado la inserción de los ficheros, éstos deben aparecer en el "menú de especies" para poder trabajar con ellos.

Para ello pulse el link "refresh" del *componente 4*, o el icono con el mismo dibujo.

Este proceso puede tardar unos breves segundos. Mientras dure se le aparece una página de espera.

Una vez cargado el "menú de especies" con sus ficheros en la página principal, desaparece la página de espera, y usted puede volver a la página principal para trabajar con los ficheros que haya subido.

No es necesario cerrar la página de "Gestión personal de archivos" durante el proceso de trabajo en la página principal, pudiendo insertar o eliminar sus ficheros en todo momento de la sesión.

7.6.5. Eliminar ficheros

Para eliminar uno o varios ficheros siga estos pasos:

1. Pulse el botón del *componente 5* del fichero que quiera eliminar.
2. Vuelva al paso 1 para cada fichero que quiera eliminar.
3. Pulse el link "delete" del *componente 6*.

Cuando pulse el link "delete" se le aparecerá una página de espera mientras el "menú de especies" de la página principal se refresca y elimina la posición de los ficheros que ha eliminado.

Para ver todos los iconos que aparecen en la aplicación y su significado lea la sección 7.2.

7.7. Introducción de email para el cómputo

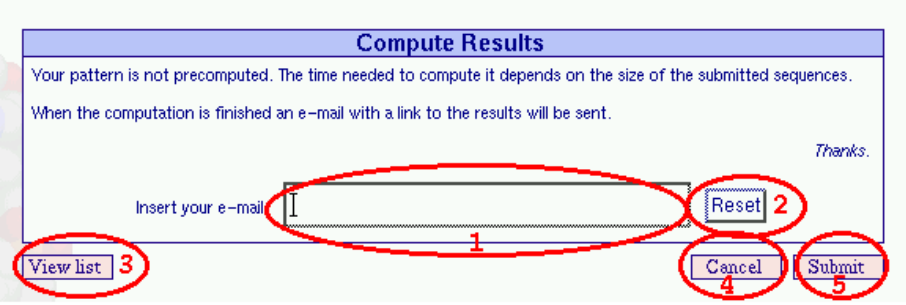


Figura 7.6: Pantalla de la página de email para el cómputo

MRepatt guarda una base de datos con todos los patrones buscados en los cromosomas del menú de especies. Los patrones de longitud uno y dos estarán siempre calculados, al igual que la mayoría de longitud tres.

No obstante, hay patrones que aún nadie a calculado anteriormente.

Cuando hay patrones que no han sido calculados para los cromosomas que se piden, aparece una ventana con la información de la imagen adjunta.

Esta ventana informa que en su consulta hay patrones que necesitan ser calculados, y le pide su e-mail. La introducción del email es opcional, no obstante se recomienda si la suma del tamaño de los cromosomas a computar es elevada (mas de 200 Mb.), ya que el computo podría tardar unos minutos. Con la introducción del email usted recibirá un correo con un link a su petición cuando el proceso haya finalizado.

A continuación se detallan las funciones de esta página, que serán explicadas con mas detalle en los apartados siguientes:

- Consultar el tamaño total de los cromosomas ha computar.

- Abortar el cálculo.
- Seguir con el cálculo.

7.7.1. Consultar el tamaño total de los cromosomas ha computar

Si su petición comprende mas de un cromosoma podría darse el caso que sólo alguno de los cromosomas que componen su petición se tengan que computar. Para ver la lista de los cromosomas que faltan por computar pinche sobre el link "View list" del *componente 3* y se le aparecerá una nueva ventana con esta información.

No aparece ninguna estimación del tiempo, puesto que entre diferentes ejecuciones existen diferencias, y la estimación sería imprecisa. No obstante para que tenga una idea, las velocidades suele ser de 1Mb/segundo.

7.7.2. Abortar el cálculo

Para abortar el cálculo y cerrar la ventana debe pulsar sobre el link "Cancel" del *componente 5*.

7.7.3. Seguir con el cálculo

Para seguir con el cálculo introduzca su e-mail en el texto del *componente 1*, aunque como ya se ha dicho es opcional. Dado que todos los resultados pedidos por los usuarios se guardan en una base de datos, si no introduce su e-mail tendrá que repetir periódicamente su petición en la página principal para saber si los cálculos han finalizado, y ver los resultados.

El email que nos facilite servirá únicamente para:

- Enviarle un e-mail de aviso cuando haya finalizado una búsqueda, con un link hacia los resultados.

Le garantizamos que no se hará un uso fraudulento de su dirección, ni se proporcionará está información a terceras personas. Tampoco se le enviará publicidad ni ninguna otra información que no sea la de avisarle sobre la finalización de una búsqueda en MRepatt.

Después de introducir su e-mail pulse el link "Submit" en el *componente 5*, entonces se cargará una página que le preguntará si desea esperar a recibir el correo o prefiere monitorizar el proceso para posteriormente ver los resultados.

Si introduce incorrectamente su e-mail y desea borrar el contenido del campo de texto del *componente 1*, pulse el link "Reset" del *componente 4*.

7.8. Lista de cromosomas a computar

En esta página se muestran para cada patrón la lista de cromosomas que faltan a computar.

