Búsqueda de Genes utilizando modelos de Markov ocultos

Estudiantes: Diego Espinosa

Pablo Gallo

Federico Sotto

Tutor: Carlos Testuri

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación.

Facultad de Ingeniería, Universidad de la República.

Montevideo, 2005

RESUMEN

Actualmente, la secuenciación de genomas de distintas especies plantea desafíos en el análisis e interpretación de volúmenes importantes de información sobre la composición de los mismos. Uno de los problemas es resolver qué regiones, dentro del genoma, constituyen genes (región de ADN que controla una característica hereditaria de un organismo). Una metodología es la descodificación de la información genética mediante la aplicación de modelos ocultos de Markov, a la búsqueda de genes. Los modelos ocultos de Markov permiten describir procesos con señales observables, modelando propiedades estadísticas estables de las mismas; permitiendo implementar sistemas de reconocimiento y predicción de las observaciones.

El presente trabajo consiste en el desarrollo de una aplicación de búsqueda de secuencias de nucleótidos candidatas a clasificar como gen; la misma es extensible y consta de (a) un núcleo central que implementa los problemas de evaluación de secuencias y entrenamiento de modelos ocultos de Markov, el cual incorpora un esquema de utilización de memoria flexible, y (b) una interfaz gráfica a dichas implementaciones que específicamente resuelve el problema de identificar genes contenidos en un genoma. Los resultados prácticos obtenidos no son los que inicialmente se esperaban; no obstante, para los casos estudiados, se identificaron correctamente aproximadamente la mitad de los genes.

TABLA DE CONTENIDO

1 Introducción

El objetivo de este documento es presentar una visión general del trabajo que se ha desarrollado en el marco del Proyecto de Grado. También se abordarán aquellas cuestiones específicas que han sido significativas a lo largo del mismo. En este capítulo se presenta una visión general del proyecto y del tema investigado. Se comenzará realizando una descripción del problema puntual que se desea resolver. Luego se describirá las motivaciones que nos llevaron a la realización del proyecto. Se enumerarán los objetivos planteados y los resultados esperados, explicando el enfoque aplicado para alcanzarlos. Por último se presenta un resumen de las conclusiones del proyecto a las que hemos llegado.

1.1 Descripción del problema

Actualmente el área de la biología molecular está teniendo un crecimiento explosivo debido en gran parte al aumento del poder de cálculo de las computadoras. Desde mediados de la década de 1990, ha surgido con mucha fuerza el término bioinformática, que no es otra cosa que la aplicación de la informática como herramienta para la investigación en el área de la biología, y en particular de la biología molecular.

Los científicos de hoy en día cuentan con una gran cantidad de información que aun no ha sido procesada. A modo de ejemplo, se conoce la secuencia de bases de las moléculas de ADN (ácido desoxirribonucleico, material genético de todos los organismos celulares y casi todos los virus, por mas información ver capítulo 2) de muchas especies, sin embargo, se sabe muy poco acerca de los genes contenidos en esas moléculas de ADN, en proporción a la cantidad de genes que existe. No sólo no se conoce la función de los mismos, sino que en general tampoco se está seguro de si una porción del ADN es verdaderamente un gen. Es éste el problema que se procurará resolver: dada una secuencia de ADN, descubrir los potenciales genes que se encuentran contenidos en la misma, utilizando modelos de Markov ocultos. Una vez que los genes son identificados, entonces es posible realizar distintos estudios sobre los mismos, como ser su función o sus propiedades.

Muchos de los conceptos que manejamos quizá no sean conocidos por el lector, es por ello que hemos dedicado todo el capítulo 2 de este documento a explicar con mayor profundidad los términos relacionados al dominio del problema. Por otra parte, el capítulo 3 presentará una breve reseña acerca de los modelos de Markov ocultos.

1.2 Motivaciones

Si bien hoy en día existen varias aplicaciones que ya resuelven este problema, creemos que el realizar este trabajo nos brinda la posibilidad de conocer con mayor profundidad el área de la bioinformática. Por otra parte, en Uruguay la bioinformática no es un área que esté recibiendo mucha atención, por lo que este proyecto, si bien no realiza aportes nuevos en lo que respecta a la bioinformática, si puede sentar un precedente que sirva de base para futuros desarrollos en esta área, aquí en nuestro país.

El hecho de poder vincular dos áreas tan disímiles como son la computación y la biología, fue algo que suscitó nuestro interés por este proyecto. Por otra parte, hasta el momento no habíamos participado en ningún proyecto interdisciplinario, por lo que encontramos en éste, no sólo una oportunidad, sino también un desafío.

1.3 Objetivos y Resultados Esperados

La siguiente lista presenta los objetivos a alcanzar en este proyecto:

- Implementación de una aplicación capaz de encontrar genes en una secuencia de ADN utilizando modelos de Markov, procurando obtener un alto porcentaje de aciertos (aproximadamente un 80%), y bajos porcentajes de falsos positivos y falsos negativos. Dichos modelos deben poder ser definidos por los expertos en el área de la bioinformática y biología molecular a los efectos de ajustar el comportamiento de la aplicación de acuerdo al organismo que se quiere estudiar.

- Evaluar la aplicabilidad de los modelos de Markov ocultos (HMMs) para la búsqueda de genes. Los resultados de dicha evaluación se presentan en el capítulo 6.

- Dividir en dos grandes módulos la implementación de la aplicación. Por un lado, un motor que resuelva los problemas de entrenamiento y evaluación de un HMM, el cual pueda ser reutilizado, no sólo en los problemas relacionados al área de la biología molecular, sino también en problemas relacionados a otras disciplinas. Por otro lado, la aplicación propiamente, que, utilizando el motor de HMMs, realice la búsqueda de genes.

- Establecer una primera instancia de relacionamiento con la Facultad de Ciencias en lo que respecta a la bioinformática, dejando abierta la posibilidad de nuevos desarrollos en el área.

1.4 Conclusiones

La siguiente lista presenta un resumen de las conclusiones ha las que hemos arribado al finalizar el proyecto:

- Los resultados que se obtienen con la aplicación finalmente desarrollada no son los que nos propusimos obtener al inicio del proyecto. Si bien se logró encontrar alrededor de un 50% de genes; el porcentaje de falsos positivos es muy alto.

- La forma en la que aplicamos los modelos ocultos de Markov para resolver el problema de la búsqueda de genes, creemos no es la mejor. A su vez, del estado del arte de los HMMs aplicados a este problema, podemos ver que en general son utilizados como parte de una solución más compleja y no como la solución final

- Se logro desarrollar una aplicación dividida en dos grandes módulos: por un lado un motor que resuelve los problemas de entrenamiento y evaluación de los HMMs (independiente del dominio) y por otro lado, un módulo que se encarga de la lógica referente al dominio y de la interfaz de usuario (UI).

- Se lograron implementar los requerimientos funcionales y no funcionales más importantes.

- Experiencia positiva en cuanto al relacionamiento con la Facultad de Ciencias de la Universidad de la República, mas específicamente con los docentes Héctor Musto y Héctor Romero, quienes trabajan en las áreas de Bioinformática y Biología Molecular, sin cuya ayuda hubiera resultado imposible desarrollar la aplicación.

1.5 Organización del resto del documento

El resto de los capítulos detallan el proceso de software de acuerdo a las diferentes etapas del desarrollo del proyecto:

- En el capítulo 2 se describe en detalle el dominio de la aplicación, dándose la definición de los conceptos biológicos que se utilizaron a lo largo del proyecto.
- El capítulo 3 presenta una introducción a los modelos de Markov ocultos. Hace hincapié en su definición y en los principales problemas relacionados con los mismos, a saber el problema de la evaluación y el problema del entrenamiento.
- En el capítulo 4 se describe el problema y la solución que se implementó. En particular, se describe como se aplicaron los HMMs para resolver el

problema de la identificación de genes. A su vez, se describen trabajos relacionados con el enfoque adoptado por nuestro grupo.

- En el capítulo 5 se describe la implementación de la aplicación realizada. Se describen los requerimientos (funcionales y no funcionales), la arquitectura, las decisiones de diseño y la solución implementada. A su vez, se describe de que manera se efectuó el análisis de los resultados.
- En el capítulo 6 se describen los resultados obtenidos, mostrando para algunos modelos utilizados, la efectividad y eficiencia de la herramienta construida.
- En el capítulo 7 se describen las conclusiones de nuestro trabajo. Las mismas se dividen en: conclusiones referentes a los objetivos del proyecto, conclusiones referentes al cronograma y conclusiones respecto a las funcionalidades del producto.
- El capítulo 8 consiste en un Anexo referente a los modelos ocultos de Markov, en el cual se describe con mayor detalle los problemas principales relacionados con los mismos.
- El capítulo 9 es un Anexo que describe la manera en que se desarrolló el trabajo. A modo de cronograma, se presentan los objetivos para cada período, los problemas que se encontraron y la manera en que se resolvieron así como también las decisiones que se tomaron.
- El capítulo 10 presenta las referencias a los documentos utilizados a lo largo del proyecto.

2 Conceptos del dominio

Como ya se mencionó en el capítulo anterior, el sistema que se desea construir tiene como objetivo descubrir genes en una molécula de ADN en organismos procariotas [Watson]. Para poder tener una mejor comprensión de éstos temas, presentaremos una breve introducción de los conceptos e ideas que manejaremos a lo largo de este documento, en base a los conocimientos adquiridos en el curso que nos dictaron los docentes de la Facultad de Ciencias [Musto].

2.1 El ADN

El ADN (ácido desoxirribonucleico) es una molécula que se encuentra en las células y que contiene la información genética necesaria para la organización y funcionamiento de los seres vivos. Básicamente la molécula de ADN está constituida por dos largas cadenas de nucleótidos sostenidas en un esqueleto de azúcares, y al mismo tiempo unidas entre sí, formando una doble hélice. La unión entre las cadenas de nucleótidos se mantiene debido a que se forman enlaces entre las bases nitrogenadas que ambas contienen.

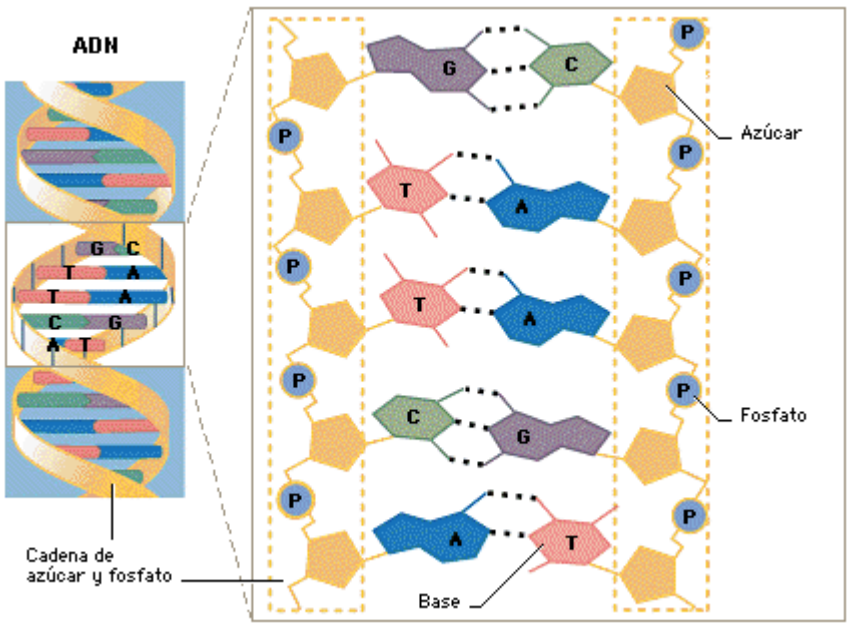


Figura 1: Esquema de parte de una molécula de ADN [Encarta]

Las bases nitrogenadas o nucleótidos que aparecen en el ADN son cuatro: adenina (A), citosina (C), guanina (G) y timina (T). En el ADN, la unión de las bases se realiza mediante puentes de hidrógeno, y este apareamiento está

condicionado químicamente de forma que la adenina sólo se puede unir con la timina y la guanina con la citosina.

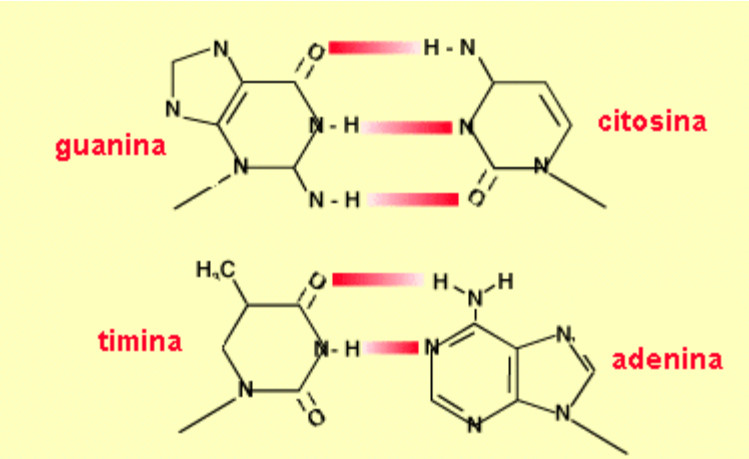


Figura 2: Estructura química y apareamientos entre las bases [EducarChile].

La estructura en doble hélice del ADN, con el apareamiento de bases limitado (A-T; G-C), implica que el orden o secuencia de bases de una de las cadenas delimita automáticamente el orden de la otra, por eso se dice que las cadenas son complementarias. Una vez conocida la secuencia de las bases de una cadena, se deduce inmediatamente la secuencia de bases de la complementaria.

La estructura de un determinado ADN está definida por la "secuencia" de las bases nitrogenadas en la cadena de nucleótidos, residiendo precisamente en esta secuencia de bases la información genética del ADN. El orden en el que aparecen las cuatro bases a lo largo de una cadena en el ADN es, por tanto, crítico para la célula, ya que este orden es el que determina las características propias de cada organismo.

La secuencia de ADN es una cadena muy larga en términos de cantidad de bases. Para tener una idea aproximada de esto, basta con saber que si la molécula de ADN de largo promedio, no se enrollara como un ovillo, sino que en realidad tuviera la forma de un hilo estirado, entonces tendría una longitud de 2 metros. Ahora bien, como la molécula se enrolla en sí misma, entonces se hace tan pequeña que es capaz de caber en el interior de una célula. El manejar cantidades tan enormes de datos (el genoma del *Escherichia coli K12*, organismo procariota sobre el cual realizamos muchas de nuestras pruebas, contiene aproximadamente tres millones de pares de bases, mientras que el genoma del ser humano tiene aproximadamente tres mil millones de pares de bases)[1] es el problema que el área de la biología molecular ha procurado resolver echando mano al poder de cálculo y procesamiento que provee la informática.

[1] Recordemos que las secuencias de ADN no sólo difieren para dos especies distintas, sino que también entre individuos de la misma especie

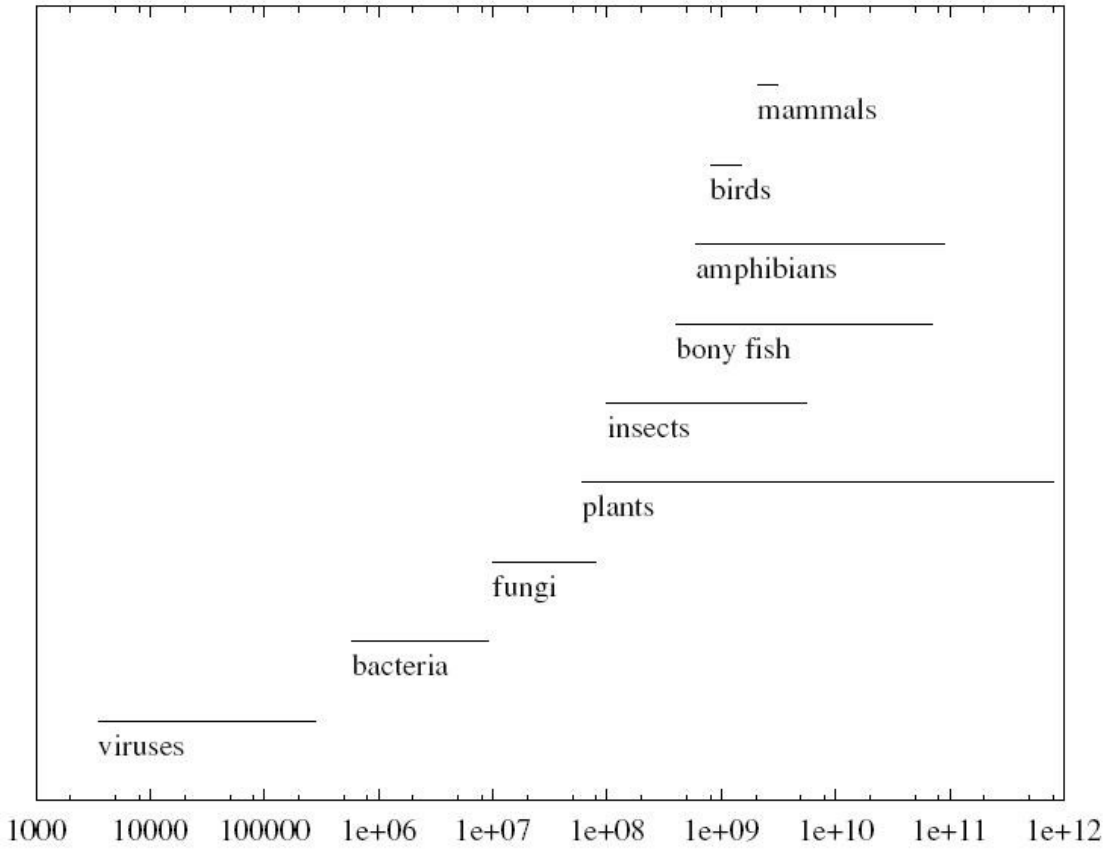


Figura 3: Intervalos de los tamaños de los genomas para varias clases de organismos [Baldi-Brunak]

En la figura 3 se presentan los intervalos de los tamaños de los genomas para varias clases de organismos. El eje X grafica la cantidad de bases nitrogenadas que posee el organismo. Como se puede apreciar, la gráfica es logarítmica respecto a este eje. Comúnmente, la variación entre un grupo y otro es mayor o igual a un orden de magnitud. El eje Y grafica la "complejidad del organismo".

