Universidad Nacional Autónoma de México
Posgrado en Ciencia e Ingeniería de la Computación

Algoritmos para el diseño de oligonucleótidos de interferencia

T E S I S

que para obtener el grado de Maestra en Ciencia e Ingeniería de la Computación

Presenta:

Janeth De Anda Gil

Tutor:

Ricardo Strausz Santiago

Ciudad Universitaria, México, Septiembre 2017

Índice general

Agradecimientos

A mis padres Josefina y Jaime, a mis hermanos Jessica y Jonathan y a mi tío Tito, porque son la fuente de inspiración más grande que tengo en mi vida y me han llenado de fortaleza en los momentos más difíciles para seguir luchando contra todos los obstáculos. Por todo el conocimiento y experiencias personales que me han compartido, las cuales me ayudan a crecer y ser mejor persona, pero sobretodo gracias por todo su tiempo, su amor y cariño incondicional. También quiero agradecer a mis hermanos por todas las noches de desvelo estudiando juntos y porque me llenan de vida con su alegría.

Al Dr. Ricardo Strausz por introducirme en este nuevo e interesante mundo del computo molecular, por la confianza que deposito en mi para trabajar en este proyecto, por su apoyo y el tiempo que tomo para compartirme un poco de su conocimiento.

Quiero agradecer a los sinodales que forman parte mi jurado Mariana, Armando, Carlos y Sergio por el tiempo que se tomaron revisando esta tesis y por sus comentarios constructivos que ayudaron a mejorar el trabajo escrito.

Al Posgrado en Ciencia e Ingeniería de la Computación - UNAM por haberme dado la oportunidad de dedicar mi tiempo a la ciencia, compartiendome sus instalaciones y sus profesores para realizar mis estudios de maestría. Así mismo quiero agradecer al CONACYT por brindarme una beca durante mis estudios de maestría, ya que me ayudaron a sustentarme y pude dedicarme a lo que me apasiona y así logre aportar un poco a la ciencia.

Por último quiero agradecer a mis amigos que hicieron mi estancia en el posgrado más amena y me apoyaron durante esta etapa. Quiero agradecer a quien me apoyo y ayudo en los momentos más estresantes, por su compañia en los desvelos, por las ideas que compartimos juntos y los sueños.

Resumen

El presente trabajo tiene como objetivo la creación de algoritmos computacionales para la búsqueda de una cadena de texto, que representa un oligonucleótido de interferencia, el cual será útil para la creación de una computadora molecular capaz de diagnosticar e interferir la fibrosis hepática. Para esto, el problema se dividió en tres partes: búsqueda del mejor oligonucleótido de interferencia, búsqueda en el genoma del oligonucleótido de interferencia y minimización del riesgo de silenciamiento génico fuera del objetivo.

Para cada parte del problema inicialmente se expuso un algoritmo de fuerza bruta con su respectiva complejidad y tiempo de ejecución, pero con el fin de mejorar estos parámetros se propusieron nuevos algoritmos:

- El algoritmo desarrollado para la búsqueda en el genoma del oligonucleótido de interferencia mejora el tiempo de ejecución de los algoritmos de Boyer-Moore y de Knuth-Morris-Pratt. En el caso de la cota de complejidad, si n es la longitud del texto donde se buscará la cadena y m es la longitud de la cadena a ser buscada, la cota del algoritmo desarrollado $O(n)$, mejora la cota del algoritmo de Boyer-Moore, ya que este en el peor de los casos tiene una cota de complejidad de $O(nm)$. A pesar de que la cota de complejidad del algoritmo de Knuth-Morris-Pratt es igual que la cota del algoritmo desarrollado, este último es más sencillo de entender y de implementar.

  Realizando algunas modificaciones en el algoritmo desarrollado, este puede ser usado para resolver problemas de búsqueda de subcadenas con diferentes fines no necesariamente bioinformáticos.

- El algoritmo desarrollado para la búsqueda del mejor oligonucleótido de interferencia mejora la cota de complejidad del algoritmo de fuerza bruta, esto se logra haciendo uso de programación dinámica.

  También se expone un enfoque geométrico como opción para resolver el problema de búsqueda de un oligonucleótido de interferencia, que quizá pueda ser usado en un futuro.

- El algoritmo desarrollado para minimizar el riesgo de silenciamiento génico fuera del objetivo en el genoma humano tiene un menor tiempo de ejecución que el algoritmo de fuerza bruta. Esto gracias a un análisis con la fórmula de la Tm (temperatura de disociación), que reduce el problema al problema de la búsqueda de una cadena que con cierto número de letras distintas podría ser subcadena de una segunda cadena.

Estos algoritmos se implementaron y agruparon en una aplicación, con el fin de que se utilice para la búsqueda de oligonucleótidos de interferencia, donde el parámetro de temperatura de disociación (Tm) puede ser elegido por el usuario. Así también se pueden realizar búsquedas de olignucleótidos en la base de datos que prefiera el usuario, ya que la aplicación tiene la capacidad de descargar y usar la base de datos que le sea más útil en ese momento.

Se probo la aplicación usando como entrada los genes $PPAR\gamma 1$ y $Hsp47$, para la búsqueda de oligonucleótidos de interferencia que pudieran ayudar a inhibir la fibrosis hepática. Se examinaron los resultados obtenidos y se escogieron dos oligonucleótidos de interferencia para ser probados en el laboratorio y comprobar si ayudan a inhibir la fibrosis hepática, así también se propuso un factor característico que podría ser el causante de que alguno de los oligonucleótidos no ayuden a la interferencia.

Introducción

El computo basado en ADN (*DNA computing*) es una manera de hacer computo que usa como recursos el ADN, la biología molecular y la bioquímica en vez de circuitos de silicon; las computadoras de ADN resultan ser más rápidas y pequeñas que cualquier otra computadora construida hasta ahora.

A finales del siglo pasado un grupo lidereado por Ehud Shapiro, propuso usar el computo basado en ADN para resolver problemas biológicos relacionados con la medicina [Benenson et al., 2004], en lugar de problemas matemáticos [Adleman, 1994]; en particular, construyeron un par de "computadoras moleculares", que en teoría podían diagnosticar e interferir el cáncer de próstata y de pequeñas células de pulmón, haciendo uso del siguiente paradigma "simplificado":

Administrar TCTCCCAGCGTGCGCCAT si y sólo si
↑ ASCL1 y ↑ GRIA2 y ↑ INSM1 y ↑ PTTG [1]

Esta nueva tecnología puede ser usada para diagnosticar e interferir la fibrosis hepática, ysando como células objetivo las células estelares hepáticas (HSC por sus siglas en inglés). La fibrosis hepática es causada principalmente por virus de hepatitis y el abuso de alcohol. Esta enfermedad es uno de los estados del daño crónico que sufre el hígado, resultado de la acumulación progresiva y falta de reestructuración de la colágena en la matriz extracelular (ECM por sus siglas en inglés).

Las lesiones hepáticas provocan que los hepatocitos mueran ocasionando la secreción de citocinas proinflamatorias y fibrogénicas en las células de Kupffer, y promoviendo la activación de las HSC, convirtiendose estas en las principales productoras de colágeno fibrótico [Friedman, 2004].

Se han realizado enormes esfuerzos para crear estrategias terapéuticas innovadoras contra el avance de la fibrosis; la dirección de estos intentos ha cambiado de inhibir la activación de las HSC [Albanis et al., 2003] a la estrategia de revertir el proceso fibrótico por diferentes métodos [Elsharkawy et al., 2005, Suárez Cuenca et al., 2008].

De todos estos estudios, el trabajo [Sato et al., 2008] muestra que la inhibición de gp46, una proteína chaperona homóloga de la Hsp47 humana en ratas, detiene el proceso fibrótico. Sin embargo, esta estrategia se centra sólo en el control de los daños en los tejidos, pero aún mantiene activadas las HSC presentes en el hígado.

Hay al menos dos maneras de evitar esta limitación: una es la inducción de la apoptosis; y la otra es la reversión del fenotipo activado al estado quiescente original. En el artículo [She et al., 2005] se ha hecho un gran avance, demostrando que la sobreexpresión del receptor

[1] ↑ ASCL1 se indica para la sobreexpresión del gen *achaete-scute complex homolog 1*; análogamente, GRIA2 se indica para *glutamate receptor inotropic AMPA 2*, INSM1 para *insulinoma associated 1* y PTTG1 para *pituitary tumor transforming 1*. TCTCCCAGCGTGCGCCAT es también conocido como *oblimersen* y es propuesto para ser un medicamento en la terapia antigénica para pequeñas células cancerígenas de pulmón

activado por proliferador de peroxisoma ($PPAR\gamma 1$) a través de un vector adenoviral en un cultivo de HSC activadas, invierte el proceso y restaura la capacidad de estas células para acumular ésteres de retinol.

Toda esta información proporciona una oportunidad única para crear una computadora de ADN que puede usarse para determinar el fenotipo del estado de las HSC, y en caso de que el diagnóstico sea positivo, inducir la expresión de $PPAR\gamma 1$ [Farfan et al., 2010].

En concreto, se propone como "paradigma simplificado" para diagnosticar e interferir la fibrosis hepática el siguiente algoritmo:

Si ↑Hsp47 entonces ↑PPARγ1 y ↓Hsp47.

Una parte fundamental para llevar a cabo el diseño de dicha computadora molecular es encontrar un oligonucleótido de interferencia, mismo que se describirá en detalle a lo largo de esta tesis. Un oligonucleótido de interferencia es una hebra corta de ADN que se une a un ARNm para formar una doble hebra, que el ribosoma será incapaz de leer, evitando así la creación de una proteína.

En el capítulo uno de este trabajo, se presenta una introducción biológica y biomédica con el fin de entender algunos procesos y términos que ayuden a entender un poco el problema y posibles soluciones biomédicas.

En el capítulo dos se describe un método para tratar de solucionar el problema planteado de encontrar un oligonucleótido de interferencia que ayude a la inhibición de la fibrosis hepática y se detalla la abstracción computacional de este método con el fin de hacer uso de la ciencia de la computación para desarrollar nuevos algoritmos. Así también se explican los algoritmos de fuerza bruta que pueden solucionar el problema y se hace un análisis de los mismos.

En el capítulo tres se exponen algunos algoritmos que se encuentran en la literatura que pueden ayudar a solucionar parcialmente el problema descrito.

En el capítulo cuatro se describen las aportaciones de este trabajo, entre las cuales se encuentran tres algoritmos propuestos que son soluciones parciales del problema: búsqueda del mejor oligonucleótido de interferencia, búsqueda en el genoma del oligonucleótido de interferencia y minimización del riesgo de silenciamiento génico fuera del objetivo. Se expone un análisis de la cota de complejidad de estos algoritmos y se realiza una comparación con los ya descritos en la literatura y/o con los algoritmos de fuerza bruta.

En este mismo capítulo se detalla la aplicación desarrollada, sus características y forma de usarla. Se exponen las pruebas realizadas con la aplicación haciendo uso de ciertas cadenas de genes para la búsqueda de un oligonucleótido de interferencia que pueda ayudar a inhibir la fibrosis hepática. Se examinan varios resultados obtenidos y se proponen dos oligonucleótidos de interferencia para ser probados en el laboratorio y comprobar si ayudan a inhibir la fibrosis hepática, así también se propone un factor característico que podría ser el causante de que alguno de los oligonucleótidos no ayuden a la interferencia.

Por último se describe un enfoque geométrico como opción para resolver el problema de búsqueda de un oligonucleótido de interferencia, el diseño y análisis de algunos algoritmos para su respectiva implementación y los resultados de los mismos.

Capítulo 1

Elementos de biomedicina

1.1. Fibrosis hepática

El hígado está compuesto principalmente de hepatocitos, que suelen aparecer organizados en forma de cordones de una o dos células de grosor, separados por sinusoides. El sinusoide es la unidad microvascular hepática que tiene una capa de células endoteliales, la cual se distingue por ser porosa y está separada de los hepatocitos por el espacio de Disse que contiene matriz extracelular basal de baja densidad, la cual es suficientemente porosa para permitir el intercambio metabólico entre los hepatocitos y el flujo sanguíneo [Friedman, 2000, Hernández and Friedman, 2011]. En este espacio residen las HSC, las cuales en estado quiescente funcionan como almacén de ácido retinoico (vitamina A).

Sin embargo en respuesta a un virus, alcohol u otro agente dañino, las HSC sufren un proceso denominado "activación", en el cual se produce una acumulación de proteínas de tipo fibrilar en la matriz extracelular y se empieza a presentar el problema de la fibrosis hepática.

La activación de las células estelares hepáticas puede ser el resultado de la interacción de diferentes tipos celulares hepáticos:

- Los **macrófagos activados o células de Kupffer**, responden a un estimulo lesivo transformando las células estelares hepáticas a miofibroblasto, incrementando la síntesis de proteínas, de elástina y de colágeno. También incrementa la respuesta al factor de crecimiento derivado de las plaquetas (FCDP). Estas últimas son una fuente importante de citocinas proinflamatorias y profibrogénicas que promueven el crecimiento, transformación y síntesis de la matriz extracelular [Páramo Hernández et al., 2010].

- La **exposición de los hepatocitos a agentes citotoxicos** hacen que liberen sustancias con actividad mitogénica similar al FCDP y también puede favorecer la inducción de colágeno tipo I, incluso la destrucción de los hepatocitos es un mecanismo que ayuda a la activación de las células estelares hepáticas [Páramo Hernández et al., 2010].

Se han realizado diferente esfuerzos para crear estrategias terapéuticas innovadoras contra el avance de la fibrosis, como inhibir la activación de las HSC [Albanis et al., 2003] y revertir el proceso fibrótico por diferentes métodos [Elsharkawy et al., 2005, Suárez Cuenca et al., 2008].

1.1.1. Estrategias contra su avance

El trabajo [Sato et al., 2008] muestra que la inhibición de gp46, una proteína chaperona homóloga a la Hsp47 humana en ratas detiene el proceso fibrótico.

Hsp47 es una proteína inhibidora de serín proteasas, que actúa en la célula como chaperona para la colágena [Dafforn et al., 2001]. Se encuentra localizada en el cromosoma humano 11q13.5, un conocido "punto caliente" en una serie de cánceres humanos. Se ha mostrado que Hsp47 en células en hígados sometidos a fibrosis o cirrosis se encuentra asociado con el aumento de colágena de tipo I y tipo III [Masuda et al., 1994], debido a que ayuda a permitir la formación de fibras y por lo mismo está sobreexpresada en muchas enfermedades fibroticas y manifiesta las propiedades de un autoantígeno.

Basándose en algunos estudios se ha demostrado que el tratamiento con oligonucleótidos antisentido de Hsp47 es útil para controlar la formación de cicatrices [Wang et al., 2003], la glomerulosclerosis [Sunamoto et al., 1998], la fibrosis pulmonar y la cirrosis [Nagata, 2003].

1.1.2. Estrategias para revertirla

Estudios experimentales y clínicos han aportado evidencia considerable que sugiere que la fibrosis, e inclusive la cirrosis deben ser considerados como procesos potencialmente reversibles. Una condición necesaria es la eliminación o el tratamiento eficiente del agente que causa el padecimiento hepático.

Con la reversión de la fibrosis, la población de células estelares hepáticas en el hígado sufre apoptosis (muerte celular), o bien puede revertir su estado hacia un fenotipo quiescente. Por lo que el ambiente en el tejido hepático cambia y las células dejan de secretar señales pro-fibrogénicas y de supervivencia, lo que favorece también la reversión de las HSC [Kisseleva and Brenner, 2013, Trautwein et al., 2015]. La inducción de las células estelares hepáticas hacia un estado quiescente puede promoverse mediante la activación de mecanismos que modifiquen el perfil de expresión genética de las células. Un mecanismo esencial es promover la producción de $PPAR\gamma 1$ (*Peroxisome proliferator activated receptor gamma*). $PPAR\gamma 1$ es un receptor nuclear dependiente de ligando (un ligando es una molécula que se une al centro activo de una proteína para que ésta pueda realizar su función), se encuentra en el cromosoma 3p25 y actuá como un factor de transcripción expresandose predominantemente en tejido adiposo [King, 2015].

PPAR$\gamma 1$ es un factor de transcripción importante en la fibrosis hepática, ya que además de estar involucadro en la fase de agotamiento de los retinoides de la fibrosis hepática, tiene un efecto anti-inflamatorio debido al ligando dependiente de la represión de factores proinflamatorios, por lo que logra un efecto inhibidor en el nivel de macrófagos. Existen diferentes pruebas de laboratorio que demuestran que la expresión de PPAR$\gamma 1$ o la adición de sus ligandos revierte los cambios bioquímicos y morfológicos característicos de la activación de la célula estelar hepática [Hazra et al., 2004].