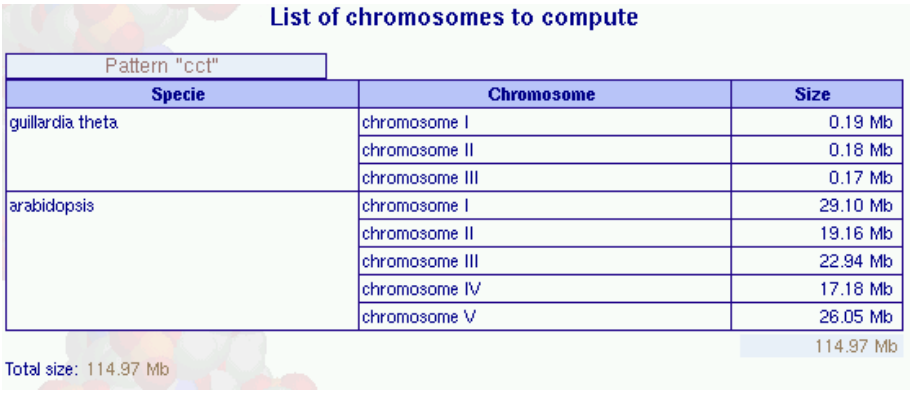


Figura 7.7: Pantalla de la página de listado de los cromosomas a computar

Para cada patrón aparece una tabla con cada uno de los cromosomas que faltan y la especie a la que pertenecen. Con cada cromosoma también aparece su tamaño en Mb.

Al final de cada tabla aparece el tamaño total que hay que computar para el patrón identificador de la tabla.

Al final de la página se muestra el tamaño total que debe procesarse. Este tamaño debe servir como referencia al tiempo necesario de cómputo.

No aparece ninguna estimación del tiempo, puesto que entre diferentes ejecuciones existen diferencias, y la estimación sería imprecisa. No obstante para que tenga una idea, las velocidades suelen ser de 1Mb/segundo. Así en nuestro ejemplo, el tiempo estimado que podría tardar estaría alrededor de los dos minutos.

7.9. Iniciación del cómputo

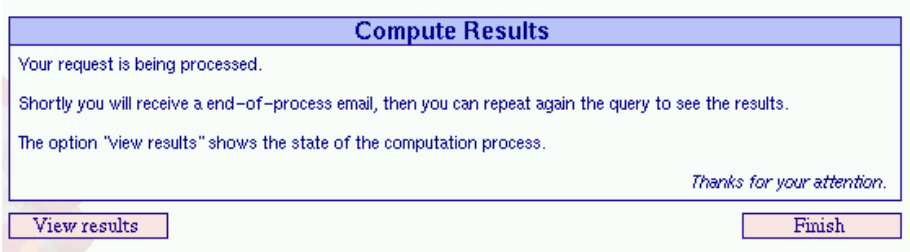


Figura 7.8: Pantalla para la inicialización del cómputo

Esta página le informa que su petición se esta procesando.

La duración del cómputo depende del tamaño total de los cromosomas a computar.

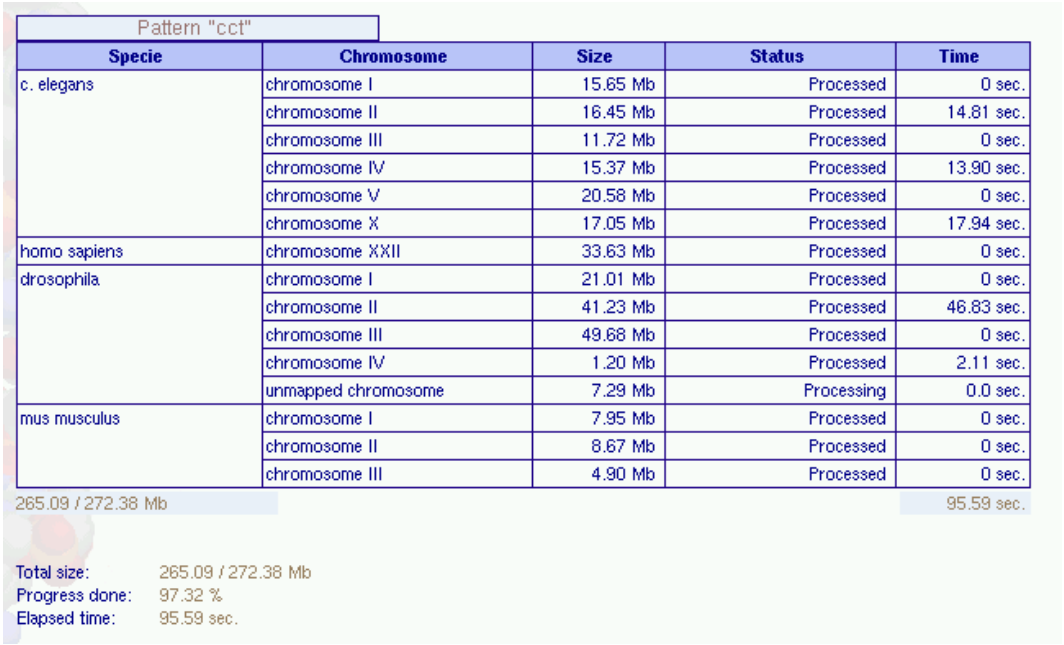
Se le muestran dos posibles alternativas:

- "View Results": Monitoriza el estado de sus petición. Si sigue este link podrá comprobar el estado de cada uno de los cromosomas que se están computando. Cuando finalicen todos, automáticamente se le aparecerán los resultados. Esta opción no excluye el envío del correo que recibirá cuando finalicen los resultados. Por tanto, si una vez que entre en la monitorización, cierra la ventana, podrá saber que su petición ha finalizado cuando reciba el mensaje por e-mail.