2.2 Procariotas y Eucariotas

Existen dos grandes familias de seres vivos: los eucariotas y los procariotas. Los primeros son aquellos cuyas células mantienen el material genético (o sea el ADN) encerrado en el compartimiento celular denominado núcleo. Los organismos procariotas son aquellos cuyas células no contienen un núcleo. El ADN de los organismos eucariotas y procariotas, si bien en general mantienen la misma estructura, presentan peculiaridades que los diferencian. A modo de ejemplo, en general, la molécula de ADN de un organismo procariota, generalmente es "cerrada". Es decir que los extremos de la doble hélice se unen, formando una especie de lazo. En cambio, los extremos de las moléculas de ADN de los

organismos eucariotas, en general quedan "libres". Algunas diferencias más entre estos dos tipos de organismos las mencionaremos más adelante.

En el marco del proyecto utilizamos solamente genomas de seres procariotas.

2.3 Genes

La secuencia de nucleótidos del ADN, contiene en su interior ciertas regiones llamadas genes. Un gen, es una secuencia ordenada de nucleótidos que codifica un determinado producto funcional, como ser una proteína o una molécula de ARN (ácido ribonucleico)[2]. Mediante procesos de transcripción y traducción que se producen a nivel molecular, la secuencia de nucleótidos de un gen es "copiada" en moléculas de ARN y en proteínas para el uso del organismo. Así, las características del organismo quedan determinadas, en gran parte, en base a los genes que éste contiene. Los genes se encuentran en ambas hélices de la molécula de ADN.

El ADN no sólo está compuesto por genes, existen regiones de ADN que no codifican para ninguna proteína o molécula de ARN. Incluso, en los organismos basados en células eucariotas, algunas de esas regiones se encuentran en el interior de regiones codificantes (parten al gen). Esto no sucede en células procariotas. En las células eucariotas, en general las regiones codificantes del ADN son muy escasas en proporción al largo de la molécula. En cambio, en las células procariotas, la mayor parte de la secuencia de nucleótidos forma parte de algún gen.

El poder determinar exactamente los genes de un determinado organismo, y qué función cumple cada uno de ellos, es una tarea en la cual hoy en día está embarcada gran parte de la comunidad biológica mundial. El alcanzar este grado de conocimiento, abriría un gigantesco abanico de posibilidades para el tratamiento de enfermedades congénitas, para el control de epidemias, etc.

2.4 Codones

Casi la totalidad de los genes están compuestos por una cantidad de bases que es múltiplo de tres. Esto es así, debido a que el proceso de traducción (en el que se pasa de ARN a proteínas), se realiza de a conjunto de tres bases. Un codón no es otra cosa que una sucesión de tres bases nitrogenadas. Dado que se disponen de cuatro nucleótidos (A,C,T,G), entonces se tienen 64 codones posibles. Así, los genes se pueden considerar como una secuencia de codones.

[2] Material genético de ciertos virus (virus ARN) y, en los organismos celulares, molécula que dirige las etapas intermedias de la síntesis proteica (producción de proteínas). Las proteínas son cualquiera de los numerosos compuestos orgánicos constituidos por aminoácidos unidos por enlaces peptídicos que intervienen en diversas funciones vitales esenciales, como el metabolismo, la contracción muscular o la respuesta inmunológica.

Dado que los codones son ternas de bases, y que casi la totalidad de los genes están compuestos por codones (existen muy pocas excepciones, o sea genes con una cantidad de bases que no es múltiplo de 3), sucede que en cada cadena de la molécula de ADN existen tres marcos de lectura posible. Con esto queremos decir que la secuencia no debe "leerse" base por base, sino codón por codón, y naturalmente, al hacerlo así, existen tres posibles "lecturas" de la secuencia. Los genes pueden aparecer en cualquiera de los tres marcos de lectura posibles, debido a que las regiones no codificantes no necesariamente tienen una cantidad de bases múltiplo de 3. El ADN está compuesto por dos cadenas de nucleótidos complementarias (teniendo ambas genes en su interior), por lo que en una molécula de ADN, al buscarse los genes, deben considerarse en total seis marcos de lectura distintos.

Se ha observado que el primer codón de todos los genes pertenece a un subconjunto del total de codones. De hecho, dependiendo del organismo, existen a lo sumo 4 ó 5 codones que dan comienzo a un gen. También se ha observado que existen codones que siempre se encuentran al final de los genes, y también no son nunca más de 3 ó 4. Dada una secuencia de bases, todas aquellas subsecuencias que comiencen con un codón de inicio y terminen con un codón de fin son llamadas ORFs (Open Read Frame), y son potenciales genes.

3 Modelos ocultos de Markov (HMMs)

Los modelos ocultos de Markov (Hidden Markov Models, o HMMs) pueden ser utilizados para la resolución de problemas de una gran cantidad de áreas [Rabiner] [Baldi-Brunak]. En el marco de este proyecto utilizamos los mismos para la búsqueda de genes en una secuencia de ADN. Este apartado simplemente será una breve introducción a los conceptos básicos que manejan los HMMs. En el capítulo 8, desarrollamos con más detalle este tema.

Un HMM queda totalmente determinado en base a los siguientes parámetros:

- N – La cantidad de estados que tiene el modelo. Denotaremos el conjunto de estados del modelo como $S = \{S_1, S_2, \ldots, S_N\}$ y el estado en el tiempo t como q_t.

- M – La cantidad de símbolos (posibles observaciones) que maneja el modelo. Se denotará al conjunto de símbolos del modelo como $V = \{V_1, V_2, \ldots, V_M\}$. El conjunto de símbolos también recibe el nombre de alfabeto.

- A – La distribución de probabilidades de transición de un estado a otro. Denotaremos $A = \{a_{ij}\}$, donde $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i),\ 1 \le i, j \le N$.

- B – La distribución de probabilidades de observar un símbolo en un determinado estado j. Denotaremos $B = \{b_j(k)\}$, donde $b_j(k) = P(V_k \text{ en } t \mid q_t = S_j),\ 1 \le j \le N$ y $1 \le k \le M$.

- π – La distribución inicial de los estados, donde $\pi = \{\pi_i\}$ siendo $\pi_i = P(q_1 = S_i)$ para $1 \le j \le N$.

Para referirnos a las tres distribuciones de probabilidades mencionadas anteriormente, utilizaremos la siguiente notación compacta: $\lambda = (A, B, \pi)$

Son tres los problemas que deben ser resueltos de forma eficiente para poder trabajar con un HMM. El primero de ellos es el problema de evaluación que no es otra cosa que el cálculo de la probabilidad de que ocurra una secuencia observaciones O, o sea $P(O \mid \lambda)$. El segundo problema es el de alineación el cual dada una secuencia de observaciones y un modelo, busca encontrar la secuencia de estados óptima. El tercer y último problema es el "entrenamiento" del modelo, donde el objetivo es ajustar las distribuciones de probabilidades del HMM a fin de que $P(O \mid \lambda)$ sea maximizada.

Dado que en este proyecto sólo nos interesan el primer y el tercer problema, de aquí en más nos concentraremos solamente en ellos.

3.1 Problema de Evaluación

El problema de evaluación consiste en calcular $P(O \mid \lambda)$. La forma más eficiente es mediante el uso de la variable de "*forward*". Esta se define como la probabilidad de que ocurra una determinada secuencia parcial de observaciones hasta el tiempo t, y se esté en el estado S_i en el tiempo t. O sea:

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i \mid \lambda)$$

La variable $\alpha_t(i)$ se calcula recursivamente de la siguiente forma:

$$\alpha_1(i) = \pi_i b_i(O_1) \quad , \quad 1 \le i \le N$$

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) a_{ji} \right) b_i(O_{t+1}) \quad , \qquad 1 \le t \le T-1 \quad \text{y} \quad 1 \le i \le N$$

Así, para calcular $P(O \mid \lambda)$ basta simplemente con sumar los $\alpha_T(i)$ para todos los estados del modelo, o sea que:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

3.2 Problema de Entrenamiento

Actualmente no existe una forma analítica capaz de encontrar una solución global para este problema. Nosotros en particular utilizamos el procedimiento de Baum-Welch el cual permite llegar a óptimos locales. Para poder explicar este procedimiento es necesario definir previamente algunas variables:

La variable de "*backward*" se define como la probabilidad de que se produzca una determinada secuencia parcial de observaciones desde el tiempo $t+1$ hasta el final, dado el estado S_i en el tiempo t, y el modelo λ. O sea:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T \mid q_t = S_i, \lambda)$$

Al igual que la variable de *forward*, la variable de *backward* se calcula de forma inductiva:

$$\beta_T(i) = 1, \qquad 1 \le i \le N$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \qquad t = T-1, T-2, ..., 1 \quad \text{y} \quad 1 \le i \le N$$

En base a estas variables de *forward* y *backward* podemos definir la probabilidad de estar en el estado S^i en el momento t dada una secuencia de observaciones y un modelo, o sea

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

de las siguiente manera:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}$$

También definimos la probabilidad de estar en el estado S^i en el momento t y en el estado S^j en el momento $t+1$ dada una secuencia de observaciones y un modelo, o sea:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)$$

que se calcula de la siguiente manera:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$

En base a estas variables es que podemos ajustar los parámetros de un HMM. Sea $\lambda = (A, B, \pi)$, el modelo que se desea "entrenar", y $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$, el modelo "entrenado". Así, tenemos que:

$\overline{\pi}_i$ = frecuencia esperada (cantidad de veces) en el estado S_i cuando t es 1, o sea

$$\overline{\pi}_i = \gamma_t(i)$$

$\overline{a}_{ij}$ = la cantidad esperada de transiciones desde el estado S_i al estado S_j, dividido la cantidad esperada de transiciones desde el estado S_i, o sea

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$\bar{b}_j(k) =$ la cantidad esperada de veces en el estado S_j, y que se observe el símbolo V_k, dividido la cantidad esperada de veces en el estado S_j, o sea

$$\bar{b}_j(k) = \frac{\sum_{t=1}^{T-1} \gamma_t(i)}{\sum_{s.a.\ O_t = V_k}^{T-1} \gamma_t(i)}$$

Si se repite este proceso de reestimación de los parámetros, sustituyendo a λ por $\bar{\lambda}$, entonces, podremos mejorar la probabilidad de que la secuencia O sea observada (a partir del modelo), hasta que se alcance un determinado punto crítico en el que no hay más "mejoras".

En aquellos casos en los que $\pi 1 = 1$ y $\pi i = 0$ para todo i entonces la reestimación de la distribución de las probabilidades no sería necesaria.

4 Descripción del problema y de la solución

Dado que no todo ORFs es un gen, no basta solamente con identificar los ORFs de una secuencia de ADN. Los ORFs son simplemente candidatos a genes. Es necesario contar con algún mecanismo que permita identificar cuáles son los verdaderos genes, y esto no es otra cosa que el problema que nosotros procuraremos resolver. El objetivo de nuestro trabajo es el de utilizar modelos de Markov ocultos a fin de determinar cuándo un ORF es un gen. Ahora bien, los estudios que se han realizado, y los datos estadísticos que se extraen en base a los genes que se conocen, nos brindan información adicional, la cual también puede utilizarse para aplicar filtros adicionales (y no quedarse solo con el resultado obtenido usando un HMM). De esta manera se pueden obtener resultados más exactos al momento de determinar si un ORF es un gen.

Un primer dato muy relevante es que, mientras los codones de inicio pueden encontrarse también en el interior de un gen, esto no ocurre con los codones de fin (salvo en casos muy excepcionales)[3]. Esto permite restringir considerablemente la cantidad de ORFs con posibilidades de ser genes. También se sabe que no es posible que en un mismo marco de lectura, existan genes que incluyen a otros genes más pequeños. Supongamos que uno de los codones de inicio es "ATG" (que en general lo es), un codón de fin es "TGA" (que también se sabe que es un codón de fin) y que tenemos la siguiente subsecuencia de bases en unas de las hélices de la molécula de ADN:

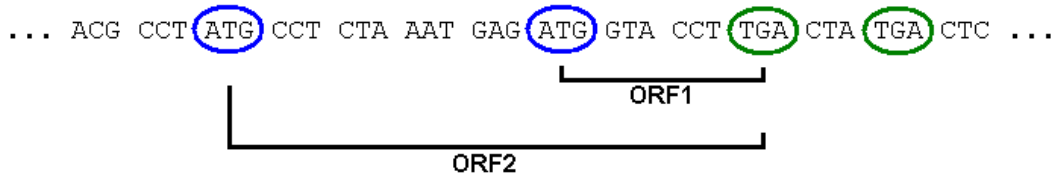


Figura 4: Identificación de ORFs en una secuencia de bases.

Como se puede ver en la figura superior, si bien el codón de fin aparece dos veces en la subsecuencia (marcado en verde), como no es posible (sin considerar a las excepciones), que exista un codón de finalización en el interior de un gen, entonces, para la subsecuencia dada, sólo existen los dos ORFs que se indican en la figura. Ahora bien, debido a que nunca ocurre que un gen contenga a otro en su interior, entonces: o bien el ORF1 de la figura no es un gen, o el ORF2 no es un

[3] Obviamente, si se buscara que el resultado de la aplicación fuera exacto (sin falsos positivos y sin falsos negativos), entonces éstos casos excepcionales también deberían ser considerados. Sin embargo, dado que son estadísticamente insignificantes hemos optado por ignorarlos dada la alta complejidad algorítmica que requiere la consideración de los mismos. Probablemente lo que ocurra es que en el resultado aparezca como gen una subsecuencia de bases que está incluida dentro del verdadero gen, o a lo sumo dos subsecuencias.

gen, o ninguno de los dos es un gen. Lo que sí es imposible es que ambos ORFs sean genes. La determinación de cual de los dos ORFs es gen, no obedece un patrón específico. Usualmente el ORF mas largo es gen, pero esto no siempre es así, y la cantidad de casos en que esto ocurre no es insignificante.

Otro dato importante que se conoce acerca de los genes, es que la longitud de los mismos varía entre 40 y 4000 bases nitrogenadas. Casi seguramente, ninguno de los dos ORFs que se muestran en la figura anterior sean genes por ser demasiado "cortos".

4.1 La aplicación de los HMM al problema

Los modelos ocultos de Markov se utilizarán para determinar que ORFs se considerarán genes de acuerdo a cuan bien se ajustan al modelo utilizado. Es decir, luego de que un ORF fue identificado, se utilizará el modelo de Markov oculto para calcular la probabilidad de ese ORF dado el modelo, y luego se tomará una decisión en función de esa probabilidad y de otros factores como el largo del ORF.

La idea es entrenar al modelo HMM (problema 1) con un training set de genes de una especie similar de forma de capturar las características de los genes de la misma de maximizar las posibilidades de detección de los genes de la especie estudiada. Luego de que el modelo es entrenado se deberá evaluar cada uno de los ORFs encontrados en el genoma y calcular su probabilidad. Con dicha probabilidad y el ORF, se aplican pos-procesamientos a los efectos de eliminar aquellos ORFs que de acuerdo al conocimiento del dominio no son posibles genes.

4.2 Trabajos relacionados

A continuación presentamos una breve reseña de uno de los trabajos relacionados más interesantes el cual utiliza modelos ocultos de Markov.

4.2.1 El enfoque de GeneMark.hmm

GeneMark.hmm [Lukashin], considera que el proceso de anotación de una secuencia de ADN bacterial consiste en definir el rol funcional de cada nucleótido que forma parte de la cadena de ADN. Para entender esto debemos definir primero que significa rol funcional.

Sea $S = \{b_1, b_2, ..., b_L\}$, un genoma bacterial completo, dónde L es el largo del genoma y b_i son cada uno de sus nucleótidos. Podríamos definir una secuencia 'funcional' $A = \{a_1, a_2, ..., a_L\}$ dónde cada a_i toma un valor entero de 0, 1 o 2. Si a_i es 0, entonces el nucleótido forma parte de una región no codificante. Si a_i vale 1 entonces el nucleótido forma parte de un gen que reside en la hebra directa del genoma. Finalmente si a_i es 2, el nucleótido a_i forma parte de un gen que reside

en la hebra complementaria del genoma. El objetivo de GeneMark.hmm es encontrar la verdadera secuencia funcional A, dada la secuencia S.

Para poder detectar genes en ambas tiras del genoma, GeneMark.hmm utiliza el modelo que se muestra en la figura 4. El mismo consiste en estados que se corresponden con las unidades funcionales de los genomas bacteriales; un estado que modela un gen típico en la cadena directa, otro para un gen típico en la cadena complementaria, otro para un gen atípico en la cadena directa, otro para un gen atípico en la complementaria, otro para las regiones no-codificantes, otros dos para los codones de inicio y de fin en la cadena directa y finalmente dos más para los codones de inicio y de fin de la cadena complementaria.