Para diagnosticar e interferir la fibrosis hepática se puede hacer uso de los dos elementos mencionados anteriormente que intervienen en la fibrosis hepática, Hsp47 cuya inhibición reduce la producción de colágeno y $PPAR\gamma 1$ cuya expresión favorece la reversión de las HSC, haciendo uso del "paradigma simplificado" ya mencionado en la introducción, que hace uso de olignucleótidos de interferencia :

$$\text{Si } \uparrow\text{Hsp47entonces } \uparrow\text{PPAR}\gamma 1 \text{ y } \downarrow\text{Hsp47}.$$

Hay un concepto parecido a oligonucleótidos de interferencia que son los ARN interferentes, en especifico a los siARN (*small interfering RNA*), que son hebras simples con una longitud de 20 a 25 nucleótidos altamente específicas diseñadas para que se acoplen a la secuencia de nucleótidos de un ARNm objetivo, interfiriendo con ello la expresión del gen respectivo.

Para poder diseñar un siARN se debe de conocer la secuencia de nucleótidos del ARNm del gen que nos interesa silenciar. Así mismo se debe proponer una cadena de nucleótidos y su cadena complementaria (antisentido), las cuales se pueden obtener a partir de programas ya disponibles, los cuales constan de algoritmos basados en las características comunes de los siARN funcionales estudiados.

Basados en algunas investigaciones donde se analízan diversos siARNs efectivos, se pueden destacar algunas características importantes para el diseño del oligonucleótido de ARNi, como las siguientes: es importante verificar la efectividad (potencia) estabilidad y especificidad del siARN. Lo anterior incluye evitar los *motivos* conocidos que puedan activar la respuesta inmune innata. Un *motivo* es una secuencia breve de nucleótidos que suele servir para una determinada función. En este caso se han identificado *motivos* ricos en GU de siARNs que en cuya presencia podrían activar la respuesta inmune innata; por ejemplo, se diseñaron nuevos siARNs dirigidos a $\beta - gal$, seleccionando secuencias que evitaban los *motivos* GUGU o poli (U) en cualquiera de las dos hebras, y el resultado para $\beta - gal$ 924 y 2891 fue que se induce poco o ningún IFN-α (interferón el cual activa las defensas protectoras del sistema inmune que participan en la erradicación de patógenos) en cultivos PBMC (células mononucleares de sangre periférica), mientras que la respuesta a $\beta - gal$ 478 se redujo en más de un 90 % en comparación con la respuesta a la original $\beta - gal728$ de doble hélice. Para $\beta - gal728$ se ha identificado que 5'-UGUGU-3' es un *motivo* inmunoestimulante [Judge et al., 2005].

1.2. Desnaturalización y temperatura de disociación del ADN

El ácido desoxirribonucleico (ADN) esta constituido por una doble hélice formada de nucleótidos, cada nucleótido esta formado por dos partes: un azúcar (desoxirribosa) con un grupo fosfato unido a él, y una de las cuatro bases nitrogenadas siguientes: adenina (A), guanina (G), citosina (C) o timina (T).

La desnaturalización consiste en la separación de las dos hebras del ADN y esta puede lograrse mediante el calor. De acuerdo con la estructura de la doble hélice, la alteración de un sector de la molécula potencializa la desestabilización de un sector mayor, o sea, el proceso de desnaturalización tiene un carácter cooperativo. Las gráficas obtenidas del proceso se conocen como curvas de disociación y la temperatura de su punto medio se conoce como temperatura de disociación, que se simboliza por Tm. La estabilidad del ADN y, por tanto, su Tm dependen de varios factores como la naturaleza del solvente, el tipo y concentración de los iones, así como el valor del pH. Si estos factores se mantienen constantes y solo se varía el ADN, entonces Tm es una función lineal creciente del contenido de pares GC, lo que indica la mayor estabilidad de estos, unidos por tres puentes de hidrógeno, que de los pares AT unidos solo por dos.

Se han propuesto muchas fórmulas útiles para calcular la Tm para experimentos de PCR. Los principales factores que afectan este valor son: concentración de sales, concentración de ADN, presencia de agentes desnaturalizantes (DMSO o formamida), secuencia, longitud y condiciones de hibridización.

La ecuación mas simple para calcular la Tm de secuencias con 14 nucleótidos o menos, está dada por la Regla de Wallace [Wallace et al., 1979]

$$Tm = (A + T) \cdot 2 + (G + C) \cdot 4 \tag{1.1}$$

en donde A, G, C, y T corresponden número de las bases A, T, G, C presentes en la secuencia Esta ecuación se desarrolló para oligonucleótidos que hibridizarían con sus correspondientes complementarios unidos a una membrana con una concentración de NaCl 0.9M

Para secuencias de más de 13 nucleótidos, la ecuación usada es la formúla de Marmur

[Marmur and Doty, 1962] :

$$Tm = 64{,}9 + 41 \cdot (G + C - 16{,}4)/(A + T + G + C) \tag{1.2}$$

Capítulo 2

Planteamiento del problema

2.1. Objetivo

Desarrollar algoritmos computacionales para la búsqueda de una cadena de texto, que representa un oligonucleótido de interferencia, útil para la creación de una computadora molecular capaz de diagnosticar e interferir la fibrosis hepática.

2.2. Descripción biológica del problema

Hacer uso de oligonucleótidos de interferencia puede ser una estrategia para bloquear el progreso de la fibrósis hepática. Por lo que se plantea buscar un oligonucleótido de interferencia de doble hebra óptimo, donde una hebra servirá como inhibidor específico del gen Hsp47 y la otra hebra expresará $PPAR\gamma 1$, esto porque la inhibición de Hsp47 reduce la producción de colágeno y la expresión de $PPAR\gamma 1$ favorece la reversión de las HSC, ayudando así a revertir la fibrosis hepática.

Por lo tanto una hebra, a la que llamaremos α, del oligonucleótido de interferencia debe ser totalmente complementaria a un trozo del ARNm de Hsp47 y a su vez ser parcialmente complementaria a un trozo de PPAR$\gamma 1$, para que una vez que el oligonucleótido de doble hebra entre al cuerpo humano las dos hebras puedan separarse, con el fin de que cada hebra realice la función para la que fue diseñada. La temperatura del cuerpo humano es de aproximadamente 37°, por lo que se propone que las dos hebras del oligonucleótido tengan una Tm menor o igual a 37° para asegurar su desnaturalización dentro del cuerpo humano.

Debido a que es posible que el oligonucleótido propuesto interfiera con otros genes, no solo con Hsp47, primero se debe verificar si es totalmente complementario en alguna parte de los ARNm maduros del genoma humano. En caso de que no sea totalmente complementario es necesario tener una tolerancia de empalme entre el oligonucleótido propuesto y los ARNm maduros del genoma humano, la cual indicará que el oligonucleótido muy probablemente silencia otro gen. La tolerancia estará relacionada con la Tm del oligonucleótido propuesto y los ARNm maduros del genoma humano. Si la Tm de la hebra α con el ARNm fuera 20°C mayor que la Tm de la doble hebra del oligonucleótido de interferencia, entonces es posible y muy probable que el oligonucleótido silencie otro gen que no es el gen objetivo. Esto es debido a que las dos hebras de oligonucleótidos no permanecen todo el tiempo unidas, se separan y vuelven a encontrarse,

pero al estar separadas, una de las hebras puede ser afin con otra hebra y entonces unirse a ella, a este fenómeno se le conoce como competencia.

2.3. Propuesta de solución del problema

Se representará al oligonucleótido de interferencia como una cadena α, cuyo alfabeto esta en Σ={A,C,G,T}. La cadena se encuentra formada de la siguiente manera $a_1a_2...a_\ell$, donde ℓ es la longitud de la cadena y a_i son letras que pertenecen a Σ. Dada la descripción biológica del problema se propone que esta cadena debe de tener las siguientes carácterísticas:

- Sea O una cadena que representa el gen objetivo, es decir, el gen que se desea inhibir, en este caso Hsp47, la cadena α debe ser subcadena totalmente complementaria de una parte de la cadena O. Una cadena totalmente complementaria de acuerdo con la regla de complemento de Watson y Crick, se refiere a que cada letra de la cadena α será igual a su letra complemento en la subcadena de O, es decir, para la letra A su complemento es la letra T y viceversa, y para la letra G su complemento es la letra C y viceversa, por ejemplo, 3'-AGTTC-5' es totalmente complementaria a 5'-TCAAG-3', mientras que 3'-AGTTC-5', es parcialmente complementaria a 5'-TTCAG-3', porque la segunda y tercera letra no son complementarias entre las dos cadenas. Por lo tanto todas las subcadenas complementarias que pueden ser formadas de la cadena O pueden ser candidatas a ser α.
- Sea A una cadena que representa el gen actuador, en este caso PPARγ1, la cadena α debe ser subcadena parcialmente complementaria de la cadena A. Con cadena parcialmente complementaria nos referimos a que no todas las letras deben ser complementarias entre la cadena α y la subcadena de A, sino que el número de letras complementarias debe ser tal, que al evaluar la fórmula de la Tm ya sea la de Wallace o Marmur, se debe de cumplir que la Tm debe ser menor o igual a 37 °C (temperatura promedio del cuerpo humano).

Cabe aclarar en este punto que los oligonucleótidos parcialmente complementarios se unen con la hebra con la cual sus enlaces son más fuertes, hablando de Tm, esto se refiere al valor de Tm más alto. Por ejemplo, si se tiene una hebra de oligonucleótido representaod por la cadena A=3'-GTCTTG-5' unida a otra hebra de oligonucleótido representada por la cadena α=5'-AAG-3' y se desea saber en que parte están unidos los dos oligonucleótidos y si con una Tm menor o igual a 4°C se puede desnaturalizar; primero se obtiene el complemento de α, el cual es $\overline{\alpha}$=5'-TTC-3'. Después se requiere tomar el número de letras iguales entre A y α, pero la cadena A es más larga que α, por lo que A=GTCTTG se puede ver como un conjunto de subcadenas de tamaño 3 (ya que α tiene 3 letras), como el siguiente {GTC,TCT,CTT,TTG}, ahora si se calcula el número de letras A y T iguales (C_{AT}) y el número de las letras G y C iguales (C_{GC}) entre las subcadenas de A y $\overline{\alpha}$ como se muestra en la figura 2.1

El siguiente paso es calcular la Tm, en este ejemplo se hará uso de la fórmula de Wallace (ec. 1.2). En la siguiente figura se muestra el calculo de la Tm para cada subcadena de A con la cadena $\overline{\alpha}$.

La figura anterior muestra que la subcadena GTC con TTC tiene la máxima Tm, que es de 6°C. Esto quiere decir que los oligonucleótidos A y α=AAG se desnaturalizan a temperaturas de 6°C o mayores.

Para el caso del problema tratado en este trabajo se deben calcular todas las coincidencias C_{AT} y C_{GC} entre todas las subcadenas de O (Hsp47) (candidatas a ser $\overline{\alpha}$) y las subcadenas de A ($PPAR\gamma1$). Se obtienen los valores de la Tm al evaluar las coincidencias una subcadena de O

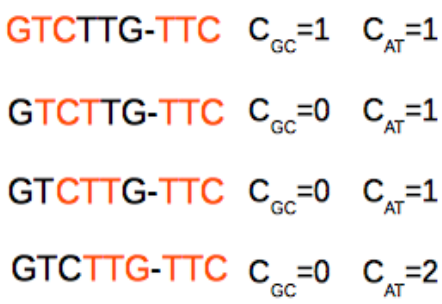

Figura 2.1: Coincidencias de letras entre las subcadenas de GTCTTG ({GTC, TCT, CTT, TTG}) y TTC. La parte de color rojo muestra una subcadena de la cadena GTCTTG que se esta comparando con la cadena TTC. Del lado derecho C_{AT} es el número de las letras iguales A y T entre una subcadena de A y la cadena α, y C_{GC} es el número de letras C y G iguales entre una subcadena de A y la cadena $\overline{\alpha}$

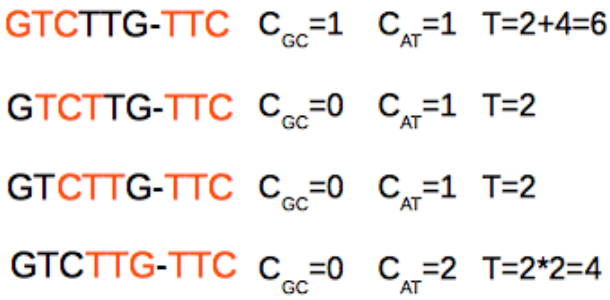

Figura 2.2: Evaluación de Tm entre las subcadenas de GTCTTG ({GTC,TCT,CTT,TTG}) y TTC. En la parte derecha se muestra el cálculo de la Tm con la fórmula de Wallace de las subcadenas A y α

con todas las subcadenas de A y se selecciona la Tm máxima. De esta manera se tendrá un valor de Tm máxima por cada subcadena de O, pero es necesario que la temperatura cumpla la restricción de que la Tm no debe de sobrepasar los 37°C, por lo que se seleccionarán las subcadenas de O que tengan el valor más pequeño de Tm máximo y que no sobrepase los 37°C.

Para seleccionar la mejor cadena se proponen el siiguiente conjunto de caracterśticas:

1. Del conjunto de subcadenas que se tienen hasta el momento tendrá más prioridad la subcadena O que cuente con mayor número de coincidencias C_{AT} y C_{GC} con la cadena A, ya que será una estructura de doble hebra más estable.

2. Tendrá prioridad una subcadena cuyo valor de Tm máxima se repite menos veces en la cadena A, por ejemplo, si se tiene la cadena A=ATGTTATCAT y la subcadena α_1=AT existen tres subcadenas de A con valores de Tm máxima, mientras que con la cadena α_4=TT tiene la Tm máxima en un sólo lugar como se muestra en la figura 2.3, por lo tanto la mejor subcadena candidata es α_4. Esta restricción tiene el fin de tener un mayor control sobre la unión que formará la cadena α con la cadena PPARγ1

3. El mejor candidato α, no debe aparecer como subcadena complementaria en otro ARNm maduro del genoma humano que no sea el gen objetivo, con el fin de que ésta no inhiba una expresión génica indispensable para el buen funcionamiento del cuerpo humano.

4. El mejor candidato α, debe de cumplir con la tolerancia de incompatibilidad; es decir, si la Tm entre la cadena α con alguna subcadena de los ARNm maduros del genoma

humano es 20°C mayor que la Tm entre α y A, entonces es posible y muy probable que el oligonucleótido silencie ese gen además del gen objetivo, por lo que se concluiría que no es un buen candidato.

Figura 2.3: Repeticiones de AT y TT en la cadena A

2.4. Definición computacional del problema

Se tiene una cadena A de longitud ℓ_A que representa el gen actuador y una cadena O de longitud ℓ_O que representa el gen objetivo, cuyo alfabeto es $\Sigma = \{A, G, C, T\}$

Sea $C_{GC} : \Sigma \times \Sigma \times \mathbb{N} \to \mathbb{N}$ una función que cuenta el número de letras G o C que son iguales entre dos cadenas de la misma longitud, es decir.

$$C_{GC}(a_{i\ldots}a_\ell, b_i \ldots b_\ell, \ell) = \begin{cases} 0 & \ell = 1 \\ 1 + C_{GC}(a_{(i+1)\ldots}a_\ell, b_{(i+1)} \ldots b_\ell, \ell - 1) & \ell > 0, a_i = b_i, a_i = G \text{ ó } a_i = C \\ 0 + C_{GC}(a_{(i+1)\ldots}a_\ell, b_{(i+1)} \ldots b_\ell, \ell - 1) & \ell > 0, a_i \neq b_i, a_i \neq G \text{ y } a_i \neq C \end{cases}$$

donde las a_i representan cada una de las letras de una cadena y las b_i representan las letras de otra cadena, tal que $1 \leq i \leq l$.

Sea $C_{AT} : \Sigma \times \Sigma \times \mathbb{N} \to \mathbb{N}$ una función que cuenta el número de letras A ó T que son iguales entre dos cadenas de la misma longitud, es decir.

$$C_{AT}(a_{i\ldots}a_\ell, b_i \ldots b_\ell, \ell) = \begin{cases} 0 & \ell = 1 \\ 1 + C_{AT}(a_{(i+1)\ldots}a_\ell, b_{(i+1)} \ldots b_\ell, \ell - 1) & \ell > 0, a_i = b_i, a_i = A \text{ ó } a_i = T \\ 0 + C_{AT}(a_{(i+1)\ldots}a_\ell, b_{(i+1)} \ldots b_\ell, \ell - 1) & \ell > 0, a_i \neq b_i, a_i \neq G \text{ y } a_i \neq C \end{cases}$$

Sea S_A el conjunto de subcadenas de longitud ℓ de la cadena A, es decir,

$$S_A = \{A_1, A_2, \ldots, A_i, \ldots, A_{(\ell_A - \ell)}\}$$

donde A_i es la subcadena que comienza en la posición i de la cadena A.