- "Finish": Cierra la ventana actual. Cuando su petición haya finalizado se le enviará un correo con un link hacia los resultados. Dado que todos los resultados pedidos por los usuarios se guardan en una base de datos, si no introdujo su e-mail tendrá que repetir periódicamente su petición en la página principal para saber si los cálculos han finalizado y ver los resultados.

Si su petición no es mayor de 300 Mb, le sugerimos que siga la monitorización para obtener rápidamente los resultados.

7.10. Lista de cromosomas que se están procesando



Pattern "cct"

Specie	Chromosome	Size	Status	Time
c. elegans	chromosome I	15.65 Mb	Processed	0 sec.
	chromosome II	16.45 Mb	Processed	14.81 sec.
	chromosome III	11.72 Mb	Processed	0 sec.
	chromosome IV	15.37 Mb	Processed	13.90 sec.
	chromosome V	20.58 Mb	Processed	0 sec.
	chromosome X	17.05 Mb	Processed	17.94 sec.
homo sapiens	chromosome XXII	33.63 Mb	Processed	0 sec.
drosophila	chromosome I	21.01 Mb	Processed	0 sec.
	chromosome II	41.23 Mb	Processed	46.83 sec.
	chromosome III	49.68 Mb	Processed	0 sec.
	chromosome IV	1.20 Mb	Processed	2.11 sec.
	unmapped chromosome	7.29 Mb	Processing	0.0 sec.
mus musculus	chromosome I	7.95 Mb	Processed	0 sec.
	chromosome II	8.67 Mb	Processed	0 sec.
	chromosome III	4.90 Mb	Processed	0 sec.
265.09 / 272.38 Mb				95.59 sec.

Total size: 265.09 / 272.38 Mb
Progress done: 97.32 %
Elapsed time: 95.59 sec.

Figura 7.9: Pantalla de monitorización del cómputo

Esta página le informa del estado de su petición.

Esta página se irá refrescando automáticamente en intervalos de 5 segundos. Cuando todos los cromosomas hayan finalizado se le aparecerán los resultados.

Esta página está compuesta de tablas. Cada tabla hace referencia al cómputo de los cromosomas en busca de un patrón, el que se indica en la cabecera de la tabla. En nuestro ejemplo, la única tabla que aparece hace referencia a la búsqueda del patrón "CCT" (y de su complementario "AGG"). Los elementos que aparecen en la tabla son:

- "Specie": La especie agrupa varias filas. Cada fila agrupada por la especie pertenece a cada uno de sus cromosoma que se debe procesar.
- "Chromosome": Muestra el identificador del cromosoma.
- "Size": Tamaño en Mb. del cromosoma.
- "Status": Estado actual del cromosoma. Los estados puede ser tres:
  - "Waiting": El cromosoma esta a la espera de ser procesado.
  - "Processed": El cromosoma ya ha sido procesado.
  - "Processing": El cromosoma se esta procesando en estos momentos.
- Time: El tiempo en segundos que ha tardado el cromosoma en ser procesado. Si el estado del cromosoma es "Wait" o "Processing" siempre será 0. Si el estado del cromosoma es "Processed" y el valor de "Time" es 0, entonces eso indica que mientras se estaba procesando su petición, otro usuario también a pedido la misma búsqueda para ese cromosoma, y la petición de ese usuario se adelantó a la suya. Cuando su petición ha empezado a procesar el cromosoma ha comprobado que ya estaba finalizado y el tiempo de esa comprobación a sido de 0 segundos.

En el pie de cada tabla aparecen la suma de tamaños procesados y el tiempo que se ha dedicado para ello.

Los tres campos que se encuentran en la parte inferior de la página hacen referencia a:

- "Total size": Muestra el tamaño total calculado / tamaño total a calcular.
- "Progress done": El porcentaje de trabajo realizado.
- "Elapsed time": Tiempo en segundos transcurrido desde que empezó el proceso.

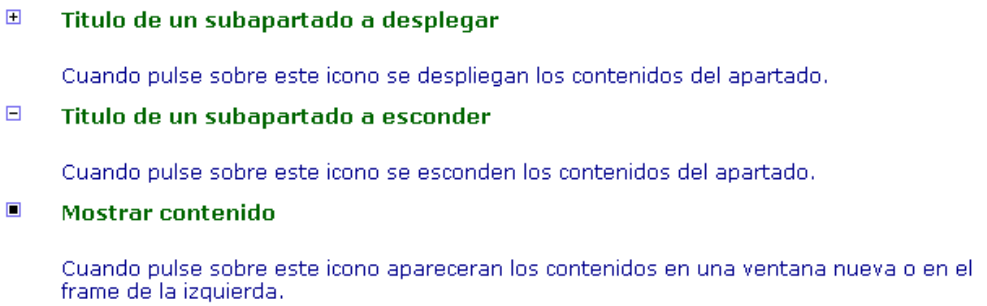
7.11. Índice de resultados

Las acciones que se pueden realizar en el índice son :

- Navegar por el índice.
- Cargar una sección en el frame de la izquierda.
- Abrir una nueva ventana con la información de una sección.