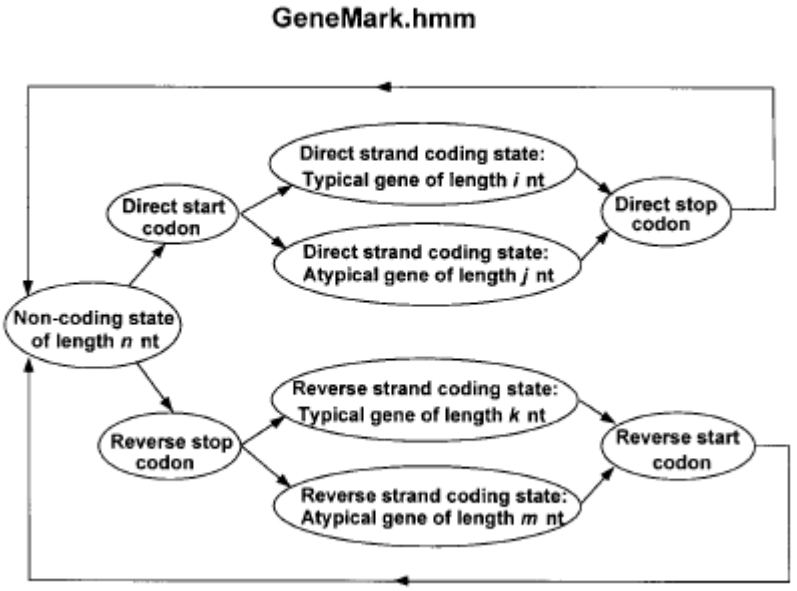


Figura 5: Modelo oculto de Markov para analizar las secuencias de nucleótidos de procariotas [Lukashin].

GeneMark.hmm utiliza un tipo especial de HMM llamado HMM con duración. Esta variante ataca una de las mayores debilidades de los modelos HMM convencionales permitiendo modelar la duración de cada estado. De este modo, el problema de encontrar la secuencia funcional A es equivalente al de encontrar el camino de estados ocultos más probables, para lo cual GeneMark.hmm utiliza el algoritmo de Viterbi [Rabiner] [Baldi-Brunak].

5 Descripción de la implementación

A continuación presentamos la implementación desarrollada en el marco del proyecto de grado. La sección se estructura de la siguiente forma: primero, se presenta un resumen de los requerimientos relevados para el sistema. Segundo, se presenta la arquitectura del sistema construido y una discusión sobre las principales decisiones de diseño adoptadas. Finalmente, se presenta el problema del análisis de los resultados y la forma en que este análisis fue realizado. Los diagramas utilizados a lo largo de esta sección para presentar la descomposición en subsistemas, así como tambien las colaboraciones entre las clases que conforman el sistema, fueron realizados en UML [UML].

5.1 Requerimientos

El sistema a construir consiste en una aplicación que asista a los investigadores del área de la bioinformática en la identificación de genes en una cadena de nucleótidos, en particular se trabajará con ADN. La identificación se realizará utilizando modelos ocultos de Markov.

El sistema debe permitir:

1. Analizar un genoma completo en busca de cuales subsecuencias de nucleótidos son posibles genes, calculando un *score* para cada una de ellas y tomando como referencia dicho *score* tomar una decisión sobre si dicha subsecuencia es un gen o no.

2. Entrenar un modelo de forma de maximizar la detección de los genes.

Considerando la gran cantidad de excepciones posibles que ocurren en el dominio, el sistema trabajará bajo determinadas restricciones que se establecen a continuación:

- Considera como posibles genes solamente aquella subsecuencias cuya longitud sea múltiplo de 3.

- No se considera como posibles genes aquellas subsecuencias que contengan un codón de fin en marco, en su interior.

- Analiza solamente la cadena directa. La extensión a la cadena complementaria quedará como trabajo a futuro.

- Como se mencionó en la sección 2.2, los genomas procariotas tienen la particularidad de ser "cerrados". Nuestra implementación no considera en un principio aquellos genes que comiencen al final del

genoma y terminen al inicio, es decir aquellos genes en los que el punto en el que el genoma fue "abierto" se encuentra dentro del gen.

5.1.1 Usuarios:

Los usuarios del sistema son biólogos investigadores que trabajan en el área de la biología molecular y bioinformática. En general poseen un manejo básico de PC y conocimientos básicos sobre HMM.

5.1.2 Requerimientos Funcionales:

A continuación se presentan los requerimientos funcionales del sistema. Para ver el resultado de la implementación de los mismos, referirse al manual de usuario de la aplicación.

5.1.2.1 Búsqueda de genes.

Dado un modelo y un genoma completo, el sistema debe evaluar el genoma de acuerdo al modelo y mostrar al usuario aquellas subsecuencias (que en definitiva son ORFs) que por su *score* sean potenciales genes. El cálculo del *score* se debe basar no solamente en el modelo y la secuencia, sino además en pos procesamientos que serán especificados por el usuario.

El sistema deberá desplegar en la salida aquellos que él "supone" son genes, pero además deberá dejar impreso un archivo con todos los ORFs y sus correspondientes *scores*. De esta manera, en caso de que alguno de los genes identificados sea un falso positivo que genera un falso negativo (probablemente como consecuencia del anidamiento de ORFs en un mismo marco de lectura), el usuario podrá beneficiarse de la existencia de este archivo. Por ejemplo, el investigador, analizando el archivo buscaría discernir cuáles de los ORFs son realmente genes a través de distintas técnicas específicas del dominio como búsqueda de genes similares en otras especies o la búsqueda por similitud del producto proteico en bases de datos de proteínas y otras.

El sistema debe permitir guardar y cargar los resultados de una búsqueda de genes mostrándolos de forma adecuada.

5.1.2.2 Entrenamiento de un modelo.

Dado un modelo y una colección de secuencias de observaciones, debe ser posible ajustar los parámetros del modelo a fin de poder entrenarlo, y así lograr que el mismo reconozca de la mejor forma posible los patrones encontrados en estas.

Las secuencias de observaciones pueden ser genes conocidos para eso organismo, pero también pueden ser genes conocidos de organismos similares al

que se pretende realizar la evaluación, o inclusive podría tratarse de estimaciones realizadas por el investigador. Ahora bien, la decisión de determinar el conjunto de secuencias de observaciones que se utilizará como *training set* depende del usuario del sistema.

5.1.2.3 Administración de modelos.

El sistema debe permitir la carga, guardado y definición de modelos HMM. Asimismo se deberá tener la posibilidad de almacenar los modelos entrenados de tal forma de no sobrescribir el modelo original.

5.1.2.4 Administración de secuencias.

El sistema debe permitir la carga de secuencias de nucleótidos (por ejemplo el genoma de un determinado organismo) desde archivos para su utilización en búsquedas o entrenamientos. Asimismo, deberá permitir cargar conjuntos de secuencias de nucleótidos (por ejemplo un conjunto de genes). Tanto las secuencias como los conjuntos de secuencias deberán estar especificadas en formato FASTA [FASTA].

5.1.2.5 Evaluación de los resultados.

El sistema debe ser capaz de poder evaluar los resultados que son obtenidos, comparándolos con la realidad conocida. O sea, si la aplicación determina que un determinado conjunto de secuencias de bases para un organismo dado son genes, entonces el sistema deberá verificar y realizar una estadística basándose en los genes que se han identificado para ese organismo.

5.1.3 Requerimientos No Funcionales:

La resolución de los algoritmos de HMM se debe hacer en un componente independiente y genérico. En cuanto al desempeño, no hay restricciones en cuanto a los tiempos de respuesta de los algoritmos y a los recursos utilizados por los mismos, pero se valorará que sean lo más acotados posibles. El sistema deberá ser capaz de funcionar de forma razonable en computadoras con al menos 256 Mb de memoria RAM, 1 giga de espacio libre en disco furo, y con una capacidad de procesamiento similar a la de un Pentium III de 700 Mhz. Por "forma razonable" queremos decir que el sistema operativo no requiera hacer uso excesivo del mecanismo de paginación para poder realizar los casos de uso.

5.2 Arquitectura de la aplicación

En esta sección se presenta la arquitectura de alto nivel de la solución propuesta.

5.2.1 Descomposición en Subsistemas:

El sistema se divide en los siguientes subsistemas:

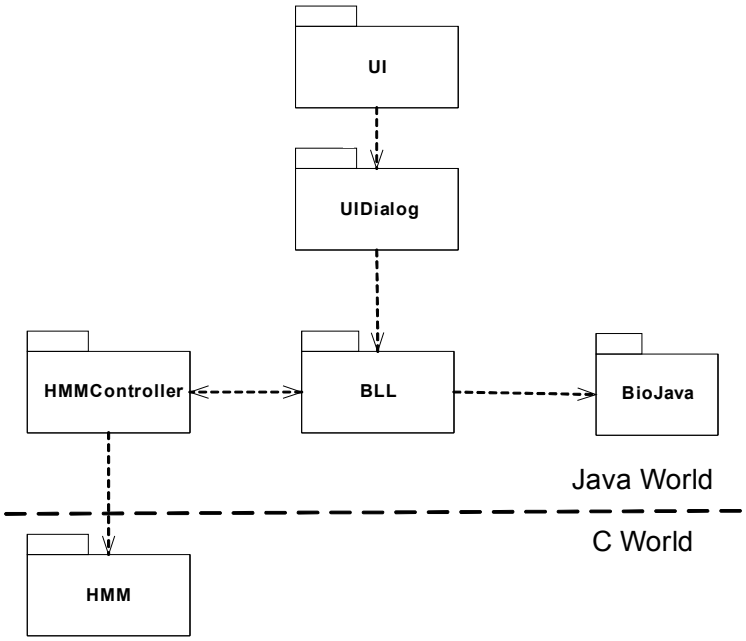


Figura 6: Descomposición en subsistemas y dependencia entre módulos.

En la figura superior, no sólo se muestra la división lógica de la aplicación, sino que también se presenta una división en lo que respecta a los lenguajes utilizados para la implementación de los subsistemas. La línea punteada delimita dos regiones, o mundos. Aquellos paquetes que se encuentren en la región superior serán implementados en Java [Java], mientras que aquellos que se encuentran en la región inferior lo serán en lenguaje C [C]. Los motivos que llevaron a esta decisión se presentan más adelante en este mismo capítulo.

5.2.1.1 HMM

El módulo HMM tiene como responsabilidad la resolución de los algoritmos de los modelos de Markov ocultos (Hidden Markov Model). A fin de considerar los requerimientos de performance, este módulo será desarrollado en lenguaje C. Al mismo tiempo, este módulo deberá ser independiente del dominio del problema, teniendo como único cometido la resolución de los algoritmos de HMM de forma genérica, tomando así en consideración los requerimientos no funcionales del sistema. Este módulo constituye una aplicación por sí sola, es decir, no es una biblioteca sino un ejecutable. Las funcionalidades del mismo se acceden por línea de comando.

5.2.1.2 HMM Controller

EL módulo HMM Controller es el encargado de mediar entre la aplicación en Java y el sistema en C, resolviendo los problemas específicos de comunicación entre ambos mundos. Es el encargado de invocar la aplicación que resuelve los problemas de los HMM, al mismo tiempo que debe capturar las señales y salidas que esta genera. De esta forma la aplicación en C puede ser accedida de forma transparente por el resto de los módulos.

5.2.1.3 BLL

El módulo BLL (Bussiness Logic Layer) contiene los objetos del negocio y la lógica de los mismos, lo que abarca los pre procesamientos a realizar antes de utilizar el motor de HMM y los pos procesamientos que refinan los resultados obtenidos, los objetos auxiliares que permiten la carga y descarga de archivos, y otras funcionalidades auxiliares relacionadas con el dominio. Es en éste módulo donde se implementan todas las funcionalidades que son utilizadas por los casos de uso.

5.2.1.4 BioJava

Los módulos BLL y SSL dependen de BioJava [BioJava], una librería "*open-source*" que provee un *framework* para el procesamiento de datos biológicos. Las principales funcionalidades que brinda son: manipuladores de secuencias y *parsers* de archivos; acceso a diversas bases de datos con contenido relacionado a la biología molecular y a la bioinformática; rutinas para el análisis y el estudio estadístico de problemas relacionados en el área en cuestión. En el marco de la aplicación desarrollada, la librería se utiliza para el procesamiento de archivos de secuencias de nucleótidos.

5.2.1.5 UIDialog

El módulo UIDialog es el encargado de controlar el flujo de los casos de uso y de mantener la sesión del usuario. Es el que genera los eventos necesarios para poder informar al usuario de los pasos que están siendo realizados por la aplicación.

5.2.1.6 UI

El módulo UI (sigla en inglés para "Interfaz de Usuario") es el que implementa la interfaz de usuario. Simplemente resuelve los problemas de presentación de forma de que el usuario pueda interactuar con el sistema.

5.3 Principales decisiones de diseño

5.3.1 La implementación del motor en lenguaje C y de la interfaz de usuario y el resto de la lógica en Java.

Una de las primeras decisiones de diseño tomadas fue la de implementar el módulo HMM en lenguaje C mientras que el resto del sistema sería implementado en un lenguaje de alto nivel.

Inicialmente no teníamos muy claro si era necesario implementar los algoritmos en un lenguaje más eficiente y el resto de la aplicación en un lenguaje de más alto nivel (o mejor dicho, que ejecute sobre una máquina virtual como ser Java o los lenguajes de la plataforma .NET de Microsoft), facilitando así la programación de los algoritmos) o si sencillamente podíamos implementar la totalidad de la aplicación en un lenguaje de alto nivel. Otra alternativa que manejamos al comienzo fue la de generar código C que se pudiera compilar y ejecutar y resolviera los algoritmos. De esta forma, se utilizarían constantes en lugar de variables lo que aumentaría el desempeño. Para evaluar estas distintas alternativas se implementaron tres versiones simplificadas del algoritmo de evaluación. La primera fue implementada en C#, uno de los lenguajes que se ejecuta sobre la plataforma .NET de Microsoft [C#] (elegido como representante de lenguaje que ejecuta sobre una máquina virtual), la segunda en C utilizando variables (lo llamamos C dinámico) y finalmente una tercera versión en C la cual tenía fijados los valores de los parámetros en el código del algoritmo (lo llamamos C estático). La siguiente tabla muestra los resultados obtenidos.

Largo de la secuencia	Tiempo en milisegundos		
	C Dinámico	C Estático	C#
1E+06	31	31	187
2E+06	63	63	406
3E+06	78	78	625
4E+06	94	109	828
5E+06	140	140	1031
6E+06	172	157	1312
7E+06	187	187	1468
8E+06	203	234	1625
9E+06	250	235	1875
1E+07	266	266	2046

Figura 7: Tabla comparativa entre distintas implementaciones.

Como se puede apreciar en la tabla, las diferencias en tiempo entre las implementaciones en C y la implementación en C# son notorias. Esto fue lo que nos motivó a utilizar C como lenguaje para implementar los algoritmos. También se puede notar que las diferencias entre la versión estática y la dinámica del

algoritmo en C sin mínimas. Esto también nos sirvió para descartar la idea de generar código en C que luego sería compilado. Por lo tanto, optamos implementar los algoritmos de los HMM en C especificándole de alguna forma los parámetros con los que se debía ejecutar.

La decisión de utilizar otro lenguaje para la implementación de la interfaz de usuario y del resto de la lógica de la aplicación (o sea sin considerar los algoritmos) se basó en el hecho de que el desarrollo se haría mucho mas rápido que utilizando C. El hacerlo nos permitiría abstraernos de una gran cantidad de detalles (como por ejemplo el manejo de punteros), aumentando así nuestra capacidad de programación y aumentando la velocidad del desarrollo.

Dado que no había un requerimiento que determinara la plataforma en que debía funcionar nuestra aplicación, optamos por Windows dado que era la que más se utilizaba por parte de los potenciales usuarios del sistema. En base a esta decisión y a nuestros conocimientos se manejaron dos alternativas a la hora de elegir el lenguaje de programación. La primera de ellas era la plataforma .NET y en particular el lenguaje C#. La otra alternativa era utilizar Java.

Lo que nos llevó a utilizar Java como lenguaje de programación fue el haber encontrado la librería *open source* Biojava [BioJava], la cual implementa muchas funcionalidades que facilitan el desarrollo de aplicaciones en el área de la bioinformática.

5.3.2 La interacción entre el mundo Java y el mundo C

La interacción entre el mundo Java y el mundo C se realiza en la clase HMMController, la cual, centraliza el manejo de la comunicación con el motor en C para la resolución de los problemas asociados a los HMM. Esta clase fue concebida con la idea de que fuera un proxy del motor HMM para el mundo java. De este modo se logró que toda la lógica asociada al procesamiento de los distintos mensajes enviados por el motor y a la preparación de las secuencias y los modelos para la ejecución del mismo quedara centralizada en un solo lugar, lo que mejora la calidad del sistema y facilita su testeo.

Otras de las decisiones de diseño importantes a las que nos vimos enfrentados fue el determinar el mecanismo que se utilizaría para la interacción entre el mundo C y el mundo Java. En un principio, esta comunicación, se pensaba realizar utilizando *pipes* y redirigiendo la entrada y la salida del motor de modo de poder controlarlo desde el mundo Java. Finalmente, y debido a la gran cantidad de datos que se precisaba pasar en un sentido y en otro se decidió utilizar esta técnica en conjunto con archivos temporales. Así, la entrada/salida estándar se utiliza para la comunicación (como por ejemplo mensajes que indican la cantidad de datos que han sido procesados, o la actividad en particular que el motor está

realizando en este momento); mientras que los archivos temporales son utilizados para el pasaje de secuencias y modelos.