Sea S_O el conjunto de subcadenas de longitud ℓ de la cadena O, es decir,

$$S_O = \{O_1, O_2, \ldots, O_j, \ldots, O_{(\ell_O - \ell)}\}$$

donde O_j es la subcadena que comienza en la posición j de la cadena O.

Sea $Tm : \Sigma \times \Sigma \to \mathbb{Z}$ la función de Tm de Wallace, tal que

$$Tm(A_i, O_j) = 4 \cdot C_{GC}(A_i, O_j) + 2 \cdot C_{AT}(A_i, O_j)$$

. Sea $Tm_{max} : S_O \to \mathbb{Z}$ dada por

$$Tm_{max}(O_i) = max\,\{Tm(O_i, A_j) \mid A_j \epsilon S_A\}$$

Tal que $1 \leq i \leq (\ell_A - \ell)$ y $1 \leq j \leq (\ell_O - \ell)$. Esta función devuelve los valores máximos de Tm de cada subcadena de O con todas las subcadenas de A.

Sea

$$S_{min} = \{O_i \epsilon S_O | Tm_{max}(O_i) \leq Tm_{max}(O_j), \forall O_j \epsilon S_O\}$$

Este conjunto devuelve las subcadenas cuyo valor de Tm es el más pequeño.

Sea $\ell_{max} : S_{min} \to \mathbb{Z}$ dada por

$$\ell_{max}(O_i) = max\,\{C_{GC}(O_i, A_j) + C_{AT}(O_i, A_j) | A_j \epsilon S_a\}$$

Esta función devuelve el número máximo de letras iguales entre un elemento de S_{min} y las subcadenas de S_A.

Sea

$$S_\ell = \{O_i \epsilon S_{min} | \ell_{max}(O_i) \geq \ell_{max}(O_j) \forall O_j \epsilon S_{min}\}$$

Este multiconjunto está compuesto por todas las subcadenas de S_{min}, que tengan el valor de ℓ_{max} más alto.

Sea $S_r : S_\ell \to \mathbb{N}$ dada por

$$S_r(O_i) = |\,\{O_j | Tm(O_i, A_j) = Tm_{max}(O_i) \forall A_j \epsilon S_A\}\,|$$

Esta función devuelve el número de veces que se repite el valor de Tm_{max} en S_ℓ (visto como multiconjunto) .

Sea

$$S_{rl} = \{O_i \epsilon S_\ell | S_r(O_i) \leq S_r(O_j) \forall O_j \epsilon S_\ell\}$$

Este conjunto contiene los elementos de S_l con el menor número de repeticiones de Tm_{max}.

Sea $p : S_{rl} \to \mathbb{N}$ dada por

$$p(O_i) = min\,\{j \epsilon \mathbb{N} | Tm(S_{rl}, A_j) = Tm_{max}(S_{rl}) \forall A_j \epsilon S_A\}$$

Esta función expresa el índice del primer elemento en S_A cuya Tm sea igual a la Tm_{max} de cada elemento de S_{rl}.

Sea

$$B = \{O_i \epsilon S_{rl} | p(O_i) \leq p(O_j) \forall O_j \epsilon S_{rl}\}$$

Este conjunto contiene los elementos de S_{rl} cuyo valor de p sea el menor.

Si G es una cadena que representa el genoma humano de tamaño ℓ_G, entonces S_G es el conjunto de subcadenas de tamaño ℓ de G, es decir, $S_G = \{G_1, G_2, ..., G_{\ell_G - \ell}\}$. La mejor cadena del conjunto B será aquella que no se encuentre dentro del conjuto S_G.

2.5. Algoritmos de fuerza bruta para solucionar el problema

El problema de la búsqueda de un oligonucleótido de interferencia adecuado se puede dividir en tres subproblemas:

1. Búsqueda del mejor oligonucleótido de interferencia.
2. Búsqueda en el genoma del oligonucleótido de interferencia.
3. Minimizar el riesgo de silenciamiento génico fuera del objetivo.

Estos se pueden solucionar con diversos algoritmos computacionales, pero se empezará exponiendo los algoritmo de fuerza bruta con su respectiva complejidad.

2.5.1. Búsqueda del mejor candidato de oligonucleótido de interferencia.

Para buscar el mejor candidato de la cadena de oligonucleótido de interferencia, se puede definir un algoritmo de fuerza bruta, que consistirá en seguir paso a paso la propuesta de solución del problema descrita en la sección 2.4, como se muestra en el algoritmo 1 para búsqueda del mejor candidato de oligonucleótido de interferencia.

Algoritmo 1 Algoritmo de búsqueda del mejor candidato de oligonucleótido de interferencia.

```
1: procedure ALGORITMO(A, O)                    ▷ A=gen actuador, O=gen objetivo
2:     for i=1:i≤ ℓ_O − ℓ:i++ do      ▷ Obtiene las subcadenas de longitud ℓ de la cadena O
3:         for j=i:j< ℓ+i:j++ do
4:             Concatenar letra o_j en O_i
5:         end for
6:     end for
7:     for i=1:i≤ ℓ_A − ℓ:i++ do      ▷ Obtiene las subcadenas de longitud ℓ de la cadena A
8:         for j=i:j< ℓ+i:j++ do
9:             Concatenar letra a_j en A_i
10:        end for
11:    end for
12:    for i=1:i≤ ℓ_O − ℓ:i++ do                ▷ i es el número de letra de la cadena O
13:        for j=1:j≤ ℓ_A − ℓ:j++ do             ▷ j es el número de letra de la cadenaA
14:            C_AT = 0
15:            C_GC = 0
16:            for k=1:j≤ ℓ:j++ do
17:                if O_{i_k}==A_{j_k} then
18:                    switch O_{i_k} do
19:                        case A
20:                            C_AT ++
21:                        end case
22:                        case C
23:                            C_GC ++
```

```
                    end case
                    case G
                        C_GC++
                    end case
                    case T
                        C_AT++
                    end case
                end switch
            end if
        end for
        Tm = 64,9 + 41 · (C_GC − 16,4)/(C_AT + C_GC)
        if Tm > Tm_max then                    ▷ Selecciona el valor de Tm más grande
            Tm_max = Tm
            indiceJ = j
            contador = 0
        else if Tm = Tm_max then
            contador++      ▷ Número de veces que aparece el valor de Tm más grande
        end if
    end for
    if Tm_max < mejorTm then          ▷ Selecciona el valor de Tm_max más pequeño
        mejorTm = Tm_max
        numeroTm = contador
    else if Tm_max = mejorTm then
        if contador < numeroTm then   ▷ Selecciona el valor de Tm_max que se repita menos veces
            indiceI = i
        end if
    end if
  end for
end procedure
```

Para calcular el tiempo de ejecución del algoritmo, se toma el número de operaciones involucradas en el mismo, para el caso del algoritmo anterior, obtener las posibles subcadenas de longitud ℓ contenidas en la cadena O, se realiza en un ciclo de tamaño $\ell(\ell_O - \ell)$, obtener las posibles subcadenas de longitud ℓ contenidas en la cadena A se lleva a cabo en un ciclo de tamaño $\ell(\ell_A - \ell)$ y para obtener la Tm se ejecutan tres ciclos con un número constante de operaciones entre cada ciclo, de esto último se tienen $[(\ell_O - \ell)[6 + (\ell_A - \ell)(4\ell + 6)]]$ operaciones. Así el algoritmo se lleva a cabo en el siguiente número de operaciones:

$$[\ell(\ell_O - \ell)] + [\ell(\ell_A - \ell)] + [(\ell_O - \ell)[6 + (\ell_A - \ell)(4\ell + 6)]]$$

Ahora para calcular la cota de complejidad se tiene que

$$[\ell(\ell_O - \ell)] + [\ell(\ell_A - \ell)] + [(\ell_O - \ell)[k_1 + (\ell_A - \ell)(k_2\ell + k_1)]]$$

donde $k_1 = 6$ y $k_2 = 4$. Si $\ell_A >> \ell$, $\ell_A >> k_2$ y $\ell_O >> \ell$, entonces la expresión se reduce a

$$\ell\ell_O + \ell\ell_A + [\ell_O(\ell_A k_2 \ell + \ell_A k_1)]$$

Diremos que $n = \ell$, además $\ell < \ell_O$, por lo que $\ell_O = c_1 n$ para alguna constante c_1, así mismo $\ell < \ell_A$, por lo que $\ell_A = c_2 n$ para alguna constante c_2, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = n \cdot c_1 n + n \cdot c_2 n + [c_1 n(c_2 n \cdot k_2 n + c_2 n \cdot c_2)] = c_1 n^2 + c_2 n^2 + c_1 c_2 k_2 n^3 + c_1 c_2 k_1 n^2$$

Pero como $c_1 n^2 < c_1 c_2 k_2 n^3$, $c_2 n^2 < c_1 c_2 k_2 n^3$, $c_1 c_2 k_1 n^2 < c_1 c_2 k_2 n^3$, entonces $c_1 n^2 + c_2 n^2 + c_1 c_2 k_2 n^3 + c_1 c_2 k_1 n^2 < 4 c_1 c_2 k_2 n^3$, es decir, $f(n) < 4 c_1 c_2 k_2 n^3$, pero $4 c_1 c_2 k_2$ es una constante, por lo tanto la cota de complejidad es

$$O(n^3)$$

2.5.2. Búsqueda en el genoma humano del oligonucleótido de interferencia.

Para conocer si la cadena propuesta como oligonucleótido de interferencia (α) es un buen candidato para ser un oligonucleótido de interferencia, es indispensable que no sea una subcadena complementaria de los ARNm maduros del genoma. Esto con el fin de que está no inhiba la expresión de un gen indispensable para el humano.

Para llevar a cabo esta tarea, es necesario comparar cada una de las letras de la cadena α con cada letra de los ARNm maduros del genoma humano, para saber si existen varias posiciones donde la cadena α sea complemento en el genoma humano. Si G es la cadena de los ARNm maduros del genoma humano cuyo alfabeto es Σ, y está compuesta de la siguiente manera $g_1 g_2 ... g_{\ell_g}$, donde ℓ_g es la longitud del genoma; si la cadena α esta compuesta de la siguiente manera $\alpha_1 \alpha_2 ... \alpha_\ell$, donde ℓ es la longitud de la cadena, comparamos $\alpha_1 ... \alpha_\ell$ con $g_1 g_2 ... g_\ell$, después $\alpha_1 ... \alpha_\ell$ con $g_2 g_3 ... g_{\ell+1}$ y así sucesivamente con todas las subcadenas posibles de G y verificamos si la cadena α es complemento de alguna subcadenas de G. A continuación se describe el algoritmo 2 para la búsqueda del oligonucleótido de interferencia en el genoma humano y se hace un análisis de su complejidad:

Algoritmo 2 Algoritmo de búsqueda del oligonucleótido de interferencia en el genoma humano

```
1: procedure ALGORITMO(G, α)   ▷ G=ARNm maduros del genoma, α=oligonucleótido de interferencia
2:     for i=1:i≤ ℓ_g − ℓ:i++ do                ▷ i es la i-ésima letra de G
3:         letrasIguales=0
4:         for j=1:j≤ ℓ:j++ do                  ▷ j es la j-ésima letra de α
5:             if g_i==α_j then
6:                 letrasIguales++
7:             end if
8:             if letrasIguales==ℓ then
9:                 Imprimir "El oligonucleótido de interferencia se encuentra en el índice i − ℓ del genoma humano"
10:                letrasIguales=0
11:            end if
12:        end for
13:    end for
14: end procedure
```

Para calcular el tiempo de ejecución del algoritmo anterior se tiene que hay un ciclo con 5 operaciones anidado dentro de otro ciclo con una operación adicional. Por lo que el algoritmo se lleva a cabo en el siguiente número de operaciones:

$$(\ell_g - \ell)(5\ell + 1) = 5\ell(\ell_g - \ell) + (\ell_g - \ell)$$

Ahora para calcular la cota de complejidad se tiene que

$$k_1\ell(\ell_g - \ell) + (\ell_g - \ell)$$

donde $k_1 = 5$. Si $\ell_g >> \ell$ entonces la expresión se reduce a

$$k_1\ell\ell_g + \ell_g$$

Diremos que $n = \ell$, además $\ell < \ell_g$, por lo que $\ell_g = c_1 n$ para alguna constante c_1, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = k_1 n \cdot c_1 n + n = k_1 c_1 n^2 + n$$

Pero como $n < k_1 c_1 n^2$, entonces $k_1 c_1 n^2 + n < 2k_1 c_1 n^2$, es decir, $f(n) < 2k_1 c_1 n^2$, pero $2k_1 c_1$ es una constante, por lo tanto la cota de complejidad es

$$O(n^2)$$

2.5.3. Minimizar el riesgo de silenciamiento génico fuera del objetivo

Con el fin de minimizar el riesgo de silenciamiento génico fuera del blanco, es indispensable considerar una tolerancia de incompatibilidad, en este caso se propone que esta tolerancia esté relacionada con la Tm de α y las subcadenas de los ARNm maduros del genoma humano. Sí esta es 20°C mayor que la Tm máxima entre α y las subcadenas del gen de $PPAR\gamma 1$ entonces es posible y muy probable que el oligonucleótido silencie otro gen que no es el gen objetivo.

Por lo tanto es necesario evaluar la función Tm en todas las posibles subcadenas de los ARNm maduros del genoma humano, para asegurar que no exista un sitio en el genoma humano donde la Tm sea más alta o igual a 20°C más la Tm entre el oligonucleótido propuesto y el gen de $PPAR\gamma 1$.

El algoritmo más sencillo para resolver este problema es parecido al de búsqueda en el genoma del oligonucleótido de interferencia de la sección anterior, pero cuenta con algunas modificaciones. En el algoritmo anterior se compara cada letra de la cadena complementaria del oligonucleótido de interferencia con cada letra del genoma humano y cada que existe una coincidencia de letras se aumenta un contador; en este caso, existen dos contadores uno para las coincidencias de las letras AT (C_{AT}) y otro para las coincidencias de las letras GC (C_{GC}), con el fin de poder hacer el cálculo de la Tm.

Una vez obtenido el valor de Tm, se compara con el valor anterior de Tm (asignado a una variable $Tm_{anterior}$), se selecciona el valor más alto y se asigna a la variable $Tm_{anterior}$.

A continuación se describe el algoritmo 3 para minimizar el riesgo de silenciamiento génico fuera del objetivo y se realiza un análisis de su complejidad.

Algoritmo 3 Algoritmo para minimizar el riesgo de silenciamiento génico fuera del objetivo

procedure ALGORITMO(G, α) ▷ G=ARNm maduros del genoma, α=oligonucleótido de interferencia
  for i=1:i$\leq \ell_g - \ell$:i++ **do** ▷ i es la i-ésima letra de G
    for j=1:j$\leq \ell$:j++ **do** ▷ j es la j-ésima letra de α
      if $g_i == \alpha_j$ **then**
        switch g_i **do**
          case A
            $C_{AT}++$
          end case
          case C
            $C_{GC}++$
          end case
          case G
            $C_{GC}++$
          end case
          case T
            $C_{AT}++$
          end case
        end switch
      end if
    end for
    $Tm = 64{,}9 + 41 \cdot (C_{GC} - 16{,}4)/(C_{AT} + C_{GC})$
    if $Tm > Tm_{anterior}$ **then** ▷ Selecciona el valor de Tm más grande
      $Tm_{anterior} = Tm$
    end if
    if $Tm_{anterior} > 20 + Tm_{oligo}$ **then**
      Imprimir "Es muy problable que el oligonucleótido de interferencia inhiba la expresión génica fuera del objetivo".
    end if
  end for
end procedure

Para calcular el tiempo de ejecución del algoritmo anterior se tiene que hay un ciclo con 4 operaciones anidado dentro de otro ciclo con 5 operaciones adicionales. Por lo que el algoritmo se lleva a cabo en el siguiente número de operaciones:

$$(\ell_g - \ell)(4\ell + 5) = 4\ell(\ell_g - \ell) + 5(\ell_g - \ell)$$

Ahora para calcular la cota de complejidad se tiene que

$$k_1\ell(\ell_g - \ell) + k_2(\ell_g - \ell)$$

donde $k_1 = 4$ y $k_2 = 5$. Si $\ell_g >> \ell$ entonces la expresión se reduce a

$$k_1\ell\ell_g + k_2\ell_g$$

Diremos que $n = \ell$, además $\ell < \ell_g$, por lo que $\ell_g = c_1 n$ para alguna constante c_1, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = k_1 n \cdot c_1 n + k_2 n = k_1 c_1 n^2 + k_2 n$$

Pero como $k_2 n < k_1 c_1 n^2$, entonces $k_1 c_1 n^2 + k_2 n < 2k_1 c_1 n^2$, es decir, $f(n) < 2k_1 c_1 n^2$, pero $2k_1 c_1$ es una constante, por lo tanto la cota de complejidad es

$$O(n^2)$$

Capítulo 3

Marco teórico

3.1. Algoritmo de alineamiento local Smith-Waterman

El algoritmo de Smith-Waterman en este caso se tomó como inspiración para realizar el algoritmo que resolviera la parte del problema descrito de la búsqueda del mejor candidato de ARN de interferencia.