Existen cuatro tipos de iconos a la izquierda de cada apartado, que indican si el título a la derecha es un apartado o un contenido:

Figura 7.10: Pantalla del índice de resultados



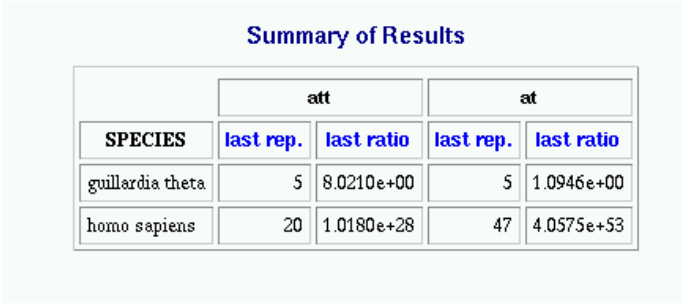
7.11.1. Mostrar los contenidos en el frame de la izquierda

Para mostrar los contenidos en el frame pulse el título del contenido que quiera ver.

7.11.2. Mostrar los contenidos en una ventana nueva

Para mostrar los contenidos en el frame pulse directamente el icono *"Mostrar contenido"*, y NO sobre el título del contenido.

7.12. Resumen de Resultados



Summary of Results

	att		at	
SPECIES	last rep.	last ratio	last rep.	last ratio
guillardia theta	5	8.0210e+00	5	1.0946e+00
homo sapiens	20	1.0180e+28	47	4.0575e+53

Figura 7.11: Pantalla de la página de resumen de resultados

En esta página puede verse una tabla con el resumen de resultados para cada grupo escogido.

Las columnas clasifican los resultados por cada patrón, mientras que las filas indican el grupo al que pertenecen los resultados.

Dentro de cada celda "patrón/grupo" aparecen dos resultados:

- *last rep.*: Indica la repetición máxima encontrada del patrón en es grupo. Así por ejemplo, en la figura adjunta con esta ayuda, se ha encontrado en el grupo "guillardia theta", un máximo de 5 repeticiones consecutivas para el patrón "att", es decir, se ha encontrado una secuencia "attattattattatt" (o "aataataataataat") y esa es la secuencia máxima de repeticiones de "att" en el grupo "guillardia theta".

  El número de veces que se ha encontrado una ocurrencia de repetición máxima no se especifica en esta tabla, sino que se especifica en la "tabla de resultados" de cada patrón.

  Por tanto, en nuestro ejemplo, sabemos que hay almenos una ocurrencia del patrón "att" repetido consecutivamente 5 veces, no obstante no se indica cuantas ocurrencias con esta repetición se han encontrado.

- *last ratio*: Indica el valor del ratio real-teórico para las ocurrencias con repetición máxima encontrada. Para saber mas sobre el ratio y cómo se calcula lea la sección 7.13.

7.13. Cálculo realizados

7.13.1. Cálculo de las ocurrencias teóricas

El cálculo se efectúa para cada repetición de un patrón encontrada. La fórmula que se utiliza es la siguiente:

$$no_k = (\mathbb{P}[p^k] + \mathbb{P}[p_c^k]) * (|\mathcal{C}| - k * |p| + 1)$$

dónde

no_k	Es el número de ocurrencias teórico para un patrón repetido k veces consecutivas.
$\mathbb{P}[p^k]$	Probabilidad de aparición de el patrón introducido por el usuario, repetido consecutivamente k veces.
$\mathbb{P}[p_c^k]$	Probabilidad de aparición de el patrón complementario, repetido consecutivamente k veces.
$\|\mathcal{C}\|$	Es el número de bases del cromosoma.
$\|p\|$	Es la longitud del patrón.

Dado que buscamos tanto el patrón introducido por el usuario y el patrón complementario, sumamos la probabilidad de aparición de ambos. El resultado del número de ocurrencias es el producto resultante de la suma de estas dos probabilidades por la longitud total del cromosoma (a esta longitud le restamos la longitud del patrón y le sumamos uno).

Por ejemplo, si buscamos el número de ocurrencias de orden 3 (3 repeticiones consecutivas del patrón), la longitud del cromosoma es de 10000 bases, la longitud del patrón es

2, la probabilidad de aparición del patrón de orden 3 es 0.000216 y la del complementario es 0.000343, tenemos:

$$no_k = (0{,}000216 + 0{,}000343) * (10000 - 2 * 3 + 1) = 0{,}000559 * 9995 = 5{,}5872$$

En el siguiente apartado se explica como se realiza el cálculo de la probabilidad de un patrón. Ambos patrones (el complementario y el introducido) se calculan de la misma forma.

7.13.2. Cálculo de la probabilidad de un patrón

Para realizar el cálculo de las ocurrencias teóricas necesitamos las dos tablas explicadas en la ayuda sobre la distribución. Pulse ¡A HREF="javascript:help()"¿aquí¡/A¿para ver la página de ayuda sobre la distribución, dónde aparecen las tablas utilizadas en los ejemplos que a continuación se explicarán.

La fórmula que se aplica es la siguiente:

$$\mathbb{P}[p^k] = \mathbb{P}[p(1)] * \prod_{i=1}^{k*|p|-1} (\mathbb{P}[p(i+1)|p(i)])$$

dónde

$\mathbb{P}[p(1)]$ Es la probabilidad de aparición del primer carácter del patrón.
Dicha probabilidad la obtenemos de la tabla de porcentajes de cada base. Por ejemplo, supongamos la tabla de la página de ayuda sobre la distribución, con el grupo "guillardia theta" y el patron "att", entonces
$\mathbb{P}[p(1)] = 0{,}3679(\mathbb{P}[a] = 0{,}3679)$.