Además, la clase HMMController es la responsable de coordinar la traducción y escritura de los modelos y las secuencias, para que el motor luego las pueda procesar.

Las operaciones más importantes que esta clase provee son básicamente dos. La primera, se encarga de resolver el problema de entrenamiento de un modelo. La segunda resuelve el problema de evaluación de las probabilidades de un conjunto de secuencias, ya sea utilizando secuencias en memoria, como también utilizando secuencias desde archivos.

Otra importante responsabilidad es leer los mensajes enviados por el motor e interpretarlos correctamente generando los eventos adecuados para que el resto del sistema se mantenga actualizado sobre el estado de los distintos procesamientos que son realizados por el motor.

5.3.3 Decisiones sobre el entrenamiento

5.3.3.1 El entrenamiento y el manejo de memoria.

Uno de los mayores problemas al que nos enfrentamos al tratar de implementar el algoritmo de entrenamiento fue el manejo de la memoria. Dado que el algoritmo de entrenamiento realiza repetidas veces los mismos cálculos, es posible introducir técnicas de programación dinámica a fin de reducir el uso del procesador, almacenando información que luego puede ser reutilizada.

El problema es que debido a la gran cantidad de datos que deben manejarse, la cantidad de memoria necesaria para el almacenamiento fácilmente desborda la capacidad actual de la mayoría de los computadores. Esto conduce a que el sistema operativo realice la paginación al disco duro, lo cual repercute negativamente en el desempeño del sistema y lleva a que el tiempo de ejecución de los algoritmos sea aun peor a los que se pueden obtener sin utilizar la programación dinámica.

La solución elegida para la resolución de este problema, consiste en la creación un modulo, al que llamamos DataProvider, cuya responsabilidad es saber calcular los valores de las variables alfa, beta y de los coeficientes de escalado y almacenarlos en memoria cuando corresponda, para que la siguiente vez que el algoritmo requiera esos valores no los tenga que calcular nuevamente.

El DataProvider regula el almacenamiento en memoria a través de un parámetro que especifica el porcentaje de los valores que deben ser almacenados en memoria. Por ejemplo, 100% significa que el DataProvider almacenará todos los resultados en memoria, mientras que 0% significa que ningún resultado será

almacenado, y que por lo tanto siempre volverán a ser recalculados. Cuanto más se almacena en memoria, la velocidad con la cual se ejecuta el algoritmo de entrenamiento es mayor, pero aumenta el riesgo de que la memoria no sea suficiente y se pagine en disco, con lo cual el desempeño caerá drásticamente. Cuanto menos se almacena en memoria, la velocidad con la cual se ejecuta el algoritmo de entrenamiento es menor, ya que se deben calcular nuevamente más variables.

A partir de este porcentaje el DataProvider determina la cantidad de lugares que puede almacenar por secuencia y los distribuye uniformemente a lo largo de la misma. Es decir, si al DataProvider se le indica un 50% para una determinada secuencia, 1 de cada 2 lugares será almacenado en memoria, y el restante será calculado a partir del anterior o el siguiente, de acuerdo a lo que sea más conveniente.

La cantidad de memoria consumida por el algoritmo de entrenamiento, depende de la cantidad de estados y símbolos de emisión del HMM (arquitectura definida por el usuario), y de la cantidad y el largo de las secuencias de nucleótidos que componen el *training-set*, Para el organismo que más se estudio en el marco del proyecto de grado, el *Escherichia coli K12* (tres millones de pares de base aprox), utilizando modelos de máximo nueve estados y cuatro símbolos de emisión, y *training-sets* con ciento treinta secuencias de un largo promedio de trescientas bases nitrogenadas, 512 MB de memoria era suficiente para poder almacenar todos los cálculos intermedios en memoria.

5.3.3.2 La utilización del escalado.

Uno de los problemas con los que nos encontramos en el desarrollo del proyecto (y que además desde un comienzo supimos que deberíamos solucionar) fue el de lidiar con las limitaciones de precisión en los cálculos que se realizan con las computadoras. La variable de *forward* por ejemplo, consiste en la suma de una gran cantidad de términos, dónde cada término es menor que 1 (y generalmente significativamente menor que 1). Además, cuanto más larga es la secuencia más pequeños son cada uno de los valores.

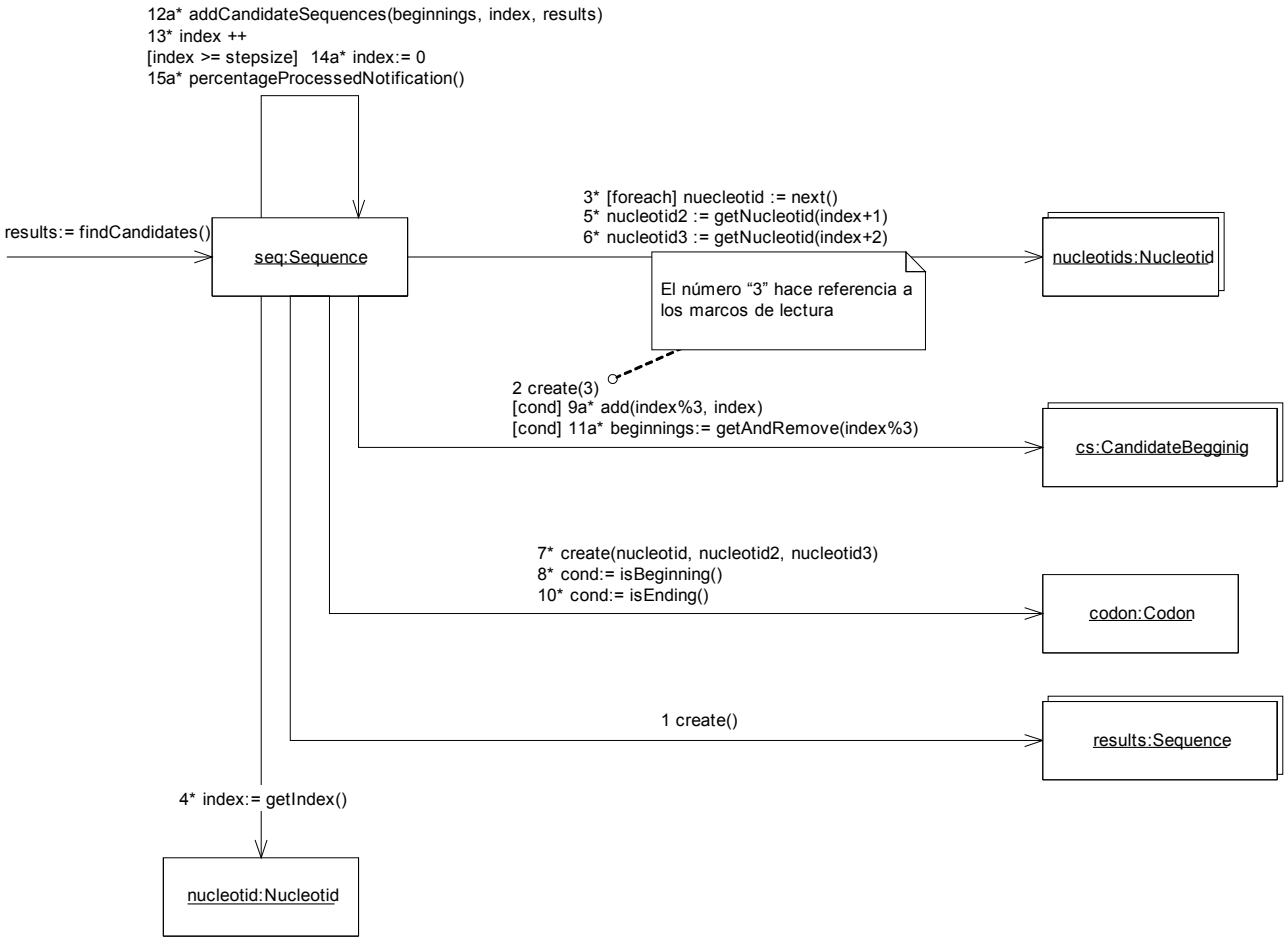
Por esta razón, y para largos de secuencias suficientemente grandes (un poco mayores que 100), cualquier computadora actual será excedida en su precisión. En el lenguaje C, los números en punto flotante (que generalmente son los que se utilizan para las implementaciones que resuelven estos problemas) se representan con 32 bits, lo cual permite manejar sin iconvenientes números cuyo valor absoluto sea mayor que 10^{-16} aproximadamente. Valores más pequeños que ese umbral son transformados a ceros, dado que la máquina no puede representarlos en esos 32 bits. En el caso particular de nuestro proyecto, algunas de las secuencias son de más de 3000 observaciones, por lo que se nos hizo

indispensable introducir un mecanismo de escalado. El procedimiento de escalado se detalla en el Anexo 1 del presente documento.

5.3.4 Decisiones sobre la búsqueda de genes

5.3.4.1 La búsqueda de secuencias candidatas

La primer fase en la búsqueda de genes es la detección de las secuencias candidatas, es decir la identificación de aquellos marcos de lectura (ORF) con posibilidades de ser genes. A continuación mostramos los diagramas de colaboración correspondientes a la operación encargada de realizar este proceso.



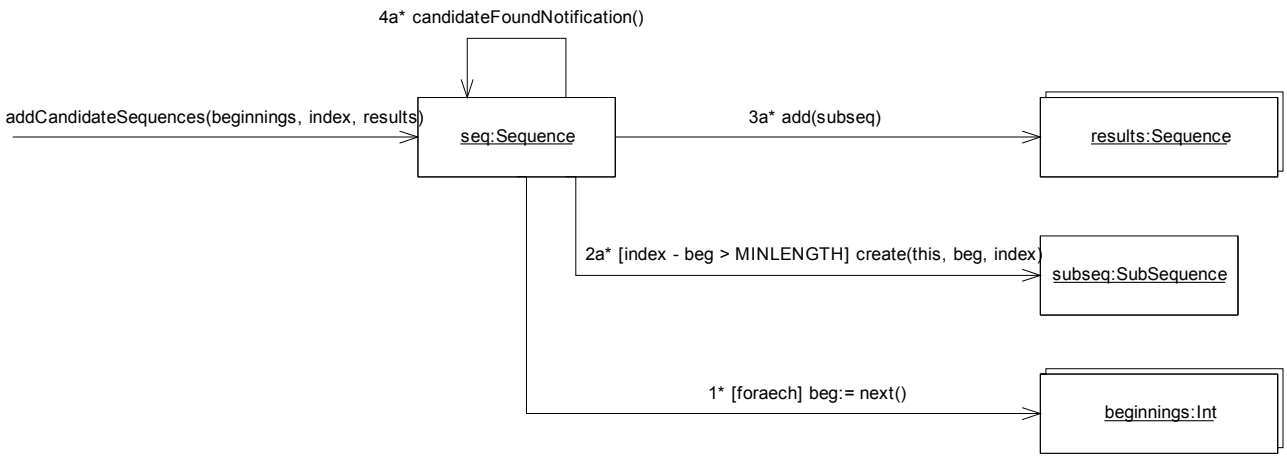


Figura 8: Diagramas de colaboración que muestran como se resuelve la búsqueda de subsecuencias candidatas (ORFs) en una secuencia.

5.3.5 Las políticas de pos-procesamiento

Luego de calculadas las probabilidades de las distintas secuencias candidatas, la herramienta aplica varios pos-procesamientos los cuales son configurables y extensibles por parte del usuario. La decisión de definir un mecanismo de pos-procesamiento configurable y general se tomo debido a que existían varias formas posibles de utilizar los resultados de la evaluación de las secuencias utilizando HMM para evaluar si un determinado ORF es un gen o no.

De todas formas, para el análisis de los resultados se definieron tres pos-procesamientos estándares que permiten tomar una decisión sobre un determinado ORF. Estos tres pos-procesamientos son los siguientes:

- Cálculo de un *score* a partir de la probabilidad
- Resolución de los casos de inclusión de ORFs.
- Toma de decisión sobre si un ORF es un gen o no.

El problema con la probabilidad que resulta de la evaluación con el HMM es que para toda secuencia incluida en otra, la probabilidad de la primera siempre es mayor o igual que la de la segunda; lo que no concuerda con la realidad del dominio, en donde como regla general, cuanto más larga es la secuencia, más probable es que sea un gen.

Por ello, el cálculo del *score* a partir de la probabilidad se hace normalizando la probabilidad obtenida contra el modelo deseado contra la que se tendría si se la evaluara utilizando el modelo nulo. El modelo nulo no es más que un modelo que

consta de un solo estado cuya probabilidad de transición hacia si mismo es 1 y cuyas probabilidades de emisión de los cuatro símbolos del alfabeto es 0.25. Es decir, en fórmulas:

$score = \frac{p}{0.25^n}$ donde n es el largo de la secuencia que se está evaluando.

Esta forma de calcular el *score* fue sugerida por los docentes especialistas en el dominio como forma de obtener un *score* más independiente de la longitud de la secuencia.

Como consecuencia de la utilización del escalado para no perder precisión en los cálculos, no es posible considerar directamente la probabilidad de emisión de una secuencia, dado que la misma tiende muy rápidamente a cero a medida de que el largo de la secuencia aumenta. Es por ello que el motor que resuelve el problema de evaluación de un HMM, devuelve el logaritmo de la probabilidad de que una observación se dé para un modelo particular, y no la probabilidad en si misma. Así, el *score* en realidad queda expresado de la siguiente forma:

$score = \log(p) - n\log(0.25)$

El segundo procesamiento consiste en analizar los ORFs incluidos unos dentro de otros de forma tal de poder determinar cuál de ellos debe ser considerado gen (en caso de que alguno de ellos lo sea. Puede darse el caso de que se tengan muchos ORFs anidados y que ninguno sea realmente un gen). El pos-procesamiento desarrollado con este fin permite que se configure la cantidad de ORFs, del más largo al más corto, que serán considerados a la hora de tomar la decisión. Este parámetro se encuentra en el archivo de configuración de la aplicación. Tomando en cuenta estos ORFs se toma una decisión en base a los *scores* de cada uno.

Los resultados mostrados en el documento utilizan la política de que si alguno de los ORFs en esta situación debe ser un gen, entonces se debe elegir el más largo. Esta decisión se tomó considerando la realidad del dominio, dado que en la mayoría de los casos es la decisión correcta. Además, luego de realizar algunas pruebas, se comprobó que el *score* calculado en el paso anterior no era un buen referente para la toma de esta decisión.

El tercer pos-procesamiento consiste en decidir si un determinado ORF es un gen o no. Esa decisión se toma en función del largo del ORF y del *score*. Si el largo de la tira es mayor que un umbral configurable en el archivo de configuración de la aplicación entonces el ORF se considera un gen (en el caso de un grupo de ORFs incluidos unos dentro de otros, solamente uno será considerado). Si el largo del ORF se encuentra entre un mínimo configurable y el umbral anterior entonces se utiliza el *score* para decidir si es un gen o no (en el caso de grupos, si el

representante del grupo obtenido en el paso anterior lo es o no). El pos-procesamiento implementado toma la decisión en función de si el *score* es mayor que un *score* mínimo configurable.

Este pos-procesamiento se hace de esta forma debido a que, a partir de determinados largos (dependientes del tipo de organismo en cuestión), la probabilidad de que un ORF no sea un gen es muy baja. En particular se consideró que en esos casos era más probable que el programa erróneamente los rechazara por su *score*, a que fuera aceptado erróneamente por su longitud. Obviamente el cumplimiento de hipótesis anterior depende del umbral establecido para el largo. Es por ello que éste parámetro debe ser elegido con cuidado a la hora de buscar genes.

5.4 El análisis de los resultados.

Para analizar los resultados se creó un caso de uso cuya función es realizar la búsqueda de genes y luego comparar los resultados dados contra la lista de genes detectados para el genoma dado.

El objetivo de este análisis es poder determinar la eficacia de nuestra herramienta a la hora de encontrar genes sobre genomas conocidaos como forma de poder evaluar que tan útil podría llegar a resultar a la hora de realizar una búsqueda real sobre un genoma desconocido.

El proceso analiza los resultados obtenidos buscando determinar:

- Genes encontrados
- Genes no encontrados
- Genes no analizados
- Falsos positivos
- Negativos verdaderos

A continuación definimos que significa cada uno de estos términos en el contexto del análisis de resultados realizado.

Genes encontrados, son aquellas secuencias que siendo realmente un gen, (donde ser realmente un gen en este contexto significa estar en el archivo de genes), el programa las detectó como posibles secuencias candidatas, las analizó y concluyó correctamente que realmente eran genes.

Genes no encontrados, son aquellas secuencias que siendo realmente un gen, el programa las detectó como posibles secuencias candidatas, las analizó y concluyó erróneamente que no eran genes.

Genes no analizados, son aquellas secuencias que siendo realmente un gen, el programa no las detectó como secuencias candidatas, posiblemente debido a las restricciones impuestas, en particular el largo mínimo o los codones de inicio y de fin; y por lo tanto no fueron analizadas.

Falso positivo, es toda secuencia que no siendo realmente un gen, el programa la detectó como posible candidata, la analizó y concluyo (erróneamente) que era un gen.