El algoritmo de Smith-Waterman es una estrategia para realizar alineamiento local; es decir, ayuda a determinar regiones similares entre un par de cadenas o secuencias biológicas ya sea de ADN, ARN o proteínas.

Este algoritmo fue propuesto por Temple Smith y Michael Waterman en 1981, y esta basado en programación dinámica: cada solución parcial de estados posteriores puede ser calculada por iteración sobre un número fijo de soluciones parciales de los estados finales. A continuación se dará una breve explicación acerca de programación dinámica.

3.1.1. Programación dinámica

La programación dinámica es un método de optimización, el cual ayuda a reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas. Este método hace uso de una *ecuación funcional*, que es una ecuación que se expresa a través de una combinación de variables independientes y funciones incógnitas, donde una o más funciones incógnitas conocidas son substituidas como argumentos de una función incógnita, que debe ser resuelta, esta ecuación se obtiene para cada problema específico, a través del uso del principio de optimalidad de Bellman que permite, con mayor o menor esfuerzo, dependiendo del caso, establecer una recurrencia.

La programación dinámica se aplica cuando el espacio de búsqueda puede ser estructurado en una serie o sucesión de estados tales que:

1. El estado inicial contiene soluciones triviales de subproblemas.
2. Cada solución parcial de estados posteriores puede ser calculada por iteración sobre un número fijo de soluciones parciales de los estados anteriores.

Un algoritmo de programación dinámica consta de 3 fases [Vinuesa, 2007]:

1. Fase de inicialización y definición recurrente del puntaje óptimo.

2. Relleno de la matriz para guardar los puntajes de subproblemas resueltos en cada iteración. Se comienza por resolver el subproblema más pequeño.

3. Se realiza un análisis reverso de la matriz para recuperar la estructura de la solución óptima.

En el caso del algoritmo de Smith-Waterman la matriz contiene dos valores: una puntuación y un apuntador, este último es un indicador de dirección que apunta en una de tres direcciones: arriba, izquierda o en diagonal.

- La primera fila y columna es incializada con ceros.
- Se calculan tres puntajes: el paridad, la disparidad horizontal y la disparidad vertical:
  - Puntaje de paridad = puntaje de la diagonal+puntaje de paridad (+1 ó -1).
  - Puntaje de disparidad horizontal = puntaje de celda izquierda+puntaje de la disparidad.
  - Puntaje de disparidad vertical = puntaje de celda superior+puntaje de la disparidad.
  - Se asigna a la nueva celda el valor más alto de los tres y una flecha en dirección de la celda vecina con mayor puntaje.

$$F(i,j) = \begin{cases} F(i-1,j) + \textit{ puntaje de la disparidad} & \ell_{1i} \neq \ell_{2j} \\ F(i-1,j-1) + s(i,j) & \ell_{1i} = \ell_{2j} \\ F(i,j-1) + \textit{puntaje de la disparidad} & \ell_{1i} \neq \ell_{2j} \end{cases}$$

Se debe tomar en cuenta que el puntaje máximo no es nunca < 0 y sólo se guardan apuntadores en las celdas si su puntaje ≥ 0

En cuanto al rastreo reverso comienza desde la celda con el puntaje más alto de la tabla y termina en una celda con puntaje 0.

A continuación se muestra un ejemplo donde se desea encontrar que regiones son similares entre las palabras coelacanth y pelican, en este caso el valor de la puntuación de la disparidad es -1 y el puntaje de paridad es 1 [Vinuesa, 2007].

En este caso el valor más alto de puntaje se encuentra en la letra N y de ahí se puede verificar que los segmentos más similares entre estas dos palabras son ELACAN-ELICAN.

3.2. Algoritmos para resolver string matching

El problema de string matching consiste en encontrar una cadena (corta), como subcadena de otra cadena (larga). Este problema es idéntico a la segunda parte del problema definido, la búsqueda del ARN de interferencia en el genoma humano, por lo que se analizarán algoritmos cuyas cotas de complejidad sean las mínimas.

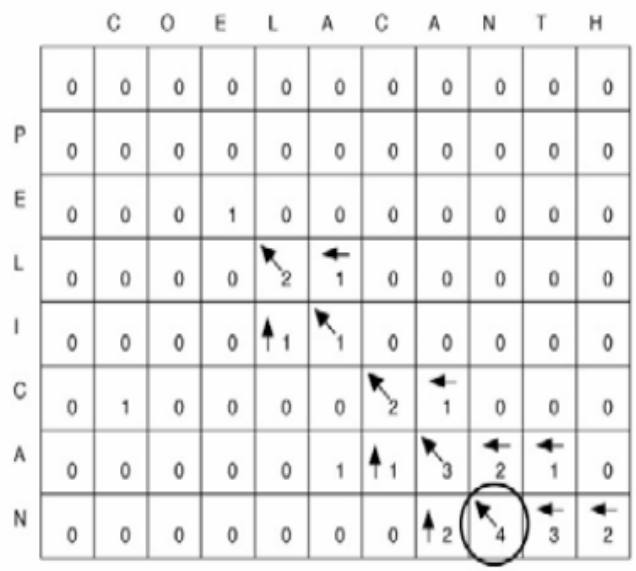

Figura 3.1: Alineamiento local entre COELACANTH y PELICAN [Vinuesa, 2007].

3.2.1. Algoritmo Boyer-Moore

En el algoritmo de Boyer-Moore regularmente no es necesario comprobar cada carácter de la cadena que es buscada, puesto que puede saltar algunos de los caracteres, por lo que puede tener un factor significativamente más bajo que muchos otros algoritmos de búsqueda. Generalmente el algoritmo es más rápido cuanto más grande es la clave que es buscada y cuando la cardinalidad del alfabeto no es tan pequeña.

El algoritmo comienza comprobando coincidencias de letras, empezando de la última letra de la cadena hacia adelante. Por ejemplo, si se tiene una cadena S con m letras, se comprueba la posición m del texto en proceso. Si encuentra la letra S_m en el texto, se mueve a la posición m-1 para ver si contiene la letra S_{m-1} de la palabra, y así sucesivamente hasta que comprueba la primera posición del texto para una S_1. Pero si por ejemplo desde el inicio la letra S_m no se encuentra en la posición m del texto, significa que no hay coincidencia para la subcadena buscada en el inicio del texto en las siguientes m posiciones, puesto que todas fallarían también, por lo que seremos capaces de saltar hacia delante y comenzar buscando una coincidencia en la posición 2m del texto [Boyer and Moore, 1977].

Esto explica por qué el rendimiento del caso promedio del algoritmo, para un texto de longitud n y patrón fijo de longitud m, es n/m: en el mejor caso, solo uno en m caracteres necesita ser comprobado. Esto también explica el resultado de que cuanto más largo es el patrón que estamos buscando, el algoritmo suele ser más rápido para encontrarlo. Pero cabe aclarar que en el peor de los casos la complejidad es $O(nm)$.

El algoritmo precalcula una tabla, la cual sirve para procesar la información que se obtiene en cada verificación fallida: se inicia en el último carácter de la subcadena y se mueve hacia la izquierda, si el carácter no está ya en la tabla, se añade y se le coloca un valor de desplazamiento igual a la distancia del último carácter hasta él. Todos los otros caracteres del alfabeto reciben un valor igual a la longitud de la cadena de búsqueda. Por lo que esto toma una cota de complejidad $O(|\Sigma|+\text{m})$ donde $|\Sigma|$ es la cardinalidad del alfabeto.

Ejemplo: Para la cadena 'coyote', la letra *'e'* tiene el valor de 0, la siguiente letra *'t'* tiene valor

de 1, la *'o'* tiene el valor de 2, la *'y'* tiene el valor de 3, de nuevo aparece la *'o'* entonces no la tomamos en cuenta y por último aparece la *'c'* que tiene el valor de 5.

La cota de complejidad del algoritmo total es de $O(|\Sigma| + m + \ell)$.

3.2.2. Algoritmo Knuth-Morris-Pratt

El algoritmo Knuth-Morris-Pratt [Knuth et al., 1977] utiliza información basada en los fallos previos, aprovechando la información de la subcadena a buscar (para esto se precalcula una tabla de fallos), para determinar donde podría darse la siguiente existencia, sin necesidad de analizar más de una vez a los caracteres de la cadena donde es la búsqueda.

En un inicio se trata de localizar la posición de comienzo de una cadena, dentro de otra, suponiendo que se tiene una tabla *'f'* precalculada, que la cadena a buscar se encuentra en un arreglo M, y la cadena donde buscamos se encuentra en un arreglo N, se comienzan a comparar letra por letra y si ocurre un fallo; es decir, no coincide alguna letra entre M y N, en vez de volver a la posición siguiente a la primera coincidencia, se salta hacia donde la tabla indique. Se utiliza un puntero de avance absoluto de N, que considera donde se compara el primer carácter de ambas cadenas, y se utiliza como un puntero relativo (sumado al absoluto) el que utiliza para el recorrido en la cadena M. Suponiendo que se tiene f(j) precalculado y como entrada un texto y un patrón, la implementación del algoritmo KMP es la siguiente:

Algoritmo 4 Algoritmo de búsqueda del oligonucleótido de interferencia en el genoma humano

```
1: procedure ALGORITMO(N, M)                    ▷ N=Texto, M=patrón
2:     int k=0;
3:     int j=0;
4:     while k < n&&j < m do    ▷ n es la longitud de la cadena N y m es la longitud de la cadena de M
5:         while j > 0&&texto[k + 1]! = patron[j + 1] do
6:             j=f[j]
7:             if texto[k+1]==patron[j+1] then
8:                 j++
9:             end if
10:            k++;
11:        end while
12:    end while
13: end procedure
```

Dado que se tienen dos ciclos anidados, se puede acotar el tiempo de ejecución por el número de veces que se ejecuta el ciclo externo, que es menor o igual a n por el número de veces que se ejecuta el ciclo interno, que es menor o igual a m. Note que el número total de veces que el ciclo interior es ejecutado es menor o igual al número de veces que se puede decrementar j, dado que el número máximo de elementos de la tabla $f(j)$ es j. Pero j comienza desde cero y es siempre mayor o igual que cero, por lo que dicho número es menor o igual al número de veces que j es incrementado, el cual es menor que n. Por lo tanto, el tiempo total de ejecución es $O(n)$. Por otra parte, k nunca es decrementado, lo que implica que el algoritmo nunca regresa a una letra anterior del texto [Bustos and Poblete, 2002].

El objetivo de la tabla (precalculada) de fallo *'F'* es no permitir que cada carácter del arreglo m sea examinado más de 1 vez. El método clave para lograr esto, consiste en haber comprobado algún trozo de la cadena n con algún trozo de la cadena m, lo que nos proporciona en que sitios potenciales puede existir una nueva coincidencia, sobre el sector analizado que indica fallo.

Partiendo de la cadena m se elabora una lista con todas las posiciones, de salto que señalan cuanto se retrocede desde la posición actual de m. Por ejemplo si la cadena a buscar es *'caerse'* y estamos examinando un texto como *'caen tras la puerta'*, cuando llegamos a la *'n'*, ya no coinciden las letras de las cadenas, puesto que en la cadena a buscar la siguiente letra es *'r'* la pregunta lógica es ¿ dónde se encuentra de nuevo (si existe) la primera letra en el texto *'caerse'* (antes del fallo), y hasta donde logra repetirse?. La respuesta a esta pregunta será el punto de salto, que en este caso se encuentra en la posición 3 (antes de la *'r'*), luego para la siguiente letra, que es justo la letra que no coincide entre las dos cadenas, se deberá de verificar si esa letra de nuevo coincide con la letra de la posición 1.

Por tanto esta tabla se confecciona con la distancia que existe desde la primera vez que aparece un tramo de la palabra en el cual, mientras sigan coincidiendo letras, se seguirá marcando la distancia, y cuando haya una ruptura de coincidencia se marca 0 o un valor previo ya calculado anteriormente, y así sucesivamente hasta terminar con el texto.

La tabla tiene sus 2 primeros valores fijados de modo que la función de fallo empieza siempre examinando el carácter número 3 del texto. La razón es que, si para el segundo carácter se marcara 1 nunca se lograría un salto pues siempre retornaría a dicho punto. El primero, por necesidad, se marca -1 pues de ese modo le es imposible regresar más atrás, sino siempre adelante.

Esto se puede realizar recursivamente. Para empezar, f(1)=0 por definición. Para calcular f(j+1) suponga que ya se tienen almacenados los valores de f(1), f(2),. . ., f(j). Se desea encontrar un i+1 tal que el (i+1)-ésimo carácter del patrón sea igual al (j+1)-ésimo carácter del patrón.

Para esto se debe cumplir que i=f(j). Si b_i+1=b_j+1, entonces f(j+1)=i+1. En caso contrario, se reemplaza i por f(i) y se verifica nuevamente la condición.

El algoritmo resultante es el siguiente (note que es similar al algoritmo KMP), donde se tiene un patrón como entrada:

Algoritmo 5 Algoritmo de búsqueda del oligonucleótido de interferencia en el genoma humano

```
1: procedure ALGORITMO(M)                                   ▷ M=patrón
2:     int f[m]
3:     int j=1
4:     while j < m do  ▷ n es la longitud de la cadena N y m es la longitud de la cadena de M
5:         i=f[j]
6:         while i > 0&&patron[i + 1]! = patron[j + 1] do
7:             j=f[j]
8:         end while
```

```
 9:        if patron[i+1]==patron[j+1] then
10:            f[j+1]=i+1
11:        else
12:            f[j+1]=0
13:        end if
14:        k++
15:    end while
16: end procedure
```

El tiempo de ejecución para calcular la función de fracaso puede ser acotado por los incrementos y decrementos de la variable i, que es $O(m)$.

Por lo tanto, el tiempo total de ejecución del algoritmo, incluyendo el preprocesamiento del patrón, es $O(m+n)$.

3.2.3. Árboles de sufijos

Los árboles de sufijos son estructuras de datos que sirven para almacenar una cadena de caracteres como información preprocesada. Esta información es útil para resolver en tiempo lineal el problema de encontrar una cadena de tamaño ℓ en una cadena más grande de tamaño m.

Debido a estas características este algoritmo supera la cota de complejidad del algoritmo de Boyer-Moore, la cual es $O(|\Sigma| + m + \ell)$, donde $|\Sigma|$ es el número de letras del alfabeto y la cota de complejidad del preproceso es $O(\ell + m)$.

Un árbol de sufijos es como un trie (un trie es una estructura de datos, que es también conocida como árbol de prefijos) que almacena todos los sufijos de una cadena, por ejemplo los sufijos de la cadena S= xabxac son: c, ac, xac, bxac, abxac, xabxac. Para la construcción del árbol, dos aristas que salen del mismo nodo no pueden tener etiquetas que empiecen con el mismo carácter (véase figura 3.2).

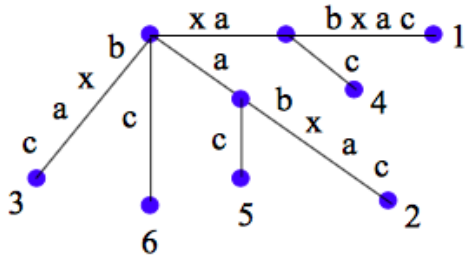

Figura 3.2: Construcción de un trie para xabxac

Para construir un árbol de sufijos existen diferentes algoritmos, uno de los toma el último carácter y lo coloca en el primer nodo junto con el carácter "$" que indica que es el terminal de una subcadena. Se toma el siguiente carácter y se revisa si ya esta en algún nodo de la primera fila de hijos. En caso de que no, se crea otro nuevo nodo hijo, en caso contrario se crean dos nodos en el segundo nivel de hijos, uno que contenga la letra o subcadena nueva junto con el símbolo "$" y otro que contenga el símbolo "$". Así se sigue el mismo proceso hasta que se

hayan colocado todos los sufijos en el árbol. Como se puede observar para buscar un sufijo a veces será necesario recorrer las letras anteriores o bien los prefijos. Por lo tanto si para el paso i-ésimo el algoritmo encuentra el prefijo más largo entre S [i.. n] y los sufijos i anteriores S [1.. n] $, ..., S [i-1.. n] $, definimos como S [i.. k] $ tal prefijo. Entonces se han realizado k + 1 comparaciones para identificar dicho prefijo. Después de eso, el resto del sufijo S [k + 1.. n]$ tiene que ser leído y asignado a la nueva edición. Por lo tanto, la cantidad total de trabajo en el paso i es $O(i)$. La complejidad general es $\Sigma_{i=1}^{n} O(i) = O(n^2)$.