$\mathbb{P}[p(i+1)|p(i)]$ Probabilidad condicional de aparición de un carácter del patrón en el cromosoma en un patrón repetido k veces. La parte derecha de la expresión indica en que carácter estamos y la parte izquierda el carácter siguiente.
Esta probabilidad la obtenemos de la tabla de probabilidades condicionales.
La parte izquierda de la probabilidad nos indica la fila, mientras que la derecha la columna en la tabla de probabilidades condicionales.
Por ejemplo, supongamos la tabla de probabilidades de la página de ayuda sobre la distribución, el grupo "guillardia theta", y la segunda repetición del patrón "at", es decir, "atat".
Si buscamos $\mathbb{P}[p(3)|p(2)]tenemos\mathbb{P}[a|t] = 0{,}2787$.

7.13.3. Cálculo del ratio "real-teórico"

El ratio real-teórico se calcula a partir de las ocurrencias reales y las teóricas para cada repetición del patrón encontrada en el grupo.

Para obtener el ratio calculamos la división entre las ocurrencias reales y las teóricas. Por ejemplo, supongamos que se han encontrado 10 ocurrencias del patrón "att" repetido 4 veces ("attattattatt") y 20 de su complementario ("aataataataat"). El número de ocurrencias teóricas para ambos patrones repetidos 4 veces es de 0,8. Entonces el ratio es $\frac{30}{5} = 37{,}5$. La notación que se hace servir para el ratio es científica, en este caso el resultado seria en notación científica: $3{,}75e + 01$. Es decir, $3{,}75 * 10^1$.

7.14. Distribución y número de bases

En esta página pueden verse dos tablas relacionadas con la distribución de las bases en los cromosomas analizados, y con la probabilidad condicionada. Ambas tablas son importantes y se usan para calcular el número de ocurrencias teóricas.

7.14.1. Tabla de distribución y número de bases

Distribution and number of bases

	guillardia theta		homo sapiens	
a	36.79%	138740 bp.	27.94%	19207491 bp.
c	13.16%	49639 bp.	22.15%	15229034 bp.
g	13.05%	49214 bp.	22.15%	15225586 bp.
t	13.16%	49639 bp.	22.15%	15229034 bp.
total	100%	377131 bp.	100%	68739104 bp.

Figura 7.12: Pantalla de la página de la distribución de las bases

En esta tabla aparecen el número total de ocurrencias encontradas de cada base A,C,G,T en los diferentes grupos, y sus porcentajes. En la figura adjunta vemos, por ejemplo, que el porcentaje de "C","G" y "T" es prácticamente el mismo en el grupo "guillardia Theta".

7.14.2. Tabla de probabilidad condicionada

Conditional Probability

	guillardia theta				homo sapiens			
	a	c	g	t	a	c	g	t
a	0.4417	0.1001	0.1209	0.3373	0.3143	0.1832	0.2593	0.2432
c	0.3677	0.1790	0.1157	0.3376	0.3404	0.2743	0.0605	0.3248
g	0.4120	0.1323	0.1719	0.2838	0.2728	0.2224	0.2747	0.2301
t	0.2787	0.1461	0.1310	0.4442	0.2012	0.2164	0.2701	0.3122

Figura 7.13: Pantalla de la tabla de probabilidades condicionadas

Esta es la tabla de transiciones de las cadenas de markov de primer orden de cada grupo.

La interpretación es la siguiente: Si nos encontramos en una base del cromosoma b_t, ¿cual s la probabilidad de que la siguiente base sea la b_{t+1}s? En nuestro ejemplo, para la *guillardia theta*, si nos encuentro en la base "a" la probabilidad de que la siguiente base sea una "c" es de 0,1001.

Las filas indican la posición actual, y las columnas la probabilidad de aparición de una determinada base justo después de la base de la fila donde nos encontramos.

7.15. Tabla de resultados

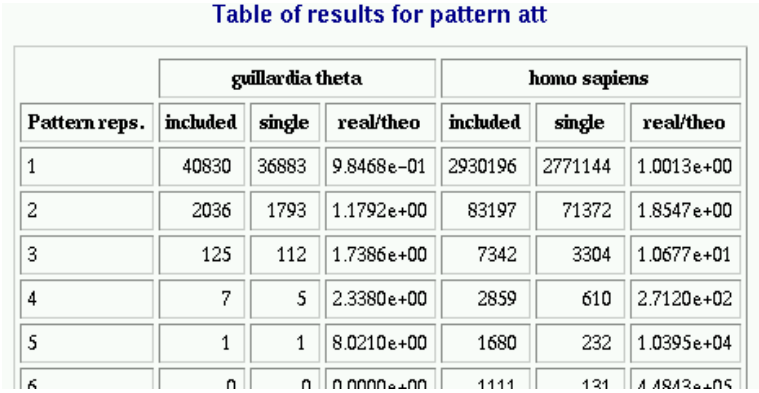


Table of results for pattern att

	guillardia theta			homo sapiens		
Pattern reps.	included	single	real/theo	included	single	real/theo
1	40830	36883	9.8468e-01	2930196	2771144	1.0013e+00
2	2036	1793	1.1792e+00	83197	71372	1.8547e+00
3	125	112	1.7386e+00	7342	3304	1.0677e+01
4	7	5	2.3380e+00	2859	610	2.7120e+02
5	1	1	8.0210e+00	1680	232	1.0395e+04
6	0	0	0.0000e+00	1111	131	4.4843e+05

Figura 7.14: Pantalla de la página de resultados

En esta página puede verse una tabla con los resultados de cada repetición del patrón encontrados para cada grupo.