Negativo verdadero es toda secuencia que no siendo realmente un gen, el programa la detectó como posible secuencia candidata, la analizó y concluyo (correctamente) que no era un gen.

Ejemplo de los resultados:

```
Genes Statistics
Genes Found: 846 ( 40.34335% )
Genes Not Found: 989 ( 47.162613% )
Not Analyzed: 261 ( 12.446352% )
Result Statistics
False Positive: 1317 ( 1.9654369% )
True Negative: 63856 ( 95.29608% )
Genes Found: 846 ( 1.2625358% )
Genes Not Found: 989 ( 1.4759432% )
```

Este análisis de resultados distingue solamente lo que podríamos llamar distinciones exactas, pero no analiza los resultados considerando que muchos de los ORFs analizados se encuentran unos incluidos en otros. El análisis de los resultados considerando la inclusión de los ORFs, arrojaría mejores resultados mirados desde la óptica de un ser humano. Para mostrar este punto, a continuación analizamos un caso concreto.

Estudiando el genoma de *Escherichia coli* podremos ver que hay un gen entre las posiciones 8238-9191 de la hebra directa del genoma. La particularidad de este gen es que es uno de los tantos casos en el que el gen no es el ORF de mayor longitud que finaliza en esa posición. Es decir estamos ante un caso como el que se muestra en la figura 3 en la que el ORF1 es el gen.

Si utilizamos el caso de uso de encontrar y comparar para analizar los resultados sobre el genoma de *Escherichia coli*, veremos entre otros resultados, lo siguiente:

```
start=8175  end=9191    score=-54.609     length=1017 False Positive
start=8238  end=9191    score=-50.155     length=954 Gene Not Found
```

Podemos apreciar que la herramienta detectó como posible candidatos a ambos ORFs y concluyó erróneamente que el más largo era un gen y que los más

cortos no lo eran (debido a la política de pos procesamiento elegida), cuando en realidad uno de ellos si lo era.

Un investigador que leyera los resultados del programa vería ORFs incluidos unos en otros y sus respectivos *scores* y tendría varias alternativas, por ejemplo: podría buscar el gen en las bases de datos de genes en búsqueda de un gen similar en alguna otra especie y de esa forma tratar de determinar cuál de todos es realmente el gen. O segundo, podría utilizar el producto proteico generado por cada uno de los distintos ORFs para hacer búsquedas por similaridad con proteínas encontradas en otras especies y de esa forma tener una mejor pista de cuál de ellos (si es que alguno de ellos lo es) es el gen. Es en estos casos, en el que la herramienta juega el rol de asistente de búsqueda.

6 Evaluación de resultados

6.1 Modelos utilizados

Para la evaluación de resultados se utilizaron los siguientes modelos:

Modelo 1: Modelo de un solo estado con probabilidad de volver a si mismo igual a 1 e iguales probabilidades de emisión de los distintos símbolos:

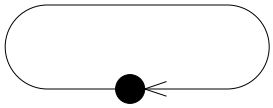


Figura 9: Modelo de un solo estado

Modelo 2: Modelo de 3 estados cíclico e iguales probabilidades de emisión para los distintos estados y símbolos:

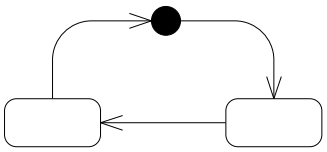


Figura 10: Modelo de tres estados que forman un ciclo.

Modelo 3: Modelo de 9 estados con ciclo interior e iguales probabilidades de emisión para todos los símbolos:

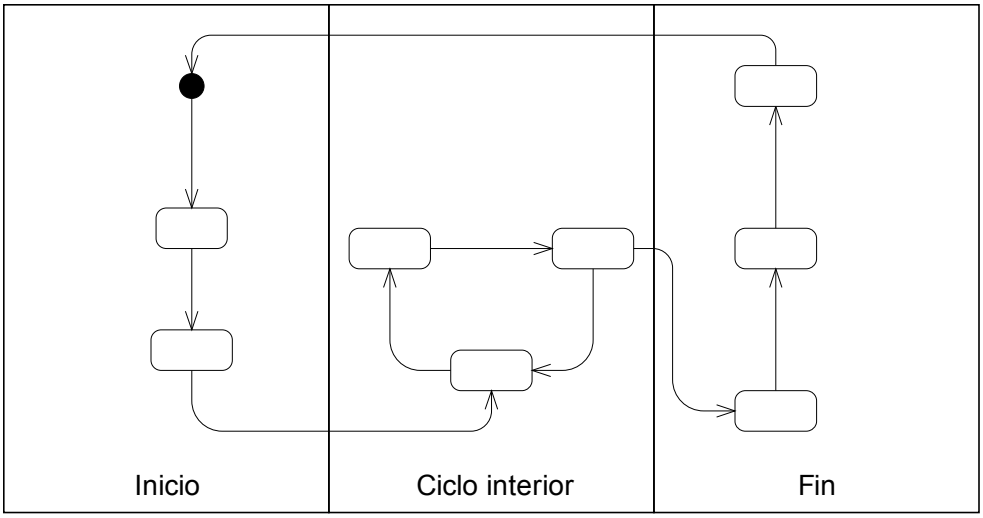


Figura 11: Modelo de 9 estados y ciclo interior.

6.2 Genoma, genes y training sets utilizados.

Para las pruebas se utilizó el genoma de E. Coli K12: NC_000913 obtenido de [PUBMED]. El conjunto de entrenamiento se obtuvo tomando 135 genes al azar sobre la hebra directa del genoma.

6.3 Resultados Obtenidos

A continuación se muestran y comentan los resultados obtenidos de ejecutar la aplicación con los modelos descritos en la sección 5.1. Dividimos el análisis de los mismos separando dos configuraciones distintas. La primera configuración considera que un gen puede tener un largo mínimo de 300 bases y un largo máximo de 750, y un valor de *score* mínimo de -15 (es decir, todos aquellos genes cuyo *score* supere el valor -15 se considera un gen). La segunda configuración considera que un gen puede tener un largo mínimo de 100 bases y un largo máximo de 1000, y un valor de *score* mínimo de -20. A su vez, para cada configuración de parámetros se analizan los resultados obtenidos utilizando el modelo original, y el modelo entrenado utilizando como conjunto de entrenamiento el mencionado en el punto 5.2

6.3.1 Configuración A

Tanto para el Modelo de un estado, como para el Modelo de tres estados que forman un ciclo, como para el Modelo de 9 estados y ciclo interior (los tres sin entrenar), los resultados obtenidos fueron idénticos y se muestran a continuación[4]:

Genes Encontrados: 1285 (61,28%)[5]

Genes no analizados: 213 (10,16%)

Genes no encontrados: 599 (28,56 %)

Total de Genes: 2097

Falsos Positivos: 2864

Verdaderos Negativos: 62107

Total de Resultados: 66855

Como se puede apreciar, se encuentran más genes que los que no se encuentran o no se analizan juntos. El resultado, lejos de ser el más deseado, no deja de ser alentador, pues se logro identificar más de la mitad de los genes del genoma en un tiempo bastante corto. Ahora bien, la salida final de la aplicación,

[4] El valor total de resultados, consiste en la suma de los genes encontrados, los genes no encontrados, los falsos positivos y los verdaderos negativos.
[5] Estos porcentajes se calculan en función del valor Total de Genes.

también incluye a los falsos positivos, los cuales constituyen un número bastante elevado, siendo superior al conjunto total de genes. Entre las varias razones que hacen que el número de falsos positivos sea tan elevado, podemos mencionar algunas: 1) Los modelos utilizados. Los modelos utilizados son extremadamente genéricos, no aprovechando ninguna característica particular del organismo cuyo genoma se está estudiando; 2) La manera en que se calcula el *score* de las secuencias de codones. Posiblemente esta sea la causa más importante de la existencia de un número de falsos positivos tan elevado. A la hora de decidir que secuencia de codones es un gen o no, se deben tener en cuenta varios factores, entre ellos el largo de la secuencia, la cantidad de bases de determinado tipo, etc. El cálculo de *score* que se utiliza actualmente se elaboró en conjunto con los docentes de la Facultad de Ciencias, pero es una primera aproximación a un política que pondere a las secuencias que realmente son genes, y penalice a las que no lo son. Un usuario puede definir su propio mecanismo para asignarle un *score* a las secuencias de codones, utilizando el mecanismo de pos-procesamientos extensibles que brinda la aplicación; 3) Los valores de configuración elegidos, esto es, la longitud mínima, máxima y horizonte elegidos. Cuanto menor sea la longitud mínima, mayor sea la longitud máxima, y menor sea el horizonte, el número de falsos positivos crecerá, ya que la cantidad de secuencias que se analizarán será mayor.

Luego de entrenados los modelos, se repite el procedimiento anterior, obteniendo los siguientes resultados:

Para el modelo de un estado:

Genes Encontrados: 1285 (61,28%)

Genes no analizados: 213 (10,16%)

Genes no encontrados: 599 (28,56 %)

Total de Genes: 2097

Falsos Positivos: 2864

Verdaderos Negativos: 62107

Total de Resultados: 66855

Para el modelo de tres estados que forman un ciclo:

Genes Encontrados: 930 (44,35%)

Genes no analizados: 213 (10,16%)

Genes no encontrados: 954 (45,49%)

Total de Genes: 2097

Falsos Positivos: 1538

Verdaderos Negativos: 63433

Total de Resultados: 66855

Finalmente, para el modelo de nueve estados con ciclo interior:

Genes Encontrados: 916 (43,68%)

Genes no analizados: 213 (10,16%)

Genes no encontrados: 968 (46,16%)

Total de Genes: 2097

Falsos Positivos: 1482

Verdaderos Negativos: 63489

Total de Resultados: 66855

Excepto para el caso del modelo de un estado solo, para los otros modelos ocurre que la cantidad de genes encontrados baja, la cantidad de genes no encontrados aumenta y la cantidad de falsos positivos también disminuye. Estos resultados a primera vista, pueden llevar a pensar que entrenar los modelos no es bueno, ya que se reconocen menos genes. Sin embargo, no hay que olvidar que la salida de la aplicación incluye a los falsos positivos, los cuales disminuyen en mayor proporción que los genes encontrados. Debemos entonces en esta instancia, mencionar los efectos que hemos encontrado tiene el entrenamiento sobre los modelos, para intentar dar respuesta a los resultados obtenidos luego de entrenar. En primer lugar debemos decir que el entrenamiento tiene un comportamiento totalmente determinista, y que si bien los valores de los coeficientes de las matrices de emisión y transición no tiene porque dar siempre lo mismo dados los valores iniciales de éstas[6], en la práctica los valores de las matrices convergen a un mismo valor, independientemente de los valores iniciales que tengan. En segundo lugar hemos observado que luego de entrenar con ciertas secuencias, si se prueba evaluarlas, se observará que la probabilidad (*score*) de muchas de ellas disminuye, pero el de otras aumenta, y en todos los casos que hemos probado, son mas las secuencias cuyo *score* aumenta, que las que disminuyen. Esto quiere decir que al entrenar, se premia la estructura de cierto tipo de secuencias, determinado por el conjunto de secuencias elegido para

[6] Esto es, porque el algoritmo utilizado para implementar el entrenamiento calcula óptimos locales, basta con modificar el orden en el que el algoritmo de entrenamiento recibe las secuencias que usa para entrenar, y los valores a los que converge el algoritmo probablemente varíen.

entrenar. Esto se puede comprobar ejecutando el caso de uso "entrenar y evaluar", el cual se implementó especialmente para corroborar lo dicho anteriormente. Mencionados estos dos aspectos, los resultados obtenidos no deberían de sorprender (si bien no son los deseados). Luego de entrenar, las tiras que tengan la estructura que pondera el entrenamiento tendrán mayor *score*, que no necesariamente coincide con las secuencias que son genes. Esto hace que ambos valores, los genes encontrados y los falsos positivos bajen: Los falsos positivos porque se entrena con secuencias que tienen la estructura de genes (genes en este caso), y los genes encontrados porque es muy probable que alguna secuencia que es gen, no tenga la misma estructura que el común de las secuencias con las cuales se entrenó. De esta manera, es coherente que la cantidad de genes encontrados disminuya, al igual que los falsos positivos, y que este último valor lo haga en mayor proporción que el primero.

Se puede observar a su vez que la cantidad de genes encontrados es menor en los modelos que tiene mayor cantidad de estados. Sin embargo la cantidad de falsos positivos disminuye en mayor proporción, lo que se puede considerar como una mejora en el resultado global. Recordamos una vez más que los modelos elegidos son genéricos, y no necesariamente benefician el cálculo del *score* .

6.3.2 Configuración B

Tanto para el Modelo de un estado, como para el Modelo de tres estados que forman un ciclo, como para el Modelo de 9 estados y ciclo interior, los resultados obtenidos son similares y se muestran a continuación:

Genes Encontrados: 1410 (67,23%)

Genes no encontrados: 666 (31,76%)

Genes no analizados: 21 (1,01%)

Total de Genes: 2097

Falsos Positivos: 22481

Verdaderos Negativos: 115570

Total de Resultados: 140127

Se puede apreciar en los resultados que se muestran arriba, el efecto que una mala elección de los parámetros del sistema tiene sobre el resultado que devuelve la aplicación. Recordemos que en la configuración B se considero que el largo mínimo es de 100, el largo máximo es de 1000 y el *score* mínimo de -20. La cantidad de genes encontrados es mayor, pero la cantidad de falsos positivos es abrumadoramente superior (22481 contra 2864). Esto es lógico ya que la cantidad

de secuencias que han de considerarse es mucho mayor, y se esta siendo mucho mas permisivo a la hora de decidir cual secuencia es gen y cual no (el horizonte bajo a -20 lo que significa que todas aquellas secuencias cuyo *score* sea superior a esta cifra el sistema la toma como un gen. Nótese, por otro lado, que la cantidad de genes no analizados disminuyo, lo cual también es lógico ya que el rango de secuencias que se cubren es mucho mayor. Cabe aclarar que a nivel global, los resultados obtenidos no son buenos en absoluto, siendo los obtenidos en la configuración A mucho mejores.

Con el único fin de realizar un análisis similar al hecho para la configuración A, se muestran los resultados obtenidos luego de entrenados los modelos y de repetir el proceso anterior. Los comentarios son los mismos que para la configuración A:

Para el modelo de un estado:

Genes Encontrados: 1410 (67,23%)

Genes no encontrados: 666 (31,76%)

Genes no analizados: 21 (1,01%)

Total de Genes: 2097

Falsos Positivos: 22481

Verdaderos Negativos: 115570

Total de Resultados: 140127

Para el modelo de tres estados que forman un ciclo:

Genes Encontrados: 931 (44.40%)

Genes no encontrados: 1145 (54,6)

Genes no analizados: 21 (1%)

Total de Genes: 2097

Falsos Positivos: 21398

Verdaderos Negativos: 116653

Total de Resultados: 140127

Finalmente, para el modelo de nueve estados con ciclo interior:

Genes Encontrados: 893 (42,6%)

Genes no encontrados: 1183 (56,41%)

Genes no analizados: 21 (0,99%)

Total de Genes: 2097

Falsos Positivos: 21268

Verdaderos Negativos: 116783

Total de Resultados: 14012

6.4 Comparación con GeneMark.hmm

Hoy en día, GeneMark.hmm es el programa que obtiene mejores resultados a la hora de buscar genes utilizando modelos de Markov ocultos [Baldi-Brunak]. Debido a esto hemos creído conveniente realizar una comparación de nuestros resultados con los de GeneMark.hmm [Lukashin].

Si bien el porcentaje de aciertos de GeneMark.hmm es superior al 70%, ese resultado no utiliza simplemente modelos de Markov ocultos, sino que además introduce técnicas estadísticas que mejoran el resultado. Ahora bien, en [Lukashin] también aparecen los resultados que se obtienen utilizando únicamente Modelos de Markov ocultos, y es con éstos resultados que realizaremos la comparación. En ambos casos los resultados se obtienen analizando el genoma de *Escherichia coli*. Para el caso particular de nuestra aplicación estaremos usando la configuración A, con el modelo de 9 estados.

Otro aspecto a tener en cuenta es que el análisis de resultados de GeneMark.hmm no sólo considera los genes encontrados y los no encontrados, sino que además distingue una nueva categoría que son los genes "parcialmente encontrados". Esta categoría incluye aquellas secuencias de bases que GeneMark.hmm considera que son genes, cuando en realidad el gen está en otro nivel de anidamiento (esto se comentó al final de la sección 5.4). Dado que nosotros no consideramos esta categoría y a fin de poder realizar la comparación, cada secuencia que GeneMark.hmm toma como gen "parcialmente encontrado" la consideraremos como un falso positivo. Al mismo tiempo el gen parcialmente encontrado lo agregamos a los genes no encontrados, dado que en los resultados de GeneMark.hmm éstos no son considerados de ésta forma.

La siguiente tabla muestra los porcentajes de genes encontrados, el porcentaje de genes no encontrados, y el porcentaje de falsos positivos respecto al total de genes.

	GeneMark.hmm	Geno+
Genes encontrados	57.9	43.7
Genes no encontrados	42.1	56.3
Falsos Positivos	45.6	70.7

Figura 12: Tabla comparativa de GeneMark.hmm con Geno+.

Como se puede apreciar en la tabla, los resultados obtenidos por GeneMark.hmm son sensiblemente mejores que los nuestros. Creemos que hay dos razones que explican esta diferencia. La primera refiere al enfoque aplicado al usar los HMMs (véase 4.2.1). La segunda es la complejidad del modelo (o arquitectura) utilizado. El modelo usado por GeneMark.hmm distingue genes típicos y atípicos, mientras que todos los modelos utilizados por nosotros no realizan ninguna distinción entre ellos.