Se puede lograr la implementación de un árbol de sufijos con un costo $O(n)$ en tiempo de contrucción y memoria [Ukkonen, 1995], donde n es el tamaño de la cadena a procesar o bien donde se desea buscar una subcadena. En este caso para mejorar el consumo de memoria, en lugar de usar un trie a secas, se usa un Patricia trie (un patricia trie es una estructura de datos que compacta un trie, fusionando un nodo hijo único con su padre), el cual en la inserción de cada sufijo, en el peor de los casos, hace un despliegue de un nodo, agregando otros dos. Las etiquetas de los ejes son siempre subcadenas de S, y se pueden representar mediante un intervalo, con dos índices [i, j] y por lo tanto cada nodo ocupa $O(1)$.

El algoritmo para construir un árbol de sufijos con un coste $O(n)$ se lleva a cabo construyendo una secuencia de árboles de sufijos implícitos. Un árbol de sufijos implícitos, para una cadena se obtiene borrando del árbol de sufijos cada copia del símbolo terminal $ de las etiquetas de las aristas y después eliminando cada arista que no tenga etiqueta y cada nodo interno que no tenga al menos dos hijos [Ukkonen, 1995] (véase la figura 3.3 y 3.4).

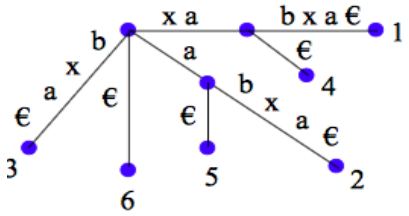

Figura 3.3: Construcción de un trie para xabxa

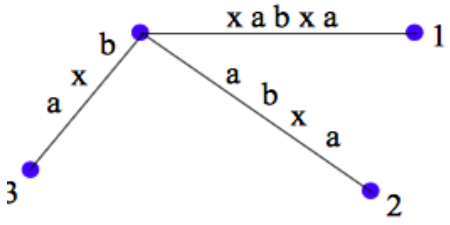

Figura 3.4: Construcción de árbol de sufijos implícito para xabxa

La búsqueda en el árbol de sufijos tomara un tiempo $O(|\Sigma|\ell + k)$, donde ℓ es el tamaño de la subcadena que se desea buscar, $|\Sigma|$ es la cantidad de letras que se encuentran en el alfabeto y es la cantidad máxima de nodos que se pueden tener en cada nivel y k es el número de veces que aparece la subcadena en el árbol.

3.3. Clasificadores por distancia mínima, diagramas de Voronoi

Para hacer búsquedas de elementos parecidos a veces se usa la regla de clasificación por distancia mínima, que es la más simple de las que se aplica. Se define de la siguiente manera: sea E un conjunto dotado de una (pseudo)métrica $d : E \times E \to \mathbb{R}$. Supongamos que se dan m prototipos $p_1, p_2, ...p_m \in E$, tales que $p_i \in c_i$, donde p_i representa a la clase $c_i (i = 1...m)$, es decir, existe un prototipo por clase. Una clasificación por distancia mínima respecto a $p_1, p_2, ..., p_m$ puede definirse para cualquier prototipo $x \in E$ como:

$x \in c_i \Leftrightarrow d(x, p_i) \leq d(x, p_j); i, j = 1...m, i \neq j$

donde d es la (pseudo)métrica definida en E.

En este tipo de clasificador, la fase de aprendizaje consiste en la elección de un buen representante de cada clase en el conjunto de prototipos que pertenecen a la misma. Normalmente el representante elegido suele ser aquel que se encuentra más centrado dentro de la distribución de los prototipos en la clase. En este clasificador la muestra x se clasifica en la clase cuyo representante se encuentra a menor distancia. Las fronteras de decisión en este tipo de espacios que separan las clases son los hiperplanos mediatrices de los segmentos formados por todos los pares de prototipos. Estas fronteras de decisión son un caso especial de clasificador lineal. El conjunto de regiones definidas por las fronteras de decisión asociadas a un clasificador de distancia mínima recibe el nombre de diagramas de Voronoi. Los diagramas de Voronoi se definen de la siguiente manera:

Dado un conjunto de puntos $p_1...p_N \in \mathbb{R}^k$, una celda de Voronoi c_i (asociada al punto p_i) se define como:

$c_i = x \in \mathbb{R}^k : d(x, p_i) < d(x, p_j), p = 1...N, j \neq i$

Todas las regiones de Voronoi de los puntos $p_i, i = 1...N$ forman una partición del espacio $\mathbb{R}^k$ de la forma $C = c_i, i = 1...N$, y se cumple que $\bigcup_{i=1}^{N} c_i = \mathbb{R}^k, c_i \bigcap_{i \neq j}, c_j = \emptyset$

3.3.1. Algoritmo por fuerza bruta

Una primera aproximación para la construcción del diagrama de Voronoi consiste en explotar la geometría de cada región de Voronoi. Por cada sitio $p_i \in P$ se construirá su región de Voronoi mediante el cálculo explícito de los n-1 semiplanos originados debido a los bisectores trazados con respecto a los demás sitios. Después, se calcula la intersección de estos n - 1 semiplanos.

Este algoritmo tiene muchas desventajas como que el cálculo explícito de los semiplanos y su intersección puede provocar problemas de precisión en la computadora. Tampoco se produce información inmediata, la cual se podría aprovechar para el vecindario de cada sitio. Para este algoritmo se tiene una complejidad de $O(n^2 \log n)$

3.3.2. Algoritmo de Fortune (barrido de recta)

El algoritmo de Fortune en honor a Steven Fortune [Fortune, 1986] también ayuda al cálculo de los diagramas de Voronoi y tiene una complejidad de $O(n \log n)$. El algoritmo de Fortune está basado en una de las técnicas clave dentro de la geometría computacional denominada

barrido de recta. La esencia de esta técnica yace en suponer que existe una recta ℓ que recorre el plano de arriba hacia abajo (o de izquierda a derecha, incluso en direcciones opuestas) y que a lo largo de su recorrido se intersecta con las estructuras que deseamos procesar. Cuando se da esta intersección, se guarda cierta información de tal forma que ayude en los cálculos. Es necesario que la información que se haya obtenido en regiones ya visitadas por la recta sea invariante. Es muy común que está técnica utilice dos tipos de estructuras de datos: cola de prioridades donde se guardan eventos que no son más que puntos donde la recta debe detenerse y un árbol binario de búsqueda donde se almacenan los elementos geométricos que se han intersectado con la recta y se necesita recordar para el procesamiento futuro. Cabe resaltar que debido a que en la computadora no se puede emular tal cual el movimiento continuo de la recta de barrido, se requiere idear una forma de discretización del movimiento de la recta que sea procesable en la computadora, de ahı'que los eventos representen está discretización.

3.4. Vecinos más próximos

Este clasificador es una generalización del clasificador por distancia mínima, ya que permite que haya más de un prototipo por clase.

En este caso la distancia entre muestras y clases se define como sigue:

$$d_{VMP}(x, c_i) = min\{d(x, p) : x \in E, p \in c_i \in C\}$$

Donde d_{VMP} es la distancia al vecino más próximo.

La muestra x se clasificará en la clase que contiene el prototipo más cercano.

En espacios de representación vectorial con producto escalar las funciones discriminantes asociadas a un clasificador basado en la regla del vecino más próximo son lineales a intervalos [Tou and Gonzalez, 1974]. Como consecuencia, se pueden obtener funciones discriminantes arbitrariamente complejas, lo que supone una ventaja clara sobre otros clasificadores (paramétricos o no paramétricos) que sólo producen ciertos tipos de fronteras (lineales, cuadráticas, etc).

La fase de aprendizaje para un clasificador por el vecino más próximo consiste simplemente en la obtención del conjunto de objetos considerados como prototipos de cada clase, el cual constituye todo el conocimiento a priori del sistema. Este diccionario puede obtenerse de forma directa o se puede refinar usando técnicas de edición y condensado [Devijver and Kittler, 1982, Hart, 1968]. Con estas técnicas se intenta reducir el número de prototipos manteniendo o incluso mejorando las prestaciones del clasificador original.

3.5. Búsqueda de vecinos en espacios vectoriales

La búsqueda de vecinos en espacios vectoriales suele basar su funcionamiento en estructuras de datos que utilizan una representación vectorial de los puntos, una de las es el árbol binario de búsqueda d dimensional, más conocido como k-dtree. La idea que conlleva la búsqueda en una estructura de este tipo es dividir el dominio de búsqueda (dando un valor de partición) donde los puntos se encuentran ordenados en dos subdominios. La comparación con el valor de

partición nos dirá en cuál de los dos subdominios se encuentra cada punto (los menores a la izquierda y los mayores a la derecha).

El kdtree se basa en la misma idea original de dividir el espacio de representación $\mathbb{R}^k$ en dos espacios disjuntos a partir de un hiperplano perpendicular al vector correspondiente a una de las coordenadas k. Estos hiperplanos H, dados como $H = x \in \mathbb{R}^k : x_j = h$, define dos mitades, R_L y R_R como $R_L = x \in \mathbb{R}^k : x_j < h$ y $R_R = x \in \mathbb{R}^k : x_j > h$. Cada hiperplano viene representado por dos valores: 1) el índice del eje de coordenada ortogonal, j, también llamado discriminador y 2) la localización del hiperplano, h, en dicho eje. Cualquier prototipo x puede ser localizado con respecto al plano de partición realizando una comparación del tipo $x_j \leq h$ (ver si la coordenada j del punto x se encuentra a la derecha o la izquierda de la partición). Cada uno de los espacios obtenidos puede ser sucesivamente dividido por hiperplanos perpendiculares a los ejes de coordenadas. De esta forma, empezando en la raíz para el eje de coordenadas 1, los hiperplanos de partición para los siguientes niveles se obtienen cíclicamente desde este valor hasta k.

Fischer y Patrick presentaron un preproceso en el que ordenan el conjunto de prototipos y los almacenan en una lista. La ordenación realizada es utilizada junto con la desigualdad triangular para evitar el cálculo de muchas distancias entre los prototipos y la muestra a clasificar [Fischer and Patrick, 1970]. Otro grupo de métodos propuestos utilizan estructuras arborescentes para almacenar los prototipos [Fukunaga and Narendra, 1975, Kalantari and McDonald, 1983]. En este caso el preproceso consiste en una descomposición jerarquizada del conjunto de prototipos en subconjuntos disjuntos. En estos métodos, la búsquedase basa en técnicas de ramificiación y poda para reducir el número de prototipos examinados. Un ejemplo, es el algoritmo propuesto por Fukunaga y Narendra [Fukunaga and Narendra, 1975], donde cada nodo p del árbol T, obtenido por descomposición del conjunto de prototipos P, representa un subconjunto $S_p \subset P$. La información real que se almacena en cada nodo es el número de prototipos pertenecientes al subconjunto, N_p, el centroide de dicho conjunto m_p y el radio del nodo definido como:

$$r_p = max\{d(x_i, m_p) : x_i \in S_p\}$$

.

Donde un nodo interno p puede ser eliminado si se cumple:

$$D_{min} + r_p < d(x, m_p)$$

donde D_{min} es la distancia del prototipo más cercano a la muestra hasta el momento y x es la muestra a clasificar. La regla de eliminación para los nodos hoja se aplica individualmente a cada uno de los prototipos. Cada uno de estos prototipos es eliminado si se cumple:

$$D_{min} + d(x_i, m_p) < d(x, m_p)$$

3.6. Algoritmo AESA

El AESA (Approximating Eliminating Search Algorithm) es un algoritmo que consta principalmente de dos fases: aproximación y eliminación. Asumiendo un orden arbitrario en el conjunto

de prototipos P, este método utiliza una matriz triangular de $\mid P \mid (\mid P \mid -1)/2$ distancias entre prototipos, calculadas en la fase de preproceso. Esta matriz de distancias será utilizada en las dos fases: 1) para guiar una búsqueda dinámica rápida por aproximación sucesiva hacia el prototipo más cercano, y 2) para dar soporte a las reglas de eliminación [Vidal, 1986].

En la estrategia para la búsqueda se selecciona un prototipo aproximadamente cercano a la muestra, se calcula la distancia entre ambos y si esta distancia es menor que la del más cercano hasta el momento se actualiza el vecino más próximo. En la segunda fase (eliminación), usando la desigualdad triangular, se eliminan para posteriores búsquedas todos los prototipos que no pueden estar más cercanos a la muestra que el actualmente seleccionada como tal. El procedimiento termina cuando el conjunto de prototipos queda vacío.

Si $u \in U$, y n es el actual vecino más cercano a la muestra x ($d(x,n) \leq d(x,u) \forall u \in U$), la desigualdad triangular propia de una métrica se aplica para obtener las siguientes desigualdades [Vidal, 1994]: $\mid d(x,q) - d(q,p) \mid \leq d(x,p), \forall p \in P, q \in U, x \in E$ y como consecuencia: $max \mid d(x,q) - d(q,p) \mid \leq d(x,p)$, donde $U \in E$.

Esta característica permite definir la función de cota inferior de la distancia de la siguiente forma:$g_U(p) = max \mid d(x,q) - d(q,p) \mid, \forall p \in P$ con $g_U(p) = 0$ si $U = \emptyset$.

Así pues, aplicando las propiedades métricas, se obtienen las condiciones suficientes para que un prototipo p no pueda estar más cercano a la muestra x que el más cercano, n, hasta el momento (ver figura 3. 5): Todo prototipo p tal que $g_U(p) > d(x,n)$ es descartado para calcular su distancia a la muestra x, por lo tanto, puede ser eliminado de P.

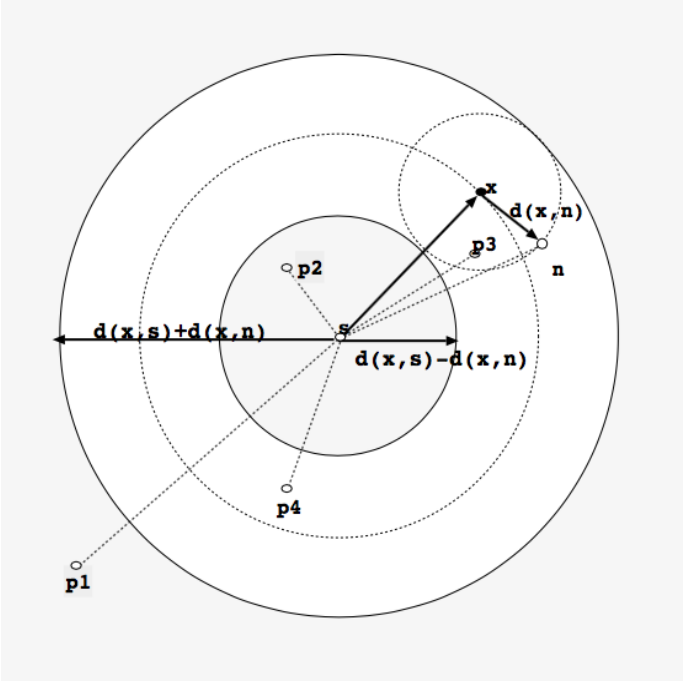

Figura 3.5: Criterio de eliminación en el plano euclideano. Todos aquellos prototipos que se encuentran en la zona sombreada no pueden estar más cercanos a la muestra x que el actual vecino más próximo n, y por lo tanto pueden ser eliminados [Vidal, 1986].

Una vez eliminados los prototipos que cumplen la condición, el procedimiento continúa seleccionando un nuevo prototipo no eliminado como candidato para calcular su distancia a la muestra. Se aplica la regla de eliminación de la forma indicada anteriormente para los prototipos no eliminados y la condición de parada se cumple cuando todos los prototipos han sido eliminados,

bien porque se les ha aplicado la regla de eliminación o porque han sido seleccionados para calcular su distancia a la muestra.

Capítulo 4

Aportaciones

En el capítulo 2 se describieron algoritmos para resolver las tres partes del problema planteado haciendo uso de algoritmos de fuerza bruta, lo que conlleva a un alto coste en su tiempo de ejecución y complejidad. Si se analiza con más detalle cada parte del problema y se hace uso de distintas técnicas o métodos computacionales, se pueden idear algoritmos más eficientes y por consiguiente con una complejidad y tiempo de ejecución menor. A continuación se describen algunas aportaciones propias de algoritmos para la resolución del problema.