Las columnas principales corresponden a los grupos que se han pedido analizar, mientras que las filas corresponden a la repetición i-ésima del patrón. Así la fila con índice 1 indica los resultados para el patrón (el introducido o su complementario) repetido 1 vez, así sucesivamente.

Dentro de cada celda "repetición/grupo" aparecen dos resultados:

- *included*: Indica el número de ocurrencias incluidas (o sobrepuestas, o acumuladas) que se han encontrado del patrón repetido el numero de veces que indica el índice de la fila. Por ejemplo, en la imagen adjunta, podemos ver que para la "guillardia theta" y el patrón "att" se han encontrado 40830 ocurrencias (estas ocurrencias pertenecen a la suma de las encontradas para "att" y su complementario "aat"), el patrón "attatt" 2036, etc...

  En nuestro ejemplo también podemos ver que el número de repeticiones máximas del patrón "att" y su complementario "aat" es de 5, habiendo encontrado sólo 1 ocurrencia para esta repetición.

Si el patrón complementario y el introducido por el usuario son el mismo, o una permutación circular (por ejemplo: "ata", cuyo complementario es "tat", una permutación circular), el número de ocurrencias *included* que aparecen son el doble de las que realmente se han encontrado.

- *single*: Indica el número de ocurrencias únicas que se han encontrado del patrón repetido el número de veces que indica el índice de la fila.

  Todo lo que se ha explicado para las ocurrencias *included* es aplicable a las ocurrencias *single*. En el apartado siguiente de esta ayuda se explica la diferencia entre unas y otras.

- *real/theo*: Indica el ratio real-teórico para cada repetición. Para saber mas sobre el ratio y cómo se calcula lea la sección 7.13.

7.15.1. ocurrencias únicas y acumuladas

Cuando encontramos una ocurrencia del patrón k veces repetido, con k mayor de uno, aparecen en esa misma ocurrencia otras de repetición menor a k.

Por ejemplo, si estamos buscando el patrón "at", y nos encontramos ante una ocurrencia como "atatat" tenemos una ocurrencia de orden 3, 2 de orden 2 y 3 de orden 1.

Las ocurrencias únicas son aquellas que no están incluidas en una ocurrencia de orden superior, mientras que las acumuladas sí que pueden estarlo.

Por ejemplo, si el cromosoma fuera "aattcgcgtaaa", y el patrón introducido fuera "a", el número de ocurrencias acumuladas para cada repetición sería el siguiente:

- Repetición 1 ("a"+"t"): 8
- Repetición 2 ("aa" + "tt"): 4
- Repetición 3 ("aaa" + "ttt"): 1

Mientras que el número de ocurrencias únicas seria:

- Repetición 1 ("a"+"t"): 1
- Repetición 2 ("aa" + "tt"): 2
- Repetición 3 ("aaa" + "ttt"): 1

7.16. Página principal de Zoom

En esta página se muestra imagen que representa las ocurrencias de todas las repeticiones encontradas en un cromosoma.

Usted podrá realizar "zooms" de las regiones que crea oportunas. La imagen siempre tendrá las mismas dimensiones y los zoom variarán la longitud de la región del cromosoma que se representa. La longitud mínima que se representa es de 100 bases.

Hay dos vías para hacer "zooms" de regiones:

- Manual, introduciendo los valores absolutos o porcentajes.
- Pinchar sobre la región.

El zoom aparecerá en una ventana nueva.

A continuación se detallan estas dos vías, y en la última sección se explica el tiempo de creación que necesita una imagen.

Para saber mas sobre la interpretación de una imagen lea la sección 7.17.

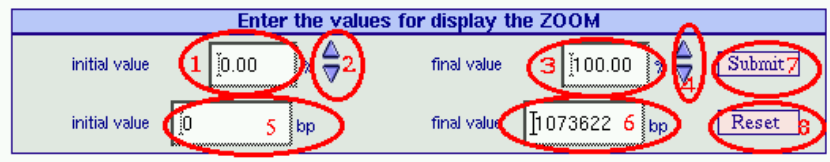


Figura 7.15: Pantalla de la página de zoom

7.16.1. Abrir un zoom introduciendo los porcentajes

Los pasos a seguir para definir una región a través de los porcentajes son los siguientes:

1. Introduzca el porcentaje inicial. Para hacerlo puede introducirlo directamente en el texto del *componente 1* o pulsando los botones de incremento/decremento del *componente 2*. Si los introduce a través del texto del *componente 1*, después de pulsar la tecla `<TAB>` verá que se modifican los valores absolutos del *componente 5*.

2. Introduzca el porcentaje final. La forma de hacerlo es análoga a la explicada en el paso 1 para el porcentaje inicial. Puede introducir el porcentaje en el texto del *componente 3* o bien modificar el valor con los botones de incremento/decremento del *componente 4*. Si los introduce a través del texto, después de pulsar la tecla `<TAB>` se modificará el valor absoluto del *componente 6*.