7 CONCLUSIONES

A continuación se presentan las conclusiones de nuestro trabajo. Primero, se presentan las conclusiones referentes a los objetivos planteados al inicio del proyecto. En segundo lugar se presentan las conclusiones referentes al cumplimiento del cronograma y finalmente se presentan las conclusiones referentes a las funcionalidades del producto.

7.1 Referentes a los objetivos del proyecto

Uno de los principales objetivos que nos planteamos fue el desarrollo de una aplicación capaz de encontrar genes en una secuencia de ADN procurando tener buenos resultados. Como se mostró en la sección 6.3 los resultados no fueron muy buenos y si bien se logró encontrar alrededor de un 50% de genes; el porcentaje de falsos positivos es muy alto. Por lo tanto creemos que este objetivo se cumplo en forma parcial.

Sin embargo, creemos que tomando en cuenta lo expuesto en el punto 5.4 sobre los casos en los que la aplicación detecta un grupo de ORFs anidados en el mismo marco de lectura, y elige como gen un ORF incorrecto; los resultados desde el punto de vista de un investigador que usará la herramienta serían bastante mejores. Cuantificar la mejora percibida por el usuario de la herramienta, es uno de los posibles trabajos a futuro.

Otro de los objetivos que nos planteamos fue el de evaluar la aplicabilidad de los modelos ocultos de Markov para la búsqueda de genes. Sobre este objetivo, creemos que al menos de la forma en que nosotros los aplicamos, no dan muy buenos resultados. Del estado del arte de los HMM aplicados a este problema, podemos ver que en general son utilizados como parte de una solución más compleja y no como la solución final. Incluso en algunos casos se utilizan variantes más avanzadas como por ejemplo IMM (Interpolated Markov Model) [Burge]. GeneMark [Lukashin], utilizando solamente los HMM, obtiene resultados apenas mejores que los nuestros. La diferencia de la solución de GeneMark, es que resuelve el problema mediante la alineación y no la evaluación.

Sobre la implementación del sistema, uno de los objetivos planteados fue el desarrollo del mismo dividido en dos grandes módulos, por un lado un motor que resuelva los problemas de entrenamiento y evaluación de HMM y por otro lado, otro que se encargue del resto de la lógica necesaria y de la interfaz de usuario. Consideramos que este objetivo se alcanzó ya que se logró desarrollar un motor totalmente independiente del dominio, que implementa los algoritmos de solución de los mencionados problemas de forma rápida y eficiente; y que creemos puede ser reutilizado en otros contextos.

A pesar de lo anterior, hay algunos aspectos como el entrenamiento del vector inicial que puede limitar los dominios en los que el motor pueda ser aplicado por lo que como trabajo a futuro sería bueno extender el motor para que soporte estos casos.

Sobre el relacionamiento con la Facultad de Ciencias, creemos que como primer experiencia puede evaluarse como positiva. Ahora bien, debido a las dificultades que encontramos en la implementación de los algoritmos se dificulto un poco la comunicación con ellos (se trataba de problemas específicos de la aplicación que nada tenían que ver con la biología molecular). El tiempo dedicado a resolver estas dificultades podría haberse utilizado para profundizar en los problemas prácticos de la aplicación junto con los docentes Musto y Romero. También es cierto que el período de receso no colaboró para reestablecer la comunicación.

7.2 Con respecto al cronograma

En lo que respecta al cronograma, el mismo fue cumplido en las primeras instancias del proyecto. Sin embargo, a mediados del proyecto, cuando estábamos implementando los algoritmos, comenzamos a retrasarnos debido a problemas para balancear el uso de memoria con el uso del procesador, a causa de errores de implementación (difíciles de localizar) y finalmente por imprecisiones en los artículos en los que nos estábamos basando para la implementación de los algoritmos, particularmente con la estabilidad numérica.

En general ocurrió que tuvimos que prestarle mucho más atención a la implementación de la aplicación en lenguaje C (el "motor" de HMMs) que lo que habíamos previsto inicialmente. El resto de las actividades que se tenían planificadas fueron cumplidas en el tiempo establecido.

7.3 Respecto a las funcionalidades del producto

Con respecto a las funcionalidades del producto, consideramos que se lograron implementar los requerimientos relevados más importantes. A su vez los que fueron dejados de lado, como por ejemplo, el análisis de la secuencia directa solamente, creemos que no afectan la evaluación de la utilidad del enfoque y la herramienta.

A su vez, y debido a la gran cantidad de datos que se debían analizar para poder evaluar la efectividad de la herramienta, decidimos implementar una nueva funcionalidad que analice los resultados de la herramienta respecto a los genes conocidos asociados al genoma. Por más información al respecto ver 5.4.

8 Anexo I - Modelos ocultos de Markov (HMMs)

8.1 Introducción

En este anexo presentaremos los conceptos básicos que hemos manejado a lo largo del proyecto en lo que respecta a los modelos de Markov ocultos. Para el desarrollo de esta sección nos hemos basado fundamentalmente en el artículo publicado por Rabiner [Rabiner] que realiza una excelente presentación de los modelos de Markov ocultos, de hecho la notación que manejaremos a lo largo de esta sección es similar a la introducida por Rabiner. También nos basamos en el libro "Bioinformatics: The Machine Learning Approach" [Baldi-Brunak], el cual no sólo presenta a los modelos de Markov ocultos, sino también su aplicación en el área de la bioinformática.

8.2 Definición

Un modelo oculto de Markov (Hidden Markov Model, o HMM) es un modelo finito, que describe la distribución de probabilidades sobre una cantidad infinita de posibles secuencias de observaciones. Permiten describir procesos con señales observables, modelando propiedades estadísticas estables de las mismas; haciendo posible la implementación de sistemas de reconocimiento y predicción de las observaciones. Un HMM maneja simultáneamente el estado del sistema, junto con el muestreo (las observaciones) del mismo. El proceso que maneja el estado del modelo es precisamente un proceso de Markov, el cual, por no ser observable recibe el nombre de "oculto".

Actualmente, los HMMs son utilizados para la resolución de problemas en muy diversas áreas. Algunas de ellas son:

- Procesamiento de imágenes.
- Técnicas de clasificación de problemas de visión para soluciones de visión artificial.
- Reconocimiento de voz y de gestos.
- Reconocimiento de formas.
- Clasificación de contenido sonoro.
- El reconocimiento de patrones en cadenas de ADN.

Los modelos ocultos de Markov quedan entonces determinados por los siguientes parámetros:

- N – La cantidad de estados que tiene el modelo. Denotaremos el conjunto de estados del modelo como $S = \{S_1, S_2, \ldots, S_N\}$ y el estado en el tiempo t como q_t.

- M – La cantidad de símbolos (posibles observaciones) que maneja el modelo. Se denotará al conjunto de símbolos del modelo como $V = \{V_1, V_2, \dots, V_M\}$. El conjunto de símbolos también recibe el nombre de alfabeto.

- A – La distribución de probabilidades de transición de un estado a otro. Denotaremos $A = \{a_{ij}\}$, donde $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)$, $1 \leq i, j \leq N$.

- B – La distribución de probabilidades de observar un símbolo en un determinado estado j. Denotaremos $B = \{b_j(k)\}$, donde $b_j(k) = P(V_k \text{ en } t \mid q_t = S_i)$, $1 \leq j \leq N$ y $1 \leq k \leq M$.

- π – La distribución inicial de los estados, donde $\pi = \{\pi i\}$ siendo $\pi i = P(q_1 = S_i)$ para $1 \leq j \leq N$.

Para referirnos a las tres distribuciones de probabilidades mencionadas anteriormente, utilizaremos la siguiente notación compacta: $\lambda = (A, B, \pi)$

Los HMMs pueden ser usados en la simulación cuando en base a los valores de A, B y π se generan observaciones. El siguiente algoritmo muestra como sería este procedimiento:

1. Se elige el estado inicial $q_1 = S_i$ en base a la distribución de estados inicial (π).

2. Se fija t en 1.

3. Se elige $O_k = V_k$ de acuerdo a la distribución de de probabilidades de emisión de un símbolo en el estado S_i ($b_i(k)$).

4. Se transita a un nuevo estado $q_{t+1} = S_j$, en base a la distribución de probabilidades de transitar del estado S_i a S_j (a_{ij}).

5. Si $t < T$ (o sea todavía no se ha llegado al final de la secuencia) Se fija t en $t+1$ y se vuelve al paso 3, en caso contrario se termina el procedimiento.

Si se encuentra entonces un modelo capaz de representar una determinada realidad (en la que quizá sea costoso el proceso de obtener las observaciones), entonces mediante el algoritmo descrito previamente es posible obtener secuencias de observaciones simuladas que son iguales estadísticamente respecto a las secuencias de observaciones reales.

8.3 Los tres problemas básicos de un HMM

Concretamente, son tres los problemas que deben ser resueltos de forma eficiente para poder aplicar los HMMs en situaciones reales.

El primer problema consiste en evaluar la probabilidad de que se produzca una determinada secuencia de observaciones dado el modelo. O sea, para $O = O_1, O_2, \ldots, O_T$ calcular $P(O \mid \lambda)$. De esta manera, si se tiene un HMM que modele una determinada realidad, entonces es posible evaluar la probabilidad de que una secuencia de observaciones se produzca en la realidad (basándose en el modelo). A éste problema nos referiremos como el problema de evaluación de una secuencia.

El segundo problema de interés es el de seleccionar la secuencia de estados que mejor explica una determinada secuencia de observaciones. O sea que dada $O = O_1, O_2, \ldots, O_T$ y λ, se pueda encontrar la secuencia $Q = q_1, q_2, \ldots, q_T$ óptima. Este problema es de interés sobre todo en los casos en los que se desean realizar comparaciones y encontrar patrones entre distintas secuencias de observaciones. Este problema recibe el nombre de problema de alineación.

El tercer problema consiste en encontrar un método que permita ajustar las distribuciones de probabilidades del HMM (λ), de forma tal que si sabemos que una secuencia de observaciones se corresponde a la realidad que modela el HMM, entonces la probabilidad de que se produzca esa secuencia en base al modelo sea maximizada. O sea que se busca ajustar λ para que $P(O \mid \lambda)$ sea maximizada. La secuencia de observaciones conocida es llamada secuencia de entrenamiento, dado que es utilizada para entrenar al modelo. A éste problema nos referiremos como el problema de entrenamiento. Es importante notar la importancia de este problema, dado que optimiza los parámetros de un modelo en base a un conjunto de secuencias de entrenamiento, obteniendo así modelos que se adaptan mejor a los fenómenos reales.

En este documento sólo profundizamos en el primer y el tercer problema (la evaluación y el entrenamiento) dado que fueron los problemas que consideramos deberían resolverse en el marco del proyecto. Por lo tanto, en los siguientes apartados presentaremos las soluciones para éstos, dejando de lado el segundo problema. Por más información acerca de los tres problemas de un HMM véase [Rabiner].

8.4 Problema de evaluación

El problema de evaluación consiste en calcular $P(O \mid \lambda)$. La forma más directa consiste en enumerar todas las posibles secuencias de estados de largo T (siendo T la cantidad de observaciones). Sin embargo, como se muestra tanto en [Rabiner] como en[Baldi-Brunak], el orden de calcular $P(O \mid \lambda)$ de ésta forma es

de alrededor de $2T \cdot NT$ operaciones, lo cual es irrealizable desde un punto de vista computacional[7]. Es por ello que introduce la definición de la variable "*forward*" como la probabilidad de que ocurra una determinada secuencia parcial de observaciones hasta el tiempo t, y se esté en el estado S_i en el tiempo t. O sea:

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i \mid \lambda)$$

La variable $\alpha_t(i)$ se calcula recursivamente de la siguiente forma:

$$\alpha_1(i) = \pi_i b_i(O_1) \quad , \quad 1 \le i \le N$$

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) a_{ji} \right) b_i(O_{t+1}) \quad , \qquad 1 \le t \le T-1 \quad \text{y} \quad 1 \le i \le N$$

El paso base determina el valor de la variable de *forward* en base a la distribución inicial de probabilidad de estar en el estado S_i, y en base a la probabilidad de que se haya emitido el símbolo O_1 en ese estado S_i. En el paso recursivo, se multiplican las probabilidades de pasar de un estado cualquiera en t al estado S_i en $t+1$ (a_{ji}), y los valores de *forward* en t para todos los estados del modelo (si la transición entre el estado S_j y S_i no es posible, entonces $a_{ji} = 0$). La suma de todas esas probabilidades se multiplica por la probabilidad de que en el tiempo $t+1$ se emita el símbolo O_{t+1}.

Como por definición se tiene que:

$$\alpha_T(i) = P(O_1, O_2, ..., O_T, q_t = S_i \mid \lambda)$$

, para calcular $P(O \mid \lambda)$ basta con sumar los $\alpha_T(i)$ para todos los estados del modelo, o sea que:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

El costo computacional de calcular $P(O \mid \lambda)$ de esta forma es notoriamente menor, requiriendo alrededor de $N2T$ operaciones, una cantidad computacionalmente razonable.

8.5 Problema de entrenamiento

El problema de entrenamiento es por lejos el más difícil de los tres. El objetivo es ajustar los parámetros del modelo, de forma tal de maximizar la probabilidad de una secuencia de observaciones. Actualmente no existe una forma analítica capaz de encontrar una solución global para este problema. Debido a

[7]. En [Rabiner] se presentan las fórmulas que calculan $P(O \mid \lambda)$ de ésta manera

esto, hoy se utilizan varios procedimientos que permiten llegar a óptimos locales. Uno de ellos es el procedimiento de Baum-Welch, también llamado EM (por expectation – modification). En [Baldi-Brunak] se presentan otros métodos para el entrenamiento de un modelo, en particular se mencionan las ecuaciones de gradiente descendiente y el "aprendizaje de Viterbi". Nosotros profundizaremos en el presentado por Baum-Welch. Para poder explicar bien este procedimiento definiremos previamente algunas variables.

Así como se definió la variable de *forward*, definimos la variable de "*backward*" como la probabilidad de que se produzca una determinada secuencia parcial de observaciones desde el tiempo $t+1$ hasta el final, dado el estado S_i en el tiempo t, y el modelo λ. O sea:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T \mid q_t = S_i, \lambda)$$

Al igual que la variable de *forward*, la variable de *backward* se calcula de forma inductiva:

$$\beta_T(i) = 1, \qquad 1 \le i \le N$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \qquad t = T-1, T-2, ..., 1 \quad \text{y} \quad 1 \le i \le N$$

El paso base arbitrariamente define la variable de *backward* en 1. El paso recursivo es la sumatoria de las variables de *backward* de todos los estados posteriores para $t+1$, multiplicadas por la probabilidad de pasar del estado S_i a ese estado posterior, por la probabilidad de que además en esa transición se emita el símbolo O_{t+1} en el tiempo $t+1$.

En base a estas variables de *forward* y *backward* podemos definir la probabilidad de estar en el estado S_i en el momento t dada una secuencia de observaciones y un modelo, o sea

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

de las siguiente manera:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}$$

Dado que $\alpha_t(i)$ calcula la probabilidad de la secuencia parcial hasta el tiempo t en el estado S_i, mientras que $\beta_t(i)$ considera la probabilidad del resto de

la secuencia de observaciones hasta el final de la misma, partiendo en t, del estado S_i. Finalmente, $P(O|\lambda)$ es el factor de normalización para que $\gamma_t(i)$ sea una medida de probabilidad de tal manera de que

$$\sum_{i=1}^{N} \gamma_t(i) = 1$$

De forma similar podemos definir la probabilidad de estar en el estado S^i en el momento t y en el estado S^j en el momento $t+1$ dada una secuencia de observaciones y un modelo, o sea:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)$$

que se calcula de la siguiente manera:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}$$

El razonamiento para obtener esta fórmula es muy similar al de la variable $\gamma_t(i)$, la figura 12 muestra lo dicho.

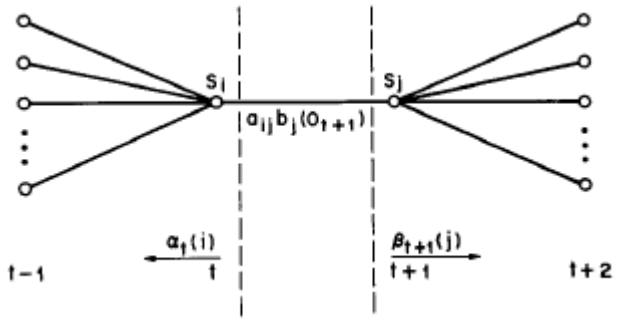


Figura 13: Ilustración de la secuencia de operaciones requeridas para calcular $\xi_t(i,j)$ [Rabiner].

Al igual que para el caso de la variable $\gamma_t(i)$, $P(O|\lambda)$ $P(O|\lambda)$ aparece como factor de normalización a fin de que $\xi_t(i,j)$ sea una medida de probabilidad.

Nótese además que, por la misma definición de $\gamma_t(i)$ y de $\xi_t(i,j)$, se tiene que

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \ ,$$

dado que la probabilidad de que en t se esté en un determinado estado es la suma de las probabilidades de que en t se esté "saliendo" de ese estado.

Si se suma $\gamma_t(i)$ para todos los t, entonces se tendrá una estimación de la cantidad de veces a lo largo del tiempo en que el estado S_i es visitado. De la misma manera, si se suman los $\xi_t(i,j)$ para todos los t, se tiene una estimación de la cantidad de transiciones que se producen del estado S_i al estado S_j a lo largo del tiempo.