4.1. Algoritmo para la búsqueda del oligonucleótido de interferencia

Para resolver el problema descrito en la sección 2.3, se pueden obtener conjuntos de subcadenas de longitud ℓ de las cadenas A y O respectivamente. Si uno observa dos subcadenas consecutivas del conjunto de subcadenas de la cadena O, por ejemplo O_i y O_{i+1}, donde la letra i representa el elemento i-ésimo del conjunto de las subcadenas de la cadena O, podrá notar que la mayoría de letras entre ellas son comunes; a saber, la subcadena $o_j o_{(j+1)} ... o_{(j+\ell-1)}$ y la subcadena $o_{(j+1)} o_{(j+2)} ... o_{(j+\ell-1)} o_{(j+\ell)}$, tienen $\ell - 1$ letras en común. Por lo que al hacer el cálculo del número de letras en común entre las subcadenas de A_i y Oi, y A_{i+1} y O_{i+1}, existe una repetición inecesaria de cálculos para obtener el número de coincidencias de letras; es decir, el número de letras que coinciden entre la subcadena $o_j o_{(j+1)} ... o_{(j+\ell-1)}$ y $a_j a_{(j+1)} ... a_{(j+\ell-1)}$, será similar a el número de coincidencias entre $o_{(j+1)} o_{(j+2)} ... o_{(j+\ell)}$ y $a_{(j+1)} a_{(j+2)} ... a_{(j+\ell)}$, excepto por las letras con índice j y con índice $j + \ell$, por lo tanto se pueden omitir cálculos. Gracias a esta cualidad del problema se puede hacer uso de la programación dinámica, la cual ayuda a resolver problemas de manera óptima usando recursividad. Esto ayudó a desarrollar un algoritmo más eficiente y con esto se redujo el tiempo en la búsqueda del oligonucleótido de interferencia.

Esta solución es similar al algoritmo de alineamiento local de Smith-Waterman, el cual ayuda a encontrar que segmentos de dos cadenas son más parecidos. Por lo que el algoritmo que proponemos es una modificación al algoritmo de Smith-Waterman, que contempla distintos puntajes para la disparidad y para las coincidencias, además de hacer uso de la función de la Tm.

Para la solución que se propuso del problema, se crea una matriz, donde las filas están representadas por las letras de la cadena O y las columnas están representadas por las letras de la cadena A.

Se propone un puntaje (+1) para las letras que coincidan entre una fila y una columna y se propone otro (0) para las que no coincidan.

Para dar solución al problema planteado, se requiere hacer uso de las fórmulas de Tm de Wallace o de Marmur descritas en el capítulo 1. Para aplicar estas fórmulas, es necesario conocer el número de letras que coinciden entre una subcadena de O y una subcadena de A. En especifico, es necesario conocer el número de coincidencias de las letras A y T, y el número de coincidencias de las letras G y C, por lo que en cada celda de la matriz se asignaran dos valores: uno con las coincidencias de las letras A y las letras T, llamado C_{AT}, y otro con las coincidencias de las letras G y las letras C, al que llamaremos C_{GC}. Entonces la función recursiva para la matriz queda de la siguiente manera:

$$F(i,j) = (0,0) \; donde \; i \times j = 0$$

$$F(i,j) = (C_{AT}, C_{GC}) \; donde \; i = 1,...,\ell_O \; y \; j = 1,...,\ell_A$$

$$C_{AT}(i,j) = \begin{cases} C_{AT}(i-1,j-1)+1 & o_i = a_j \in \{A,T\} \\ C_{AT}(i-1,j-1) & o_i \neq a_j \in \{A,T\} \end{cases}$$

$$C_{GC}(i,j) = \begin{cases} C_{GC}(i-1,j-1)+1 & o_i = a_j \in \{G,C\} \\ C_{GC}(i-1,j-1) & o_i \neq a_j \in \{G,C\} \end{cases}$$

A continuación se muestra un ejemplo en la figura 4.1, en la cual se tiene una matriz, donde cada una de sus celdas i, j contiene el número de coincidencias de las letras A y las letras T, y el número de coincidencias de las letras G y las letras C, entre dos cadenas.

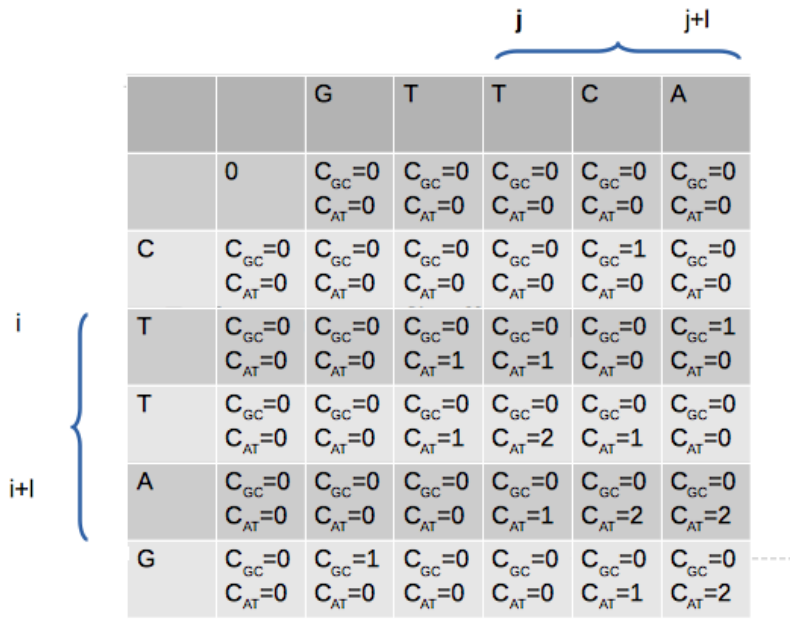

Figura 4.1: Matriz rellena con valores de coincidencias

Dada está matriz, se puede observar que también es posible conocer el número de letras que coinciden entre la subcadena del índice i al índice $i+\ell$ de la cadena O y la subcadena del índice j al índice $j+\ell$ de la cadena A, donde i y j son valores mayores a uno.

Esto se logra conociendo los valores C_{AT} y C_{GC} de la celda $i+\ell, j+\ell$ y restando los valores de las coincidencias C_{AT} y C_{GC} en la celda $i-1, j-1$. Este ajuste se debe a que se restan el número de coincidencias de letras que no corresponden a la subcadena evaluada. Por ejemplo, supóngase que la longitud de O y A es igual a 5 y se desea obtener el número de coincidencias C_{AT} y C_{GC} de las últimas subcadenas de tamaño 3 (véase figura 4.2, donde $i=3$). Se puede observar que en la fila 5 de la matriz se tienen los valores de C_{AT} y C_{GC} de las 5 letras entre las cadenas O y A, por lo que se realiza una resta de estos valores con los valores C_{AT} y C_{GC} de las celdas que corresponden a las primeras dos letras entre ambas cadenas, para obtener el número de coincidencias de las últimas tres letras entre O y A.

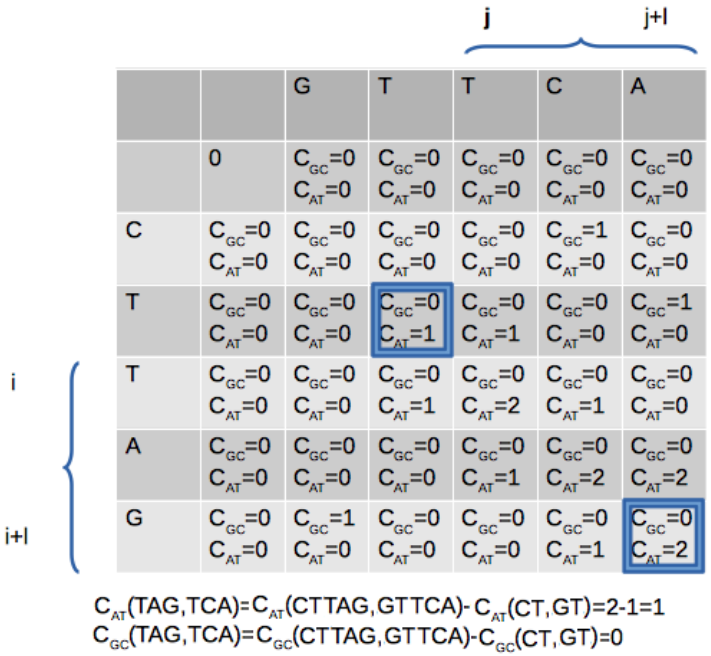

Figura 4.2: Ejemplo de valores de coincidencias de subcadenas TAG y TCA. En la parte inferior se muestra la formula para calcular las coincidencias entre las subcadenas TAG y TCA

Una vez conocido el tamaño de la subcadena que representa el oligonucleótido de interferencia, se pueden obtener los valores del número de coincidencias C_{AT} y C_{GC} entre dos subcadenas de las cadenas O y A, y por lo tanto se pueden evaluar en la función de Tm Tm, la cual corresponde a la fórmula de Wallace o Marmur.

El siguiente paso consiste en conocer el valor máximo de la función Tm evaluando las coincidencias C_{AT} y C_{GC} de cada una de las subcadenas de la cadena O con todas las subcadenas de la cadena A.

Los valores de la función Tm entre una subcadena de tamaño ℓ de O y una subcadena de tamaño ℓ de A, se pueden calcular en cada celda, a partir de la fila y la columna ℓ, así que para conocer el valor máximo de Tm de una subcadena O con todas las subcadenas de A, se compara la función Tm evaluada con los valores de coincidencias C_{AT} y C_{GC} de cada celda a partir de (ℓ, ℓ) y se selecciona la que tiene el mayor valor en cada fila.

En la siguiente figura se muestra un ejemplo, donde se calcula el valor de la función Tm evaluada

con los valores de C_{AT} y C_{GC} de cada celda del ejemplo de la figura 4.2. Del lado derecho se muestra una columna la cual contiene los valores máximos de Tm de cada una de las filas:

		G	T	T	C	A	
	0	0	0	0	0	0	Tm Mayor
C	0	T_m=0	T_m=0	T_m=0	T_m=4	T_m=0	4
T	0	T_m=0	T_m=2	T_m=2	T_m=0	T_m=0	2
T	0	T_m=0	T_m=2	T_m=4	T_m=2	T_m=0	4
A	0	T_m=0	T_m=0	T_m=2	T_m=4	T_m=4	4
G	0	T_m=4	T_m=0	T_m=0	T_m=2	T_m=2	4

Figura 4.3: Valores de Tm mayor para cada fila

Para conocer el valor de Tm para subcadenas de tamaño ℓ, como se había explicado anteriormente, es necesario hacer un ajuste para C_{AT} y C_{GC} como el siguiente: sea M la matriz que contiene los valores de C_{AT} y C_{GC} en cada celda y sea ℓ= longitud de la subcadena, entonces $C_{AT} = M.C_{AT}(i,j) - M.C_{AT}(i-\ell, j-\ell)$ y $C_{GC} = M.C_{GC}(i,j) - M.C_{GC}(i-\ell, j-\ell)$ donde i y j son mayores que 3. Por lo que la evaluación en la formula de wallace, queda de la siguiente manera:

$$Tm = 2(M.C_{AT}(i,j) - M.C_{AT}(i-\ell, j-\ell)) + 4(M.C_{GC}(i,j) - M.C_{GC}(i-\ell, j-\ell))$$

		G	T	T	C	A	
	0	0	0	0	0	0	Tm Mayor
C	0	-	-	-	-	-	-
T	0	-	-	-	-	-	-
T	0	-	-	T_m=4	T_m=2	T_m=0	4
A	0	-	-	T_m=2	T_m=4	T_m=4	4
G	0	-	-	T_m=0	T_m=2	T_m=2	2

Figura 4.4: Valores de Tm mayor de filas para subcadenas de tamaño 3

A continuación se muestra una figura que contiene un ejemplo para calcular la Tm_{max} para subcadenas de tamaño 3, usando los valores de coincidencias de la figura 4.2. Para las primeras dos filas y dos columnas no se calcula la Tm debido a que el tamaño de la subcadena es 3, por o que se evaluará la función de Tm a partir de $i = 3$ y $j = 3$. Del lado derecho, se muestra una columna que contiene los valores máximos de la función Tm para cada fila.

Siguiendo la definición del problema, es necesario que se asigne una variable que indica cuantas veces se repite el valor de Tm_{max} en cada fila.

Una vez que en cada fila de la matriz se obtenga el valor máximo de Tm acompañado de su número de repeticiones, el siguiente paso es obtener el valor mínimo de todas las Tm_{max} de las filas. Para ahorrar tiempo de procesamiento, esto se realiza de la siguiente manera: cada que se obtiene Tm_{max} en una fila, se compara con el valor anterior de Tm_{max} y se seleccionan los elementos que tienen el valor mínimo, este proceso se repite hasta que se termine el número de filas, como consecuencia se tendrá un conjunto S_{min} de elementos. De este conjunto de valores mínimos S_{min} se seleccionan los que tengan el valor de repeticiones mínimo. En el ejemplo de la figura 4.5, el valor mínimo de todas las Tm_{max} es 2 y sólo se encuentra en una fila, aunque se repita dos veces, este es el mejor candidato.

		G	T	T	C	A	
	0	0	0	0	0	0	Tm Mayor,repeticiones
C	0	-	-	-	-	-	-
T	0	-	-	-	-	-	-
T	0	-	-	T_m=4	T_m=2	T_m=0	4,1
A	0	-	-	T_m=2	T_m=4	T_m=4	4,2
G	0	-	-	T_m=0	T_m=2	T_m=2	2,2

Figura 4.5: Evaluación de mejor Tm para subcadenas de tamaño 3

4.1.1. Complejidad

Se describe el algoritmo 6 para la búsqueda del mejor candidato de oligonucleótido de interferencia, se analiza su complejidad y tiempo de ejecución.

Algoritmo 6 Algoritmo de búsqueda del mejor candidato de oligonucleótido de interferencia.

```
1:  procedure ALGORITMO(A, O)                ▷ A=gen actuador, O=gen objetivo
2:      structure Matriz do
3:          int C_AT
4:          int C_GC
5:      end structure
6:      Matriz M[ℓ_A + 1][ℓ_O + 1]          ▷ Matriz de las Tm entre las subcadenas de O y de A
7:      for i=1:i≤ ℓ_A:i++ do   ▷ Inicialza en ceros los elementos de la primera columna de M
8:          M.C_AT[0][i] = 0
9:          M.C_GC[0][i] = 0
10:     end for
11:     for i=1:i≤ ℓ_O:i++ do          ▷ Inicialza en ceros los elementos de la primera fila de M
12:         M.C_AT[i][0] = 0
```

```
13:         M.C_GC[i][0] = 0
14:     end for
15:     for i=1:i≤ ℓ_O:i++ do                          ▷ i es el número de subcadena de O
16:         for j=1:j≤ ℓ_A:j++ do                      ▷ j es el número de subcadena de A
17:             if O_i==A_j then
18:                 switch A_i do
19:                     case A
20:                         M.C_AT[i+1][j+1] = M.C_AT[i][j] + 1
21:                     end case
22:                     case C
23:                         M.C_GC[i+1][j+1] = M.C_GC[i][j] + 1
24:                     end case
25:                     case G
26:                         M.C_GC[i+1][j+1] = M.C_GC[i][j] + 1
27:                     end case
28:                     case T
29:                         M.C_AT[i+1][j+1] = M.C_AT[i][j] + 1
30:                     end case
31:                 end switch
32:             end if
33:             if ( thenj ≥ ℓ)
34:                 C_GC = M.C_GC[i+1, j+1] − M.C_GC[i−ℓ, j−ℓ]   ▷ C_GC de una subcadena
35:                 C_AT = M.C_AT(i+1, j+1) − M.C_AT(i−ℓ, j−ℓ)   ▷ C_AT de una subcadena
36:                 Tm = 64,9 + 41 · (C_GC − 16,4)/(C_AT + M.C_GC)
37:                 if Tm > Tm_max then              ▷ Selecciona el valor de Tm más grande
38:                     Tm_max = Tm
39:                     indiceJ = j
40:                     contador = 0
41:                 else if Tm = Tm_max then
42:                     contador + +                 ▷ Repeticiones de valor de Tm más grande
43:                 end if
44:             end if
45:         end for
46:         if Tm_max < mejorTm then                 ▷ Selecciona el valor de Tm_max más pequeño
47:             mejorTm = Tm_max
48:             numeroTm = contador
49:         else if Tm_max = mejorTm then
50:             if contador < numeroTm then          ▷ Selecciona el valor de Tm_max que se repita
    menos veces
51:                 indiceI = i
52:             end if
53:         end if
54:     end for
55: end procedure
```

Para calcular el tiempo de ejecución del algoritmo, se toma el número de operaciones involucradas en el mismo. El tiempo para acceder a un arreglo unidimensional es constante, esto es debido a que cada elemento del arreglo es una dirección de memoria, por lo que para acceder

a un elemento del arreglo se lleva a cabo en una operación.

Para acceder a una celda de una arreglo bidimendional (matriz) el tiempo de ejecución también es constante, esto es debido a que un arreglo bidimensional de f filas y c columnas, se traduce a un arreglo unidimensional de longitud fc, donde de la posición 1 a la posición c del arreglo unidimensional abarca los elementos de la primera fila de la matriz y de la posición $c+1$ a la posición $2c$ del arreglo unidimensional abarca los elementos de la segunda fila de la matriz, y así sucesivamente. Entonces para acceder a la posición (i, j) de la matriz, sólo es necesario conocer la posición de esa celda en el arreglo unidimensional, está se puede calcular fácilmente con la siguiente operación $ic + j$, una vez obtenido el resultado, sólo hay que dirigirse a la dirección de memoria de esa celda y obtener el dato deseado, por lo que acceder a un elemento de un arreglo bidimensional es llevado a cabo en una sola operación.