3. Pulsar el botón "Submit" del *componente 5* y se abrirá una nueva ventana con la imagen de la región especificada.

Si en algún momento quiere volver a empezar con los valores iniciales, pulse el botón "reset" del *componente 8*.

7.16.2. Abrir un zoom introduciendo los valores absolutos de las posiciones

Los pasos a seguir para definir una región a través de los valores absolutos de las posiciones son los siguientes:

1. Introduzca la posición inicial en el texto del *componente 5*, sabiendo que la primera posición es el 0. Después de pulsar la tecla "TAB" verá que se modifica el porcentaje del *componente 1*.

2. Introduzca la posición final en el texto del *componente 6*. Después de pulsar la tecla "TAB" se modificará el porcentaje del *componente 3*.

3. Pulsar el botón "Submit" del *componente 5* y se abrirá una nueva ventana con la imagen de la región especificada.

Si en algún momento quiere volver a empezar con los valores iniciales, pulse el botón "reset" del *componente 8*.

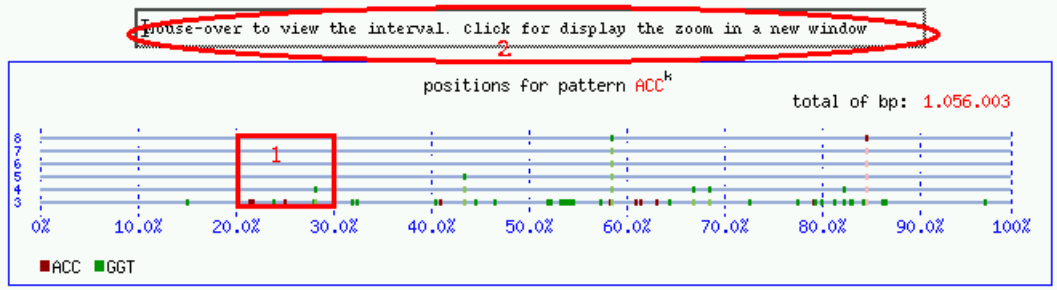


Figura 7.16: Pantalla de un mapa generado por el módulo de zoom

7.16.3. Ver los intervalos de cada región

Como puede observar, la imagen esta dividida en 10 zonas de igual longitud.

Cada una de estas zonas delimita una región que usted puede pinchar para abrir un zoom.

Cuando pasamos encima, con el ratón, de una región (por ejemplo la región del *componente 1*) podemos ver la posesión inicial y final del intervalo en el texto del *componente 2*.

7.16.4. Abrir un zoom a través de la imagen

Como ya se ha dicho, la imagen esta dividida en 10 zonas de igual longitud.

Cada una de estas zonas delimita una región que usted puede pinchar para abrir un zoom.

Cuando vea una región que le resulte interesante, únicamente pinche encima de ella. Entonces se abrirá una nueva ventana con las posiciones iniciales y finales de la región seleccionada.

7.16.5. Tiempo de carga de las imágenes

Cada vez que realiza un "zoom", la imagen se crea "on line" (sopena que quiera una imagen que pidió anteriormente).

La creación de dicha imagen tiene un coste temporal en función del tamaño del intervalo pedido.

Un tiempo orientativo suele ser de 1,5 Mb/segundo, es decir de unas 1,500,000 bases/segundo, no obstante puede fluctuar en función de la carga del servidor o de internet.

Se recomiendo para evitar esperas, que los zooms no excedan de 2 Mb. (unas 2,000,000 bases), y una vez abierta la ventana de ese zoom navegar por la imagen. Si quisiera un zoom con un tamaño de ese orden tendría que esperar varios segundos a que este apareciera (habitualmente entre 2 y 3 segundos mas el tiempo de cargar la página).

7.17. Interpretación de la imagen generada por Zoom

En función del tamaño de la región que represente la imagen, podremos ver un tipo de esquema u otro.

Si el tamaño es mayor de 100 bases nos encontramos ante una imagen escalable, mientras que si llegamos a 100 bases, la imagen es no escalable puesto que ya no se pueden aplicar mas aumentos.

En los apartados siguientes se detalla las diferentes informaciones que podemos encontrar entre estos dos tipos de imágenes.

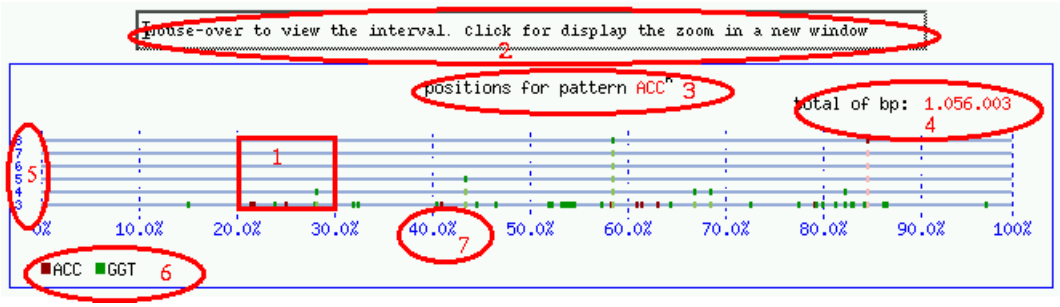


Figura 7.17: Pantalla de un mapa escalable generado por zoom

7.17.1. Información común entre una imagen escalable y no escalable

Los siguientes componentes son comunes a los dos tipos de imagen:

- *Componente 3* Título: Indica el patrón que se está buscado la región. En el caso de nuestro ejemplo, el "ACC" (y su complementario "GGT").