Ahora bien, con esta información estamos en condiciones de presentar el algoritmo de entrenamiento. Sea $\lambda = (A, B, \pi)$, el modelo que se desea "entrenar", y $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$, el modelo "entrenado". Entonces, una forma de realizar ese entrenamiento es mediante las siguientes fórmulas:

$\overline{\pi}_i$ = frecuencia esperada (cantidad de veces) en el estado S_i cuando t es 1, o sea

$$\overline{\pi}_i = \gamma_t(i)$$

$\overline{a}_{ij}$ = la cantidad esperada de transiciones desde el estado S_i al estado S_j, dividido la cantidad esperada de transiciones desde el estado S_i, o sea

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$\overline{b}_j(k)$ = la cantidad esperada de veces en el estado S_j, y que se observe el símbolo V_k, dividido la cantidad esperada de veces en el estado S_j, o sea

$$\overline{b}_j(k) = \frac{\sum_{t=1 \ s.a. \ O_t = V_k}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Se ha demostrado que, o bien λ es un punto crítico de la función de probabilidad, y entonces $\lambda = \overline{\lambda}$, o bien el modelo $\overline{\lambda}$ es más probable (mejor) que el modelo λ, en el sentido de que $P(O \mid \overline{\lambda}) > P(O \mid \lambda)$. O sea que se encontró un nuevo modelo que es mejor.

Si se repite este proceso de reestimación de los parámetros, sustituyendo a λ por $\overline{\lambda}$, entonces, podremos mejorar la probabilidad de que la secuencia O sea observada (a partir del modelo), hasta que se alcance un determinado punto crítico en el que no hay más "mejoras".

En aquellos casos en los que $\pi 1 = 1$ y $\pi i = 0$ para todo $i > 1$ entonces la reestimación de la distribución de las probabilidades no sería necesaria. De hecho, nosotros no ajustamos el vector de probabilidades iniciales dado que, para la resolución de nuestro problema, siempre asumimos que teníamos un único estado inicial.

Un dato importante a considerar es que las restricciones de los parámetros del HMM se verifican en todas las iteraciones, o sea que:

$$\sum_{i=1}^{N} \overline{\pi}_i = 1$$

$$\sum_{i=1}^{N} \overline{a}_{ij} = 1, \qquad 1 \le i \le N$$

$$\sum_{k=1}^{M} \overline{b}_j(k) = 1, \qquad 1 \le j \le N$$

8.6 Cuestiones propias de la Implementación

Al momento de implementar los HMMs surgen algunas dificultades prácticas producto de las limitaciones del mundo real que se desea modelar, e inclusive de la capacidad de cálculo y la precisión de las actuales computadoras. En [Rabiner] se mencionan algunos de éstos problemas prácticos, a continuación, presentaremos aquellos que fueron relevantes para este proyecto.

8.6.1 Escalado

El procedimiento de escalado[8] consiste en multiplicar a la variable de *forward* y de *backward* por un coeficiente que sea dependiente solamente de t (o sea, independiente de i), de tal forma de que los valores de estas variables se mantengan dentro del rango de precisión de la máquina.

[8] En [Rabiner] el procedimiento de escalado no es abordado con demasiada profundidad, adoleciendo de algunos detalles que conducen a una implementación incorrecta del escalado. Luego de "sufrir" un poco con estos problemas encontramos en [Rahimi], la corrección y aclaración de los conceptos. Es por ello que optamos por presentar con detalles el procedimiento de escalado.

Sea α la variable de *forward* sin escalado. Sea también $\hat{\alpha}$ la variable de *forward* escalada y finalmente $\hat{\hat{\alpha}}$ la variable de *forward* que se calcula en cada paso (temporal) previo al escalado. [9]

Lo que se busca entonces es una recursión para calcular $\hat{\alpha}$ de tal manera que se cumpla que:

$$\hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} = C_t \alpha_t(i) \; \hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} = C_t \alpha_t(i) \qquad (1)$$

La siguiente es la recursión:

$$\hat{\hat{\alpha}}_1(i) = \alpha_1(i) \; \hat{\hat{\alpha}}_1(i) = \alpha_1(i)$$

$$\hat{\hat{\alpha}}_t(i) = \left(\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} \right) b_i(O_t)$$

$$c_t = \frac{1}{\sum_{i=1}^{N} \hat{\hat{\alpha}}_t(i)}$$

$$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i)$$

Por inducción se demuestra que la recursión anterior cumple con la condición (1).

Paso Base:

$$\hat{\hat{\alpha}}_1(i) = \alpha_1(i), \qquad \hat{\alpha}_1(i) = \frac{\alpha_1(i)}{\sum_{i=1}^{N} \alpha_1(i)}$$

que cumple la condición (1) con $C_1 = c_1$

Paso Inductivo: si $\hat{\alpha}_{t-1}(i) = C_t \alpha_t(i)$, entonces tenemos que

$$\hat{\alpha}_t(i) = C_{t-1} \left(\sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji} \right) b_i(O_t) = C_{t-1} \alpha_t(i)$$

[9] La diferencia entre α y $\hat{\hat{\alpha}}$ es la siguiente: mientras que $\hat{\hat{\alpha}}$ es multiplicada por un coeficiente de escalado en cada paso de la recursión que la calcula, excepto en el último; α es la variable de *forward* original que no se multiplica por ningún coeficiente de escalado en toda la recursión que la calcula.

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} = \frac{1}{C_{t-1} \sum_{i=1}^{N} \alpha_t(i)}$$

$$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i) = \frac{C_{t-1}\alpha_t(i)}{C_{t-1} \sum_{i=1}^{N} \alpha_t(i)} = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)}$$

que es lo que se quería demostrar.

Como consecuencia del razonamiento anterior se pueden deducir las siguientes expresiones:

$$C_t = \frac{1}{c_{t+1} \sum_{i=1}^{N} \alpha_{t+1}(i)} = \frac{C_{t+1}}{c_{t+1}}$$

$$C_t = C_{t-1} c_t = \prod_{r=1}^{t} c_r$$

Además se define Dt que será utilizado para escalar β, y se mostrará su relación con Ct.

$$D_t = \prod_{r=t}^{T} c_r$$

$$C_t D_{t+1} = \prod_{r=1}^{t} c_r \prod_{r=t+1}^{T} c_r = \prod_{r=1}^{T} c_r = C_T$$

El escalado de la variable de *backward* ($\hat{\beta}$) lo definimos como:

$$\hat{\beta}_t(i) = D_t \beta_t(i) \qquad (2)$$

La siguiente recursión produce los valores deseados:

$$\hat{\hat{\beta}}_T(i) = \beta_T(i)$$

$$\hat{\hat{\beta}}_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)$$

$$\hat{\beta}_t(i) = c_t \hat{\hat{\beta}}_t(i)$$

donde c_t es el mismo coeficiente que el utilizado para el escalado de la variable de *forward*. Nótese además que definir $\hat{\beta}_t(i) = D_t \beta_t(i)$ no es lo mismo que imponer que

$$\hat{\beta}_t(i) = \frac{\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \qquad \rightarrow \quad \text{¡No se cumple!}$$

Al igual que para el caso de la variable de *forward*, la demostración de (2) se hará inductivamente:

Paso Base:

$$\hat{\hat{\beta}}_T(i) = \beta_T(i), \qquad \hat{\beta}_T(i) = D_T \beta_T$$

que satisface la condición de escalado de la variable de *backward* con $D_T = c_T$

Paso Inductivo: Si $\hat{\beta}_{t+1}(i) = D_{t+1}\beta_{t+1}$, entonces tenemos que

$$\hat{\hat{\beta}}_t(i) = D_{t+1} \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) = D_{t+1} \beta_t(i)$$

$$\hat{\beta}_t(i) = c_t \hat{\hat{\beta}}_t(i) = c_t D_{t+1} \beta_t = D_t \beta_t$$

que es lo que se quería demostrar.

Ahora bien, hemos mostrado como calcular los valores escalados de la variable de *forward* y de *backward*, ahora mostraremos como afectan estas modificaciones las fórmulas y procedimientos para resolver tanto el problema de entrenamiento así como también el problema de evaluación.

Problema de evaluación: en este caso, hay que evaluar el impacto de utilizar la variable de *forward* escalada en lugar de la variable de *forward* común. Por supuesto ya no es posible sumar los valores de $\hat{\alpha}$ por estar escalada, por lo que para lograr calcular la probabilidad se hace necesario buscar otro mecanismo, para ello utilizaremos la siguiente propiedad:

$$\sum_{i=1}^{N} \hat{\alpha}_T(i) = \sum_{i=1}^{N} \left(\frac{\hat{\hat{\alpha}}_T(i)}{\sum_{j=1}^{N} \hat{\hat{\alpha}}_T(j)} \right) = \frac{\sum_{i=1}^{N} \hat{\hat{\alpha}}_T(i)}{\sum_{j=1}^{N} \hat{\hat{\alpha}}_T(j)} = 1$$

por lo tanto, y como sabemos por (1) que $\hat{\alpha}_T(i) = C_T \alpha_T(i)$, tenemos que

$$\sum_{i=1}^{N} \hat{\alpha}_T(i) = \sum_{i=1}^{N} C_T \alpha_T(i) = C_T \sum_{i=1}^{N} \alpha_T(i) = C_T \cdot P(O \mid \lambda) = 1$$

y por lo tanto que

$$\prod_{t=1}^{T} c_t \cdot P(O \mid \lambda) = 1$$

así, podemos despejar $P(O \mid \lambda)$ de tal forma que nos queda que

$$P(O \mid \lambda) = \frac{1}{\prod_{t=1}^{T} c_t}$$

Ahora bien, si se multiplicasen todos los c_t excederíamos el rango de precisión de la máquina, por lo tanto $P(O \mid \lambda)$ no puede ser calculado. Lo que si se puede hacerse es calcular el logaritmo de la probabilidad (o sea $\log[P(O \mid \lambda)]$) de la siguiente forma:

$$\log[P(O \mid \lambda)] = -\sum_{t=1}^{T} \log(c_t)$$

Si bien el logaritmo de la probabilidad no es algo tan claro ni intuitivo de manejar como la probabilidad misma, es la única forma de considerar probabilidades muy pequeñas, y sirve de *score* para determinar qué tan bien se "adapta" la observación al modelo dado.

Problema de entrenamiento: para evaluar el impacto del escalado de las variables de *forward* y *backward* en el entrenamiento, basta con calcular los valores de las variables γ y ξ en función ahora de las variables $\hat{\alpha}$ y $\hat{\beta}$.

Por definición de ξ teníamos que:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$

Ahora bien, si sustituimos los valores de *forward* y *backward* por los escalados correspondientes tenemos que

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{C_t D_{t+1} P(O \mid \lambda)}$$

Pero como $C_t D_{t+1} = C_T$, y sabemos que $C_T \cdot P(O \mid \lambda) = 1$, entonces tenemos que

$$\xi_t(i,j) = \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)$$

$$\xi_t(i,j) = \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)$$

lo cual es una forma muy sencilla de calcular ξ en función de las variables de *forward* y *backward* escaladas.

Análogamente en la definición de γ teníamos que:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{P(O \mid \lambda)}$$

Sustituyendo por las variables escaladas tenemos:

$$\gamma_t(i) = \frac{\hat{\alpha}_t(i) \hat{\beta}_t(i)}{C_t D_t P(O \mid \lambda)} = \frac{\hat{\alpha}_t(i) \hat{\beta}_t(i)}{C_t D_{t+1} c_t P(O \mid \lambda)}$$

Nuevamente, como $C_t D_{t+1} = C_T$, y sabemos que $C_T \cdot P(O \mid \lambda) = 1$, entonces

$$\gamma_t(i) = \frac{\hat{\alpha}_t(i) \hat{\beta}_t(i)}{c_t}$$

Por lo tanto, las fórmulas para el cálculo de $\overline{a}_{ij}$ y $\overline{b}_i(k)$ usando el escalado, quedan de la siguiente forma[10]:

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) \hat{\beta}_1(j) \frac{1}{c_t}}$$

[10] Si se comparan las fórmulas de este documento con las que se introducen en [Rabiner] se encontrará que el parámetro $1/c_t$ no está en ninguna de las fórmulas de reestimación de a_{ij} o $b_i(O_t k)$ en [Rabiner]. Allí se afirma erróneamente que todos esos coeficientes se anulan, cuando en realidad esto no es así. Nosotros en particular, utilizando las fórmulas tal cual las propone [Rabiner] no logramos cumplir con las restricciones del modelo para todas las iteraciones del entrenamiento.

$$\bar{b}_i(k) = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i)\hat{\beta}_1(j)\frac{1}{c_t}}{\sum_{t=1}^{sa\ O_t = V_k} \sum_{t=1}^{T-1} \hat{\alpha}_t(i)\hat{\beta}_1(j)\frac{1}{c_t}}$$

8.6.2 Múltiples secuencias de observaciones

Muchas veces no es suficiente tener una única secuencia de observaciones para realizar el entrenamiento del modelo. Algunos modelos pueden tener restricciones que hacen que luego de que se salga de un determinado estado del mismo, no se pueda volver a él. Lo mismo podría ocurrir con conjuntos de estados o regiones del modelo, o sea, que luego de salir de las mismas, no haya posibilidad de volver a ellas. Para estos casos una única secuencia de observaciones no es suficiente para obtener estimaciones confiables de todos los parámetros del modelo, por lo que se hace necesario considerar múltiples secuencias de observaciones para entrenar el modelo.

Creímos conveniente considerar múltiples secuencias de observaciones para este proyecto a fin de no restringir el entrenamiento a una única secuencia, la cual podría ser los suficientemente "peculiar" (conociendo el dominio de la aplicación encontramos que siempre existen casos excepcionales) como para hacer un ajuste de los parámetros que no se corresponde con la generalidad de los casos en la realidad del dominio. Además, uno de los objetivos planteados es el de dar la libertad de trabajar con cualquier tipo de modelo, y no solamente con aquellos que con una única observación son capaces de ajustar correctamente los parámetros[11].

Consideremos un conjunto de K secuencias de observaciones como:

$$O = \left\{ O^{(1)}, O^{(2)}, \ldots, O^{(K)} \right\}$$

donde cada secuencia $O(k)$ es de la siguiente forma:

$$O^{(k)} = \left\{ O_1^{(k)}, O_2^{(k)}, \ldots, O_{T_k}^{(k)} \right\}$$

Se asume que cada secuencia de observaciones es independiente de cualquier otra secuencia. Nuestro objetivo entonces es el de ajustar los parámetros del modelo de tal forma de maximizar

$$P(O \mid \lambda) = \prod_{k=1}^{K} P(O^{(k)} \mid \lambda)$$

Notaremos $P_k = P(O^{(k)} \mid \lambda)$.

[11] Estos modelos son aquellos que cumplen que para todo estado, la probabilidad de volver a ese estado (independientemente del tiempo transcurrido) siempre sea mayor que 0.

Dado que las fórmulas de reestimación de los parámetros están basadas en frecuencias de ocurrencia de varios eventos, las fórmulas de reestimación para múltiples secuencias de observaciones simplemente consisten en agregarles la frecuencia individual con la que ocurre cada secuencia. Por lo tanto, éstas son las fórmulas para el cálculo de $\overline{a}_{ij}$ y $\overline{b}_i(h)$ considerando múltiples observaciones:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)}$$

$$\overline{b}_i(h) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \quad sa\ O_t = V_h$$

Agregar el escalado cuando se trabaja con múltiples observaciones es directo dado que cada secuencia de observaciones cuenta con su propio coeficiente de escalado. Así, las fórmulas que consideran múltiples secuencias de observaciones y a la vez el escalado para el cálculo de $\overline{a}_{ij}$ y $\overline{b}_i(h)$ son:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i) \frac{1}{c_t^k}}$$

$$\overline{b}_i(h) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i) \frac{1}{c_t^k}}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i) \frac{1}{c_t^k}} \quad sa\ O_t = V_h$$

Para el caso particular nuestro, cada gen se manifiesta una única vez en una secuencia de ADN, por lo tanto la probabilidad de que se dé esa secuencia de observaciones es la inversa de la cantidad de genes. Ahora bien, ni entrenamos con la totalidad de los genes (no tendría sentido el problema), ni conocemos a priori la cantidad de genes, por lo tanto utilizamos P_k como un peso relativo a la cantidad de secuencias que son usadas en el entrenamiento. De esta manera, aquellos genes de los cuales se sepa que su secuencia de bases tiene una constitución común o normal para determinada secuencia de ADN, se les puede asignar un valor mayor de P_k que al resto, mientras que aquellos genes que sabe

son demasiados peculiares, se les puede asignar un peso menor. Como nosotros no sabemos a priori cuando un gen es "normal" o "peculiar", optamos por asignarles el mismo peso a todas las secuencias del entrenamiento, o sea que hicimos que $P_k=1/K$.