El algoritmo descrito anteriormente tienen dos ciclos con dos operaciones que inicializan la primer fila y la primer columna en ceros de la matriz M, como se menciono anteriormente, estas operaciones se realizan en tiempo constante, por lo que se tienen un total de $[(2\ell_A) + (2\ell_O)]$ operaciones. Seguido de esto se tiene un ciclo con 12 operaciones anidado en otro ciclo con 4 operaciones adicionales. Así el algoritmo se lleva a cabo en el siguiente número de operaciones:

$$2\ell_A + 2\ell_O + \ell_O(12\ell_A + 4) = 2\ell_A + 6\ell_O + 12\ell_O\ell_A$$

Diremos que $\ell_O = n$, además $\ell_O < \ell_A$, por lo que $\ell_A = kn$ para alguna constante k, entonces la expresión $2kn + 6n + 12\ell_O\ell_A$ se puede representar como una función de n de la siguiente manera

$$f(n) = 2kn + 6n + 12kn \cdot n = 2kn + 6n + 12kn^2$$

Se sabe que $2kn < kn^2$ y $6n < kn^2$, por lo tanto $2kn + 6n + 12kn^2 < 3(12kn^2)$, es decir, $f(n) < 3(12kn^2)$, pero $3 \cdot 12k$ es constante por lo tanto la cota de complejidad es

$$O(n^2)$$

4.2. Algoritmo de búsqueda del oligonucleótido en el genoma

En el capítulo 2 se hizo un análisis del algoritmo de fuerza bruta para buscar si una cadena α (oligonucleótido de interferencia) es subcadena de otra cadena G (los ARNm maduros del genoma humano). Para resolver este problema es necesario comparar cada una de las letras de la cadena de α con las letras de G, esto tiene un costo de $O(n^2)$.

Para resolver este problema es necesario comparar cada una de las letras de G con las letras del oligonucleótido de interferencia para asegurarse si ésta última es subcadena de G, así que no se espera tener un algoritmo de menos de ℓ_g operaciones. Para que reducir la cota de complejidad $O(n^2)$ del algoritmo del capítulo 2 que solucionaba este problema, se pensó en una manera de comparar la cadena α, no letra por letra sino como un paquete, con cada paquete de subcadenas de tamaño ℓ de la cadena G. Una opción es pensar en obtener el hash de la cadena α y de cada una de las subcadenas de G, y entonces comparar los hash. Se pensó construir un hash $H(S)$ de la siguiente manera:

$$H(S) = \sum_{i=1}^{\ell} 4^i L(\ell_i)$$

donde S es la subcadena que se va a evaluar, ℓ_i es la i-ésima letra de una subcadena y $L(\ell_i)$ es una función definida de la siguiente manera

$$L(\ell_i) = \begin{cases} 1 & \ell_i = A \\ 2 & \ell_i = C \\ 3 & \ell_i = G \\ 4 & \ell_i = T \end{cases}$$

Si obtener los hash de las subcadenas de G tomara tiempo lineal, se tendría una complejidad total lineal, desafortunadamente obtener el hash para una cadena requiere sumar cada valor de la letra de la cadena, por lo que es necesario recorrer nuevamente el número total de letras ℓ de la subcadena , lo que conlleva nuevamente una complejidad $O(n^2)$.

Para evitar este problema de recorrer las (ℓ) letras de las subcadenas de G por cada letra de la misma, se puede hacer uso de que las subcadenas consecutivas de G tienen letras en común, pero debido a que en el cálculo del hash planteado se hace uso de la posición de la letra de la cadena, no se puede aprovechar la ventaja de que las subcadenas consecutivas de G tengan letras en común, por lo que se descartó el cálculo del hash planteado. Si g_i es la letra i-ésima de G, entre $g_{i+1}g_{i+2}...g_{i+\ell}$ y $g_{i+2}g_{i+3}...g_{i+\ell+1}$ se tienen las siguientes letras en común: $g_{i+2}g_{i+3}...g_{i+\ell}$. Por lo tanto si a la primera cadena le quitamos la letra g_{i+1} y le agregamos $g_{i+\ell+1}$ se tendría la segunda cadena. Esto significa hacer un corrimiento, pero hacer corrimientos con letras conlleva varias operaciones, a diferencia de los corrimientos con números binarios que se llevan a cabo en una única operación. En este esquema las subcadenas serán representadas como números binarios, donde cada número binario será una concatenación de letras, cada una representada por dos dígitos binarios de la siguiente manera: A=00, C=01, G=10, T=11, por ejemplo, la cadena AGT, tiene un valor binario de 001011 (véase figura 4.6).

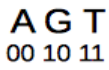

Figura 4.6: Asignación binaria a una cadena

Para obtener el valor binario de la primer subcadena de tamaño ℓ de la cadena G, se toman los valores binarios de las primeras ℓ letras de la cadena G. El corrimiento binario nos permite tener ventajas para la obtención de las siguiente subcadenas, por ejemplo para la siguiente subcadena sólo se hace un corrimiento binario en dos bits (que representan una letra) hacia la izquierda, y se concatena el valor binario de la siguiente letra.

Ejemplo: si se tiene la cadena AGTACGT, donde las subcadenas deben ser de tamaño 5, entonces los valores binarios para las dos subcadenas consecutivas AGTAC, GTACG serán los siguientes:

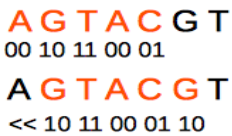

Figura 4.7: Asignación binaria a subcadenas

Para conocer si una cadena α es subcadena de la cadena G basta hacer una comparación del valor binario de la cadena α, con el valor binario de cada subcadena de G, si estos valores binarios son iguales, entonces se tiene que α es subcadena de G.

4.2.1. Complejidad

Se describe el algoritmo 7 para la búsqueda del oligonucleótido de interferencia en el genoma, donde se hace uso de valores binarios y su respectiva complejidad y tiempo de ejecución:

Algoritmo 7 Algoritmo de búsqueda del oligonucleótido de interferencia en el genoma

```
1: procedure ALGORITMO(G, α)   ▷ G=ARNm maduros del genoma, α=oligonucleótido de interferencia
2:     long binario_α
3:     long binario_G
4:     long mascara=(long)pow(2,(longitud*2))-1   ▷ máscara para tomar la cantidad de dígitos binarios de la longitud de la cadena.
5:     for i=1:i≤ ℓ:i++ do   ▷ i es la i-ésima letra de α
6:         switch α_i do
7:             case A
8:                 binario_α = binario_α00   ▷ Concatena el valor binario de A
9:             end case
10:            case C
11:                binario_α = binario_α01   ▷ Concatena el valor binario de C
12:            end case
13:            case G
14:                binario_α = binario_α10   ▷ Concatena el valor binario de G
15:            end case
16:            case T
17:                binario_α = binario_α11   ▷ Concatena el valor binario de T
18:            end case
19:        end switch
20:    end for
21:    for i=1:i≤ ℓ_G:i++ do   ▷ i es la i-ésima letra de G
22:        switch G_i do
23:            case A
24:                binario_G = binario_G00   ▷ Concatena el valor binario de A
25:            end case
26:            case C
27:                binario_G = binario_G01   ▷ Concatena el valor binario de C
28:            end case
29:            case G
30:                binario_G = binario_G10   ▷ Concatena el valor binario de G
31:            end case
32:            case T
33:                binario_G = binario_G11   ▷ Concatena el valor binario de T
34:            end case
35:        end switch
36:        if  theni ≥ ℓ
37:            if binario_α = binario_G then
38:                Imprime el oligonucleótido α se encuentra en el genoma humano
39:            end if
40:            binario_G = binario_G << 2   ▷ Corrimiento de dos dígitos a la izquierda
```

41: $binario_G = binario_G \& mascara$ ▷ Máscara binaria que limita el número de dígitos binarios necesarios
42: **end if**
43: **end for**
44: **end procedure**

Para calcular el tiempo de ejecución del algoritmo anterior se tienen dos ciclos, un ciclo con 3 operaciones, y otro con 8 operaciones. Por lo que el algoritmo se lleva a cabo en el siguiente número de operaciones:

$$3\ell_\alpha + 8\ell_g$$

Diremos que $n = \ell_\alpha$, además $\ell_\alpha < \ell_g$, por lo que $\ell_g = kn$ para alguna constante k, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = 3n + 8kn$$

Pero como $3n < 8kn$, entonces $3n + 8kn < 2(8kn)$, es decir, $f(n) < 2(8kn)$, pero $2(8k)$ es una constante, por lo tanto la cota de complejidad es

$$O(n)$$

Haciendo una comparación entre el algoritmo de desarrollo propio y el de Boyer-Moore, el algoritmo de desarrollo propio en el peor de los casos tiene una complejidad de $O(n)$, mientras que el algoritmo de Boyer-Moore en el peor de los casos toma un tiempo de (nm). Comparado el algoritmo propuesto con el algoritmo de Knuth-Morris-Pratt y los árboles de sufijos, se puede observar que es más fácil la implementación del algoritmo que se propuso, a pesar de que la cota de complejidad es la misma $O(n)$, en los tres algoritmos.

Cabe aclarar que el algoritmo de desarrollo propio debido a su naturaleza binaria, depende de la cantidad de bits que se puedan almacenar en una variable en memoria, actualmente la variable con mayor cantidad de bits es el tipo de dato long, donde se pueden almacenar 64 bits. También hay que tomar en cuenta que el número de bits de cada letra depende de la cantidad de letras que tiene un alfabeto, es decir, 1 bit puede representar un alfabeto de dos letras, 2 bits uno de 4 letras, 3 bits uno de 8 letras y así sucesivamente. Suponiendo que tenemos un alfabeto de 4 letras, cada letra podrá ser representada con 2 bits, por lo tanto a lo más en variables de 64 bits se podrán representar subcadenas de 32 letras.

Para saber que tan bueno en tiempo de procesamiento es el algoritmo propuesto con respecto al algoritmo de Boyer-Moore y el de Knuth-Morris-Pratt, se implementaron cada uno de los algoritmos en lenguaje C y se midió el promedio del tiempo en el que terminaba la ejecución de cada algoritmo cambiando el tamaño de las cadenas de entrada.

En la siguiente tabla se muestran los resultados en milisegundos de la ejecución de cada uno de los algoritmos, para la búsqueda de una cadena de 5 letras en cadenas de distintos tamaños.

Algoritmo/ No. de letras	ℓ=10000	ℓ=100000	ℓ=1000000	ℓ=10000000	ℓ=100000000
Boyer-Moore	0.6385	5.50	54.69	565.4	5845.57
Knuth-Morris-Pratt	0.60016	5.397	54.12	547.43	5769.20
Algoritmo propio	0.5899	5.124	51.8	563.61	5651.97

Para la búsqueda de una cadena de 25 letras en cadenas de 10000, 100000, 1000000, 10000000, 100000000 se tiene la siguiente tabla de tiempos de ejecución en milisegundos.

Algoritmo/ No. de letras	ℓ=10000	ℓ=100000	ℓ=1000000	ℓ=10000000	ℓ=100000000
Boyer-Moore	0.6952	5.900	59.53	592.37	6049.781
Knuth-Morris-Pratt	0.67205	5.8263	58.69	589.13	5945.43
Algoritmo propio	0.6239	5.402	57.4853	569.63	5808.766

Debido a las características del problema, se tiene un alfabeto de tamaño 4 y se quiere comparar una subcadena de tamaño no mayor a 28, este algoritmo resuelve perfectamente el problema y no es necesario hacer ninguna modificación, pero en caso de que fueran necesarios más bits, se tendría que hacer una modificación al algoritmo, en donde en vez de requerir un sólo dato de tipo long sería necesario tener un arreglo de variables de tipo long y la cota de complejidad cambiaría en un factor constante que dependería del tamaño del arreglo de las variables de tipo long.

4.3. Algoritmo para minimizar el silenciamiento génico fuera del objetivo

Como se explico en el capítulo 2 es esencial minimizar el riesgo de silenciamiento génico fuera del objetivo, por lo que es necesario considerar una tolerancia de incompatibilidad. Para esto es indispensable evaluar la Tm en todas las posibles subcadenas de los ARNm maduros del genoma humano, con el fin de asegurar que no existe un sitio en el genoma humano donde la Tm sea más alta o igual a 20°C más la Tm máxima entre el oligonucleótido propuesto y una subcadena del gen $PPAR\gamma$.

Haciendo un análisis de la función Tm usando la regla de Marmur, se puede hacer una aproximación del número de letras iguales que son necesarias entre dos cadenas, para obtener la Tm que cumple con la tolerancia de incompatibilidad. Si suponemos que la cadena que se va a analizar esta conformada por la misma cantidad de letras AT's y CG's. Entonces en la fórmula de Marmur, $Tm = 64{,}9 + 41 \cdot (yG + zC - 16{,}4)/(wA + xT + yG + zC)$, se tiene $yG + zC = wA + xT = \ell$. Sustituyendo en la ecuación se tiene

$$Tm = 64{,}9 + 41 \cdot (L - 16{,}4)/(2L)$$

Despejando L se tiene

$$L = 672{,}4/(84{,}5 - Tm)$$

Si suponemos una Tm máxima de 37°C entre el oligonucleótido de interferencia propuesto y una subcadena de $PPAR\gamma$, entonces la mínima Tm para que se cumpla la tolerancia de incompatibilidad debe de ser de 37°C + 20°C=57°C. Sustituyendo el valor en la función $L(Tm)$, se obtiene $L(57) = 24{,}4$, es decir, es necesario que existan aproximadamente 25 letras iguales entre las dos cadenas evaluadas en la función de Tm, para alcanzar la tolerancia de incompatibilidad. Por lo que, si se considera un oligonucleótido de interferencia de 27 letras, son necesarios 2 letras desiguales o menos para que sea muy probable que un oligonucleótido de interferencia pueda silenciar un gen.

Como se había mencionado en el capítulo 2, en la exposición del problema era necesario hacer algunas comparaciones con todas las letras de las subcadenas de los ARNm maduros del genoma humano, pero poniendo en práctica la última idea expuesta, se puede concluir que una vez que

se encuentren más de cierto número de letras desiguales entre las dos cadenas comparadas (en el ejemplo anterior 2 letras desiguales) ya no será necesario realizar comparaciones con el resto de letras de la subcadena. A continuación se analizara el pseudocódigo, donde se agrega una variable que cuenta el número de letras desiguales, supongamos η letras, para que una vez que este número sobrepase la cantidad de desigualdades analizadas, ya no se sigan realizando más comparaciones con las letras restantes de la subcadena. En caso de que no se sobrepase la cantidad de desigualdades se hará una evaluación de la función de Tm.

4.3.1. Complejidad

Se expone el algoritmo 8 para minimizar el riesgo de silenciamiento génico fuera del objetivo, se analiza su complejidad y tiempo de ejecución.

Algoritmo 8 Minimizar el riesgo de silenciamiento génico fuera del objetivo

```
1: procedure ALGORITMO(G, α)   ▷ G=ARNm maduros del genoma, α=oligonucleótido de interferencia
2:     int desiguales=0
3:     for i=1:i≤ ℓ:i++ do
4:         switch α_i do                                   ▷ i es la i-ésima letra de α
5:             case A
6:                 binario_α = binario_α00                 ▷ Concatena el valor binario de A
7:             end case
8:             case C
9:                 binario_α = binario_α01                 ▷ Concatena el valor binario de C
10:            end case
11:            case G
12:                binario_α = binario_α10                 ▷ Concatena el valor binario de G
13:            end case
14:            case T
15:                binario_α = binario_α11                 ▷ Concatena el valor binario de T
16:            end case
17:        end switch
18:    end for
19:    for i=1:i≤ ℓ_G − ℓ:i++ do                           ▷ i es la i-ésima letra de G
20:        C_AT = 0
21:        C_GC = 0
22:        for j=1:ℓ&desiguales ≤ 2:j++ do                 ▷ i es la i-ésima letra de G
23:            if g_i ≠ α_j then
24:                switch g_i do
25:                    case A
26:                        C_AT + +
27:                    end case
28:                    case C
29:                        C_GC + +
30:                    end case
31:                    case G
32:                        C_GC + +
33:                    end case
```

```
                    case T
                        C_AT++
                    end case
                end switch
            else
                desiguales++
            end if
        end for
        if desiguales ≤ 2 then
            Tm = 64,9 + 41 · (C_GC − 16,4)/(C_AT + M.C_GC)
            if Tm > Tm_anterior then          ▷ Selecciona el valor de Tm más grande
                Tm_anterior = Tm
            end if
            if Tm_anterior > 20 + Tm_oligo then
                Imprimir "Es muy problable que el oligonucleótido de interferencia inhiba la expresión génica fuera del objetivo".
            end if
        end if
    end for
end procedure
```

Para calcular el tiempo de ejecución del algoritmo anterior se tiene que hay dos ciclos, uno de los ciclos tiene ℓ operaciones y el otro tiene 5 operaciones y está anidado dentro de otro ciclo con 6 operaciones adicionales. Por lo que el algoritmo en el peor de lo casos se lleva a cabo en el siguiente número de operaciones:

$$4\ell + (\ell_g - \ell)(4\ell + 6) = 4\ell + 4\ell\ell_g - 4\ell^2 + 6\ell_g - 6\ell = 4\ell\ell_g - 4\ell^2 + 6\ell_g - 2\ell$$

Pero en el mejor caso donde las primeras η letras son diferentes y alcanzan el máximo número de letras desiguales permitidas (en este caso 2), el número de operaciones se reducirá, debido a que no se ejecutaran estas en el segundo ciclo en todas las ℓ letras, esto hará que el tiempo de ejecución se reduzca.