- *Componente 4* Número de bases: Indica el número de bases encontradas en la región. Dado que en un fichero "FASTA" podemos encontrar otros caracteres aparte de las 4 bases A,C,G,T (por ejemplo los caracteres de la etiqueta, o "N" para bases desconocidas), el tamaño de la región puede ser mayor que el número de bases.

- *Componente 5* Índice de las repeticiones: Cada línea azul horizontal esta asociada a un índice de la repetición. Dicho índice indica el número de repeticiones consecutivas del patrón. En nuestro ejemplo podemos encontrar el patrón repetido 8 veces en el intervalo que va del 50 % al 60 %.

- *Componente 6* Leyenda: La leyenda indica los colores que se han utilizado para representar las ocurrencias del patrón y su complementario.

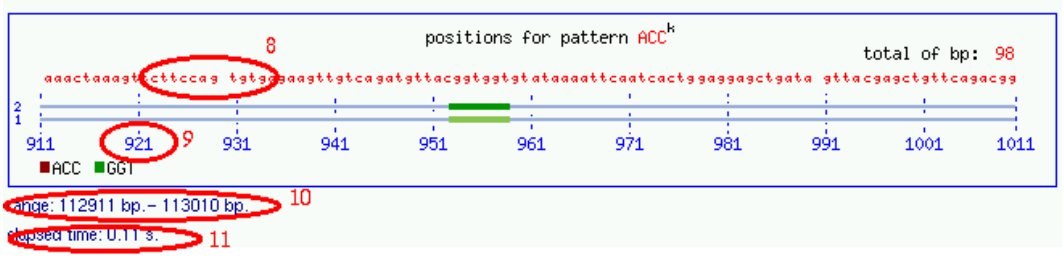


Figura 7.18: Pantalla de un mapa no escalable generado por zoom

7.17.2. Información de una imagen no escalable

En la imagen no escalable, la longitud de la región será siempre de 100 caracteres, aunque como se ha dicho, entre estos 100 caracteres pueden aparecer algunos que no sean las bases "A,C,G,T". A continuación se detallan los componentes propios de este tipo de imagen

- *Componente 8* Caracteres: En esta parte se escriben los caracteres del cromosoma que forman parte de esta región. Aparte de las cuatro bases pueden aparecer los siguiente:

  - N: Base desconocida.
  - X: Carácter perteneciente a la etiqueta "FASTA"
  - " "(Carácter en blanco): salto de línea.

- *Componente 9* Parte de las centenas de la posición: Muestra las últimas tres cifras de la posición absoluta. Para saber cual es la posición inicial hay que observar el texto del *componente 10*.

- *Componente 10* Rango del intervalo: Indica la posición inicial y final de la región mostrada. Las unidades son caracteres, es decir, el tamaño de la región (posición final - posición inicial + 1) es el número de caracteres que hay en la región (100), que puede ser superior a las bases encontradas.
- *Componente 11* Tiempo en crear la imagen: Indica el tiempo en segundos, con dos decimales, que ha tardado en crear la imagen. Este tiempo puede ser indicativo para conocer de la velocidad del servidor. En nuestro ejemplo la velocidad del servidor es de unas 900 bases/segundo.

Bibliografía

[AC75] A. V. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18(6):333–340, 1975.

[ACR99] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: A new structure for pattern matching. In *Conference on Current Trends in Theory and Practice of Informatics*, pages 295–310, 1999.

[BM77] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. ACM*, 20(10):762–772, 1977.

[BYG92] R. Baeza-Yates and G. Gonnet. A new approach to text searching. *Comm. ACM*, 35(10):74–82, 1992.

[CCG+94] A. Czumaj, M. Crochemore, L. Gasieniec, S. Jarominek, W. Plandowski T. Lecroq, and W. Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12:247–267, 1994.

[CCG+99] Maxime Crochemore, Artur Czumaj, Leszek Gasieniec, Thierry Lecroq, Wojciech Plandowski, and Wojciech Rytter. Fast practical multi-pattern matching. *Information Processing Letters*, 71(3-4):107–113, 1999.

[Cow91] R. Cowan. Expected frequencies of dna patterns using whittle's formula. *J.Appl.Prob.*, 28:886–892, 1991.

[CW79] B. Commentz-Walter. A string matching algorithm fast on the average. In *6th*, number 71 in Lecture Notes in Computer Science, pages 118–132. H.A. Maurer, 1979.

[CW94] M. Crochemore and W.Rytter. *Text Algorithms*. Oxford University Press, 1994.

[Hle97] O. Hlenberg. *Foundations of Modern Probability*. New York: Springer-Verlag, 1997.

[Hor80] R. Horspool. Practical fast searching in strings. *Softw. Pract. Exp.*, 10(6):501–506, 1980.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J.Comput*, 6(2):323–350, 1977.

[KS76] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. New York: Springer-Verlag, 1976.

[RSM00] G. Reinert, S. Schbath, and M.Waterman. Probabilistic and statistical properties of words: An overview. *Journal of Computational Biology*, 7(1-2):1–46, 2000.

[Ste95] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. NJ: Princeton University Press, 1995.

[vR98] G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, number 1448 in Lecture Notes in Computer Science, pages 14–33. Springer-Verlag, Berlin, 1998.

[WM94] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, Chung-Cheng University, 1994.

Índice alfabético