9 Anexo II - Desarrollo del Trabajo

En este apartado nos enfocaremos en como fue el desarrollo del trabajo a lo largo de todo el proyecto. El cronograma de actividades elaborado inicialmente para encarar este proyecto fue el siguiente:

Mayo – Junio:	Introducción al problema y búsqueda de información relacionada al problema (soluciones existentes), estudio de modelos de Markov ocultos y estudio del dominio de la aplicación.
Junio – Julio:	Diseño de la aplicación. Selección de las herramientas de desarrollo a utilizar.
Agosto – Septiembre – Octubre:	Implementación del motor de HMMs. "Curso" de biología molecular a cargo de docentes de la facultad de Ciencias respecto al Dominio de la aplicación.
Noviembre – Diciembre	Implementación de la aplicación que realiza la búsqueda de genes.
Enero:	Para la primera mitad de enero no había nada planificado. En la segunda mitad, continuar con la implementación de la aplicación.
Febrero:	Fin de la implementación. Verificación y Validación de la aplicación. Evaluación de los resultados obtenidos.

La idea inicial fue trabajar sobre el informe final a lo largo de todo el desarrollo.

9.1 Detalle del cronograma

9.1.1 Mayo – Junio

Este período abarca en realidad sólo la primera quincena de Junio. El objetivo propuesto era el de introducirnos en el problema que debíamos resolver, para lo que fue necesario investigar acerca de los HMM (hasta ese entonces no teníamos ningún conocimiento al respecto) y estudiar el dominio de la aplicación.

En lo que respecta al dominio de la aplicación, recibimos abundante material por parte de los docentes de la facultad de Ciencias. Al principio nos vimos abrumados por la gran cantidad de libros, los cuales además no daban una visión muy general de la realidad, sino que profundizaban en muchos aspectos los cuales eran totalmente intrascendentes para el problema que debíamos resolver. Más adelante, cuando tuvimos las clases con Héctor Musto y Héctor Romero (docentes de la Facultad de Ciencias) pudimos tener una idea más acabada de qué era lo que se debía hacer.

En cuanto al trabajo relacionado, analizamos las propuestas de otros sistemas que trabajaban con problemas relacionados a la biología molecular y que utilizaran los HMM. Sin embargo la gran mayoría de las aplicaciones resolvía problemas de alineamiento de genes (para determinar si dos genes son similares) y no de búsqueda de genes.

Para conocer mejor los HMM estudiamos el artículo publicado por Lawrence Rabiner [Rabiner], el cual presenta una excelente descripción y explicación de los modelos de Markov ocultos y los problemas asociados a los mismos. También miramos algunos libros que trataban este tema, en particular el "Bioinformatics: A Machine Learning Approach" [Baldi-Brunak], sin embargo terminaríamos basando nuestro trabajo en el artículo de Rabiner.

Fue en éste período que determinamos cuál sería el mecanismo que usaríamos para la detección de genes. Lo que se nos ocurrió fue entrenar el modelo con secuencias de observaciones que fueran genes (resolver el problema de entrenamiento de un HMM), y una vez que se tuviese el modelo entrenado evaluar la probabilidad de cada ORF presente en la secuencia de ADN (resolver el problema de evaluación). Mirando un poco en retrospectiva, creemos que esta decisión fue un poco apresurada y que debimos haber profundizado más en el estudio de los trabajos relacionados y de la realidad del dominio. Luego de haber elegido un enfoque determinado, encontramos una aplicación que resuelve este problema con HMMs pero con un enfoque distinto. Como nos pareció interesante, lo agregamos a este documento como un trabajo relacionado.

9.1.2 Junio – Julio

Éste período abarcó la segunda quincena de Junio y el mes de Julio. Al comienzo del mismo se analizaron las herramientas que se utilizarían para desarrollar la aplicación. Se hicieron implementaciones sencillas del algoritmo de *forward* en C y en C#. Se hicieron comparaciones en lo que respecta a la eficiencia de ambas implementaciones encontrándose que la versión en C era notoriamente más eficiente que la versión en C#. Como consideramos importante que la aplicación fuera performante (siempre supimos que la aplicación sería intensiva en procesamiento), se decidió que los algoritmos que resuelven los problemas de entrenamiento y evaluación de los HMM fueran implementados en C, y que luego

el resto de la aplicación se programase en otro lenguaje de más alto nivel como ser C# o Java.

Continuamos investigando en trabajos relacionados, pero lamentablemente en la mayoría de los casos sólo encontrábamos los resultados obtenidos y no los detalles de la solución aplicada para obtenerlos.

Manejamos distintas alternativas en lo que refería a la interfaz de usuario. Inicialmente habíamos considerado implementar un editor gráfico de modelos, idea que tuvimos que desechar más adelante en el proyecto dado los problemas que encontramos en la implementación de los algoritmos. También evaluamos las distintas alternativas para interactuar con la aplicación en C (el motor de HMMs) desde un lenguaje de más alto nivel.

Se trabajó sobre en el diseño de la aplicación y además se tuvieron los primeros contactos con los docentes de la Facultad de Ciencias Héctor Musto y Héctor Romero.

Finalmente, en este período fue cuando encontramos una librería implementada en Java llamada BioJava, que resolvía varios problemas relacionados con la bioinformática, por lo que determinamos que el lenguaje en alto nivel que utilizaríamos sería precisamente Java a fin de utilizar esta librería.

9.1.3 Agosto – Septiembre – Octubre:

En el transcurso de este trimestre nos abocamos a la implementación de la aplicación en C que resolviera los algoritmos de entrenamiento y evaluación de HMMs. Esto incluyó no sólo los algoritmos propiamente sino también los módulos encargados de la entrada y salida de esta aplicación (para entonces ya habíamos optado por comunicarnos con la aplicación en Java mediante el uso de la entrada y salida estándar). También se implementó un módulo para el manejo de hilos (*threads*) para utilizarlos a la hora de realizar la evaluación. Sin embargo, éste módulo no logramos hacerlo funcionar bien (con más de un hilo teníamos problemas) y dado que teníamos otros problemas más importantes que resolver, no se volvió a revisar la implementación del módulo de hilos. También se le agregó al algoritmo de entrenamiento, el manejo de múltiples secuencias de observaciones.

Inicialmente nos encontramos con problemas de memoria al querer ejecutar los algoritmos con secuencias y modelos similares a los que la aplicación debería manejar. Esto llevó a que la implementación de algunos algoritmos (sobre todo el algoritmo de entrenamiento) tuviera que ser implementado nuevamente de tal forma de que utilizara menos memoria. En este período de tiempo no logramos encontrar el equilibrio entre el uso de memoria (para almacenar resultados reutilizables) y el uso de procesamiento (recalculando los resultados cada vez que fuera necesario), dado que o bien consumíamos toda la memoria de la

computadora, o el algoritmo entrenamiento demoraba demasiado tiempo en ejecutarse.

Durante este período también procuramos validar los resultados que estábamos teniendo con el algoritmo de entrenamiento, comparándolos con resultados obtenidos con otros programas (aunque en realidad usaban métodos similares a los nuestros y no exactamente iguales), y también con otra implementación hecha por nosotros pero en un lenguaje de más alto nivel, más precisamente MATLAB [MATLAB].

En este período (mas bien los dos primeros meses), tuvimos un curso "acelerado" de biología molecular el cual fue dictado por docentes de la Facultad de Ciencias [Musto], y que nos permitió tener una más clara comprensión del dominio de la aplicación. Durante el transcurso del curso, también logramos despejar varias dudas que teníamos en cuanto a los requerimientos de la aplicación. De hecho fue en estas clases que tomamos la decisión de que el módulo en C que estábamos programando, fuera lo suficientemente independiente del problema de la búsqueda de genes, a fin de poder utilizarlo en otros dominios.

Continuamos trabajando en el documento de requerimientos y de los diagramas de casos de uso alentados en base a las charlas que estábamos teniendo con los docentes de la Facultad de Ciencias. También hicimos un resumen de los conceptos que aprendimos en las clases de biología molecular.

Al finalizar este período, no habíamos logrado tener una versión estable y funcional del motor de ejecución de los algoritmos del HMM. Básicamente nos quedaban tres grandes problemas para resolver. El primero de ellos era el de implementar el entrenamiento de forma de balancear la utilización de memoria con la utilización de CPU. El segundo era el de incorporar el escalado al entrenamiento dado que para secuencias de observaciones relativamente cortas (alrededor de 500 bases), los resultados excedían la precisión de la computadora. Finalmente, faltaba validar el entrenamiento dado que aun para ejemplos chicos estábamos teniendo algunos comportamientos extraños (luego nos dimos cuenta que esto se debía a errores de programación).

9.1.4 Noviembre – Diciembre

Este período abarcó los últimos dos meses del año (aunque después del 25 de Diciembre no hicimos nada). La idea inicial era la de estar implementando ya la parte de la aplicación que resuelve propiamente el problema de encontrar genes. Sin embargo, debido a los problemas que teníamos en la implementación de la aplicación en C, nos enfocamos a solucionar los problemas que esta presentaba. De los tres problemas que mencionamos en el punto anterior, nos enfocamos con más detalle en el primero, dado que consideramos que el incorporar el escalado al algoritmo de entrenamiento iba a ser una cuestión sencilla una vez que éste último

estuviera funcionando correctamente[12]. Fue en este momento en el que se implementó un módulo al que llamamos "DataProvider", el cual se encargó de balancear el uso de memoria con el procesamiento, almacenando parte de las variables de *forward* y de *backward* de cada secuencia de observaciones y recalculando las demás. Además se rescribió el algoritmo de entrenamiento procurando maximizar la reutilización de estos "DataProviders" (en particular para usarlos una única vez por cada secuencia de observaciones).

Hicimos una revisión de nuestro código buscando errores que hacían que, aun para secuencias pequeñas, el entrenamiento no funcionara correctamente. Inclusive se reimplementó el entrenamiento de una forma más ineficiente en término de eficiencia de procesamiento, pero con un código más sencillo y más fácil de analizar. De esta manera logramos identificar y corregir varios errores de programación que habíamos tenido.

Para verificar que efectivamente se estaba realizando bien el entrenamiento, controlábamos que las restricciones de integridad del modelo entrenado se cumpliesen (aquello de que las filas de las matriz de estados y la de emisión sumasen 1). Para fines de diciembre estábamos teniendo errores de 10^{-3} (en los casos que podíamos probar) los cuales supusimos se irían una vez introducido el entrenamiento.

Durante este período comenzamos el desarrollo de la aplicación en Java, a fin de no atrasarnos demasiado en el cronograma. Inicialmente, la aplicación en Java (usando la librería BioJava) resolvía la traducción de archivos en formato FASTA[13] al formato que requería la aplicación en C, para luego invocar a esta última. Finalmente el programa en Java capturaba la salida de la aplicación en C y la desplegaba.

9.1.5 Enero

En enero procuramos finalizar la implementación del módulo en C a fin de dedicarnos de lleno a la aplicación final que debíamos construir. A nuestro juicio en ese entonces, lo único que nos faltaba en la aplicación en C era agregar el escalado para poder trabajar con secuencias de observaciones grandes. Sin embargo, una vez introducido el escalado, el entrenamiento daba resultados erráticos de tal forma que todas las entradas de la matriz de emisión quedaban en 0, violando así las restricciones de integridad del modelo. Luego de inspeccionar con mucho detenimiento el código sin encontrar errores, optamos por analizar las fórmulas utilizadas.

[12] En [Rabiner] se afirma que para utilizar el escalado en el algoritmo de entrenamiento, basta con utilizar las variables de *forward* y *backward* escaladas, cuya implementación siempre mantuvimos por separado del resto del algoritmo de entrenamiento, previendo precisamente este hecho. Luego nos daríamos cuenta de que esa afirmación que aparece en [Rabiner] no es correcta.

[13] Es un formato en el que se especifican las secuencias de ADN, así como también el conjunto de genes de una secuencia. Es un archivo de texto plano, con cierta sintaxis particular, y luego la constitución del genoma en su totalidad, o de cada uno los genes.

Paralelamente a este análisis, se prosiguió con la implementación de la aplicación en Java. Se trabajó sobre todo en la arquitectura de la misma, reordenando el código para facilitar luego el agregar un interfaz gráfica a la aplicación para hacerla más amigable.

9.1.6 Febrero

Al comienzo de este mes fue cuando encontramos la errata de Rahimi sobre el artículo de Rabiner en lo que respecta al escalado, y en base a la misma, logramos corregir nuestra implementación del entrenamiento de tal forma de no salirse del rango de precisión de la máquina. El tiempo en el que estuvimos buscando errores en nuestro código significó una gran pérdida de tiempo, lo cual aumentó nuestro retraso.

Una vez implementado el escalado, pudimos verificar el funcionamiento del algoritmo para secuencias más largas. En general estábamos obteniendo que las restricciones se cumplían con un error de apenas 10^{-15}, un valor muy cercano al épsilon de la máquina, por lo que lo desestimamos. En algunos casos estábamos obteniendo errores más grandes, pero esto se debía a errores de implementación en la lectura de las secuencias. Al corregirlos, finalmente logramos que en todos los casos, el error en las restricciones fuera siempre menor a 10^{-15}.

Hasta ese momento, una de las cosas que estábamos haciendo era verificar que los valores de las probabilidades de un conjunto de secuencias para un modelo entrenado por ese conjunto fueran mayores que las probabilidades de ese mismo conjunto pero evaluado con el modelo sin entrenar. Hasta ese momento, haciendo esa prueba, solamente el 50% de las probabilidades mejoraba, mientras que el restante 50% empeoraba. Pero una vez que logramos que el error de las restricciones siempre fuera menor que 10^{-15}, entonces los resultados fueron muchos más alentadores, teniendo que más del 85% de las probabilidades mejoraban. En base a este resultado, y a que las restricciones se cumplían (salvo por esa despreciable diferencia de 10^{-15}) dimos por finalizada la implementación del motor de ejecución de los algoritmos de entrenamiento y evaluación de HMMs.

El restante tiempo en el mes de Febrero lo dedicamos por entero a terminar la implementación de la aplicación en Java, y comenzar a evaluar los resultados que estábamos obteniendo al trabajar con un genoma y con genes verdaderos (en particular se utilizó el genoma de *Escherichia coli K12*: NC_000913). Modificamos la forma de evaluar el *score* de una secuencia, haciendo que el mismo dependiera del largo de la secuencia (esto ya lo habíamos hablado con los docentes de la Facultad de Ciencias varios meses atrás), obteniendo inicialmente resultados poco alentadores (si bien encontrábamos casi un 50% de los genes, estábamos dando una cantidad enorme de falsos positivos, aproximadamente 9 de cada 10 resultados que dábamos, eran falsos positivos).

En este mes retomamos la comunicación con los docentes de la Facultad de Ciencias, mostrándoles los resultados obtenidos, y junto con ellos buscamos alternativas para mejorar los mismos.

9.1.7 Marzo

En Marzo teníamos pensado tener finalizado el proyecto (por lo menos estar presentándolo), sin embargo el atraso que experimentamos durante la implementación del motor de HMM nos obligó a extender el plazo de finalización.

Durante este mes se terminó de implementar la interfaz gráfica de la aplicación (el editor de modelos fue descartado), además se trabajaron en mecanismos de pos procesamiento que permitieran mejorar los resultados que estábamos obteniendo. Algunos de los pos procesamientos funcionaron y otros no. Lo mejor que logramos fue que se detectara alrededor del 50% de los genes y al mismo tiempo se redujeran los falsos positivos de tal forma de que aproximadamente 1 gen de cada 2 que encontrábamos era un falso positivo.

Finalmente en Marzo fue que retomamos la redacción de la documentación, dado que a causa de los atrasos, habíamos dejado de lado la misma para concentrarnos en la resolución de los problemas que estábamos teniendo.

10 Referencias

- [Baldi-Brunak] Pierre Baldi and Søren Brunak (2001). Bioinformatics: The Machine Learning Approach (second edition): Massachusetts Institute of Technology.

- [BioJava] http://www.biojava.org/ 12 de Mayo de 2005.

- [Burge] Christopher B. Burge and Samuel Karlin (1998). Finding the genes in genomic DNA. Current Opinion in Structural Biology 8:346–354 -.

- [C] Brian W. Kernighan y Dennis M. Ritchie (1988). The C Programming Language, Second Edition: Prentice Hall, Inc.

- [C#] Tom Archer (2001). Inside C#: Microsoft Press.

- [EducarChile] http://www.educarchile.cl/autoaprendizaje/biologia/modulo2/clase1/texto/proteinas.htm 12 de Mayo de 2005.

- [Encarta] Enciclopedia Microsoft Encarta 2004.

- [FASTA] FASTA Format Description - http://ngfnblast.gbf.de/docs/fasta.html 12 de Mayo de 2005.

- [Musto] Héctor Musto y Héctor Romero (2004). Curso: "Introducción a la biología molecular".

- [Java] James Gosling, Bill Joy, Guy Steele y Gilad Bracha (2005). The Java Language Specification, Third Edition: ADDISON-WESLEY.

- [Lukashin] Alexander V. Lukashin y Mark Borodovsky (1998). GeneMark.hmm: new solutions for gene finding. Nucleic Acids Research, Vol. 26, No. 4.

- [MATLAB] MATLAB: The Language of Technical Computing. - http://www.mathworks.com/access/helpdesk/help/techdoc/ 12 de Mayo de 2005.

- [PUBMED] http://www.pubmed.org/ 12 de Mayo de 2005.

- [Rabiner] Lawrence R. Rabiner (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition - Proceedings of the IEEE, Vol. 77, No. 2.

- [Rahimi] Ali Rahimi (2000). An Erratum for "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition" –

http://xenia.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html 12 de Mayo de 2005.

- [UML] OMG Unified Modeling Language Specification. Version 1.5 (2003) http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf 12 de Mayo de 2005.

- [Watson] Watson, Baker, Bell, Gann, Levine y Losick (2004). Molecular Biology of the Gene (fifth edition): Pearson Benjamin Cummings.