Para el cálculo de la cota complejidad diremos que $n = \ell$ y $n > 1$, además $\ell < \ell_g$, por lo que $\ell_g = kn$ para alguna constante k, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = 4nkn - 4n^2 + 6kn - 2n = 4kn^2 - 4n^2 + 6kn - 2n$$

Pero como $-4n^2 + 6kn - 2n < 4kn^2$, entonces $-4n^2 + 6kn - 2n < 2(4kn^2)$, es decir, $f(n) < 2(4kn^2)$, pero $2(4k)$ es una constante, por lo tanto la cota de complejidad es

$$O(n^2)$$

Analizando más detalladamente el algoritmo, con el fin de reducir el tiempo de ejecución, se pensó en una idea análoga al algoritmo propuesto para la búsqueda del oligonucleótido de interferencia en el genoma humano. Pero debido a que en una sola operación no se pueden hallar cuántas letras desiguales hay se tienen entre dos cadenas, entonces se planteó, que en vez de tener una cadena de caracteres representada por un valor binario, tener números binarios que representaran dos letras. Tener un valor binario que represente dos letras ahorrara más

operaciones que comparar letra por letra, más adelante se explica por que un valor binario debe representar dos letras y no más. Al igual que en el algoritmo anterior A=00, C=01, G=10, T=11 y la concatenación de dos letras será la concatenación de los números binarios, por ejemplo CG=0110.

El proceso que se sigue es realizar una XOR entre los números binarios que representan dos letras (4 bits, es decir, 2 bits por cada letra) de cada cadena, en caso de que la operación XOR arroje como resultado un cero, quiere decir que las dos letras de ambas cadenas son iguales, en caso contrario habrá una o dos letras desiguales. Para conocer si no hay ninguna desigualdad entre dos letras se realiza una operación, en caso contrario se realizarán dos operaciones. Para saber el número de letras desiguales que existen entre las dos subcadenas de dos letras después de realizar la operación XOR, se contempla que si existe una letra desigual el número binario resultante deberá de tener sus dos primeros números en cero o bien sus dos últimos números en cero, es decir el número binario deberá ser 1, 2, 3, 4, 8 ó 12, o bien el número binario deberá ser $\geq$4 ó 8 ó 12. Una vez que el número de letras desiguales sobrepase la cantidad de desigualdades requerida para cumplir con la tolerancia de incompatibilidad, ya no es necesario que se realicen más comparaciones con las letras restantes de la subcadena. En caso de que no se sobrepase la cantidad de desigualdades se hará una evaluación de la función de Tm.

Se expone el segundo algoritmo 9 para minimizar el riesgo de silenciamiento génico fuera del objetivo, su complejidad y su tiempo de ejecución:

Algoritmo 9 Minimizar el riesgo de silenciamiento génico fuera del objetivo

```
1: procedure ALGORITMO(G, α)          ▷ G=ARNm del genoma, α=oligonucleótido de interferencia
2:     int desiguales=0
3:     long mascara2=0x03             ▷ máscara para tomar sólo 4 dígitos binarios
4:     long mascara=2^(2ℓ) − 1        ▷ máscara para los dígitos binarios necesarios
5:     for i=1:i≤ ℓ:i++ do
6:         switch α_i do              ▷ i es la i-ésima letra de α
7:             case A
8:                 binario_α = binario_α00      ▷ Concatena el valor binario de A
9:             end case
10:            case C
11:                binario_α = binario_α01      ▷ Concatena el valor binario de C
12:            end case
13:            case G
14:                binario_α = binario_α10      ▷ Concatena el valor binario de G
15:            end case
16:            case T
17:                binario_α = binario_α11      ▷ Concatena el valor binario de T
18:            end case
19:        end switch
20:        switch g_i do              ▷ i es la i-ésima letra de G
21:            case A
22:                binario_G = binario_G00      ▷ Concatena el valor binario de A
23:            end case
24:            case C
25:                binario_G = binario_G01      ▷ Concatena el valor binario de C
26:            end case
```

```
27:             case G
28:                 binario_G = binario_α10                    ▷ Concatena el valor binario de G
29:             end case
30:             case T
31:                 binario_G = binario_G11                    ▷ Concatena el valor binario de T
32:             end case
33:         end switch
34:     end for
35:     for i=ℓ + 1:i≤ ℓ_G:i++ do                              ▷ i es la i-ésima letra de G
36:         C_AT = 0
37:         C_GC = 0
38:         binario2_G = binario_G
39:         binario2_α = binario2_α
40:         for j=1:ℓ/2&desiguales≤ 2:j++ do                   ▷ i es la i-ésima letra de G
41:             variable_xor = (binario_G&mascara)XOR(binario_α&mascara)
42:             if variable_xor ≠ 0 then
43:                 if variable_xor ≤ 4 ó variable_xor = 8 ó variable_xor = 12 then
44:                     desiguales++
45:                 else
46:                     desiguales=desiguales+2
47:                 end if
48:                 binario2_G >> 4   ▷ Corrimiento en 4 del valor binario de la subcadena de G
49:                 binario2_α >> 4   ▷ Corrimiento en 4 del valor binario de la subcadena de α
50:                 binario2_G = binario2_G&mascara2
51:                 binario2_α = binario_α&mascara2
52:             end if
53:         end for
54:         if desiguales≤ 2 then          ▷ se espera que sean mínimos los casos de 1 a 3
55:             for i=1:i≤ ℓ:i++ do                            ▷ Para calcular la Tm
56:                 if g_(i − ℓ − 1) ≠ α_j then
57:                     switch G_i do
58:                         case A
59:                             C_AT + +
60:                         end case
61:                         case C
62:                             C_GC + +
63:                         end case
64:                         case G
65:                             C_GC + +
66:                         end case
67:                         case T
68:                             C_AT + +
69:                         end case
70:                     end switch
71:                 end if
72:             end for
73:             Tm = 64,9 + 41 · (C_GC − 16,4)/(C_AT + M.C_GC)
74:             if Tm > Tm_anterior then          ▷ Selecciona el valor de Tm más grande
75:                 Tm_anterior = Tm
76:             end if
```

77: **if** $Tm_{anterior} > 20 + Tm_{oligo}$ **then**
78: Imprimir "Es muy problable que el oligonucleótido de interferencia inhiba la expresión génica fuera del objetivo".
79: **end if**
80: **end if**
81: $binario_G = binario_G << 2$ ▷ Corrimiento de dos dígitos a la izquierda
82: **switch** g_i **do** ▷ i es la i-ésima letra de G
83: **case** A
84: $binario_G = binario_G 00$ ▷ Concatena el valor binario de A
85: **end case**
86: **case** C
87: $binario_G = binario_G 01$ ▷ Concatena el valor binario de C
88: **end case**
89: **case** G
90: $binario_G = binario_\alpha 10$ ▷ Concatena el valor binario de G
91: **end case**
92: **case** T
93: $binario_G = binario_G 11$ ▷ Concatena el valor binario de T
94: **end case**
95: **end switch**
96: $binario_G = binario_G \& mascara$ ▷ limitar el número de dígitos binarios necesarios
97: **end for**
98: **end procedure**

Representar la cadena α en número binario conlleva 3ℓ operaciones, donde ℓ es el número de letras de α, así mismo se representan las primeras ℓ letras de la cadena G en número binario, por lo que el número de operaciones del primer ciclo es 6ℓ

Después cada dos letras codificadas en binario de la cadena G, se van a comparar con dos letras codificadas de la cadena α, esto mientras el número de letras que sean desiguales entre una subcadena de G y la cadena α sea menor a una cantidad de desigualdades especificada. Por lo que se tiene un ciclo que recorre las $\ell_G - \ell$ letras de G, y dentro se realizan primero 4 operaciones y un ciclo con máximo $8\ell/2$ operaciones (la cantidad de $ell/2$ es debida a que dos letras están contenidas en un número binario), pero en realidad se ejecutarán menos operaciones por la condición de que el número de desigualdades deberá ser menor o igual a dos. Si se cumple la condición de que el existen menos de dos o dos letras desiguales entonces se prosigue a hacer $4\ell + 5$ operaciones, pero para el caso del problema que se resuelve aquí el número de veces que se cumple esta condición es de máximo tres veces no de $\ell_G - \ell$ veces. Después de esta condición se prosigue a realizar otras 5 operaciones. Por lo que el número total de operaciones: $6\ell + (\ell_G - \ell)(8\ell/2 + 9) + 3(4\ell + 5) = 6\ell + 8\ell_G\ell/2 - 8\ell^2/2 + 9\ell_G - 9\ell + 12\ell + 15 = 4\ell_G\ell - 4\ell^2 + 9\ell_G + 9\ell + 15$

Para el cálculo de la cota complejidad diremos que $n = \ell$ y $n > 1$, además $\ell < \ell_g$, por lo que $\ell_g = kn$ para alguna constante k, entonces esta expresión se puede definir como una función de n de la siguiente manera

$$f(n) = 4knn - 4n^2 + 9kn + 9n + 15 = 4kn^2 - 4n^2 + 9kn + 9n + 15$$

Pero como $-4n^2 < 4kn^2$, $9kn < 4kn^2$, $9n < 4kn^2$ y $15 < 4kn^2$, entonces $-4n^2 + 6kn - 2n < 4(4kn^2)$, es decir, $f(n) < 4(4kn^2)$, pero $4(4k)$ es una constante, por lo tanto la cota de

complejidad es

$$O(n^2)$$

Comparando el algoritmo de fuerza bruta y los propuesto aquí se puede observar que los algoritmos propuestos realizan un menor número de operaciones ya que discriminan información. Entre los dos algoritmos descritos anteriormente se puede observar que el número de pasos que realiza el segundo se reduce si se realiza un preprocesamiento de las cadenas, porque dentro del algoritmo se eliminarían las operaciones necesarias para averiguar la letra y su conversión a número binario, por lo tanto el número de operaciones serían $(\ell_G - \ell)(8\ell/2 + 5) + 3(5) = 4\ell\ell_G - 4\ell^2 + 5\ell_G + 15 - 5\ell$ que es menor que el número de pasos que realizará el primero $4\ell\ell_g - 4\ell^2 + 6\ell_g - 2\ell$, por lo que concluimos que el segundo algoritmo es la mejor opción.

Poniendo en ejecución el segundo algoritmo y el algoritmo de fuerza bruta, usando los siguientes datos: una Tm máxima= 46.7, número máximo de letras desiguales=3 (haciendo uso de la fórmula de Marmur con una Tm máxima de 46.7,para obtener el número de letras necesarias que cumplan con la tolerancia de incompatibilidad), número de letras de la cadena=27, se obtiene que el algoritmo de fuerza bruta realiza 1,628,751,213 operaciones, mientras que el algoritmo propuesto realiza 135,538,594 operaciones. Esto muestra que el algoritmo propuesto ejecuta 1,493,212,619 operaciones menos que el de fuerza bruta, es decir, el algoritmo de fruerza bruta realiza 12 veces más operaciones que el algoritmo propuesto.

4.4. Aplicación realizada

Se realizó una aplicación conformada por varios módulos, que implementan los algoritmos descritos anteriormente. En está sección se exponen el paradigma de programación y el lenguaje usados, así como también se describe la interfaz y el funcionamiento de la aplicación y los resultados obtenidos al ponerla en uso.

4.4.1. Paradigma de la aplicación:POO

Existen diferentes paradigmas de programación cada uno de estos contiene un conjunto de estándares que representan una manera de organizar el pensamiento para afrontar distintos problemas. En este caso en especial se hará uso del paradigma orientado a objetos debido a que cuenta con las características de modularidad, la cual permite subdividir a la aplicación en partes más pequeãs, las cuales son independientes entre ellas, esto permite que el programa sea extensible y se pueda adaptar fácilmente a otros productos de software. También permite la reutilización de estas partes o módulos dentro de la misma aplicación. Esta característica de modularidad tiene un gran aporte a nuestro objetivo general.

La programación orientada a objetos cuenta con otras características como:

Abstracción: denota las características esenciales de un objeto. Un objeto es una entidad que tiene propiedades o atributos y tiene distintos comportamientos (a los que se les llamara métodos), los cuales responden a diferentes eventos. Estos objetos sirven como agentes abstractos, que pueden informar, cambiar de estado y comunicarse con otros objetos. La abstracción ayuda a armar un conjunto de clases que permiten modelar la realidad o el problema que se quiere atacar.

Encapsulamiento: Reúne todos los elementos que pueden considerarse pertenecientes a una misma entidad.

Polimorfismo: Objetos con distintos comportamientos pueden tener el mismo nombre.

Herencia: Se puede tener una jerarquía debido a la relación que tienen las clases entre sí. Esto permite que los objetos hereden las propiedades y el comportamiento de todas las clases a las que pertenecen.

4.4.2. Lenguaje de programación

Existen diversos lenguajes de programación orientados a objetos pero para la implementación de los algoritmos expuestos anteriormente se hizo uso de los lenguajes java y objective C.

Java es un lenguaje que permite que los desarrolladores de aplicaciones escriban el programa y lo ejecuten en cualquier computadora no importando el sistema operativo (esto es conocido en inglés como WORA, o "write once, run anywhere"), lo cual se logra haciendo uso de una máquina virtual. Objective C no necesita una máquina virtual en computadoras con sistema operativo Mac OS, sin embargo es necesaria cuando se tienen otros sistemas operativos. La desventaja es que acoplar Objective C con una máquina virtual resulta ser más difícil que java.

La ventaja de usar Objective C, es que la aplicación que se desarrollará es parte de una aplicación más general desarrollada para Mac OS y por lo tanto la aplicación no necesitará hacer uso de una máquina virtual.

4.4.3. Interfaz gráfica

Con el fin de que la aplicación sea amigable para el usuario se encuentra divida en varios módulos, que permiten realizar varias consultas sin importar que otro módulo este abierto. A continuación se explicara como esta conformada la aplicación:

La pantalla principal que se muestra al ejecutar la aplicación es el módulo: "Encuentra oligonucleótido de interferencia". En la parte superior izquierda de la computadora se encuentra un menú, como el que se muestra en la figura 4.8. El menú cuenta con las siguientes opciones :Actividades, Método Tm, Cargar archivos, Descargar, Editar y Ayuda.

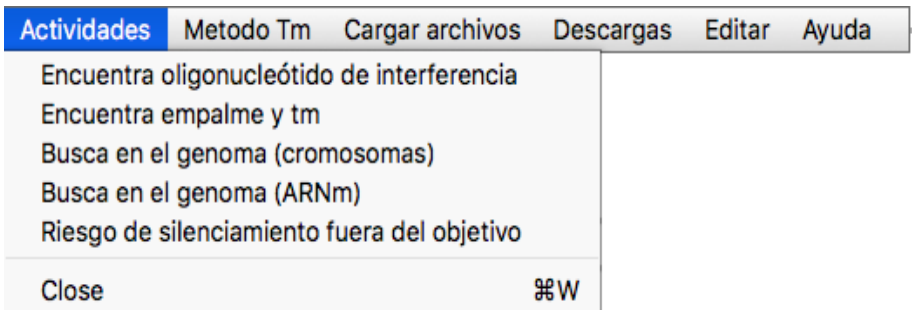

Figura 4.8: Menú de la aplicación

4.4.3.1. Actividades

Dentro del menú actividades, se encuentran todas las tareas posibles que el usuario puede realizar dentro de la aplicación como son: Encuentra oligonucleótido de interferencia, encuentra empalme y Tm, busca en el genoma (cromosomas), busca en el genoma (ARNm maduro) y riesgo de silenciamiento fuera del objetivo. Cada uno de estos módulos se describen adelante.

4.4.3.2. Encuentra oligonucleótido de interferencia

Está es la pantalla principal de la aplicación (véase figura 4.9), cuya función es desplegar el oligonucleótido de interferencia más adecuado para el silenciamiento génico dadas las características anteriormente expuestas en la solución del problema.

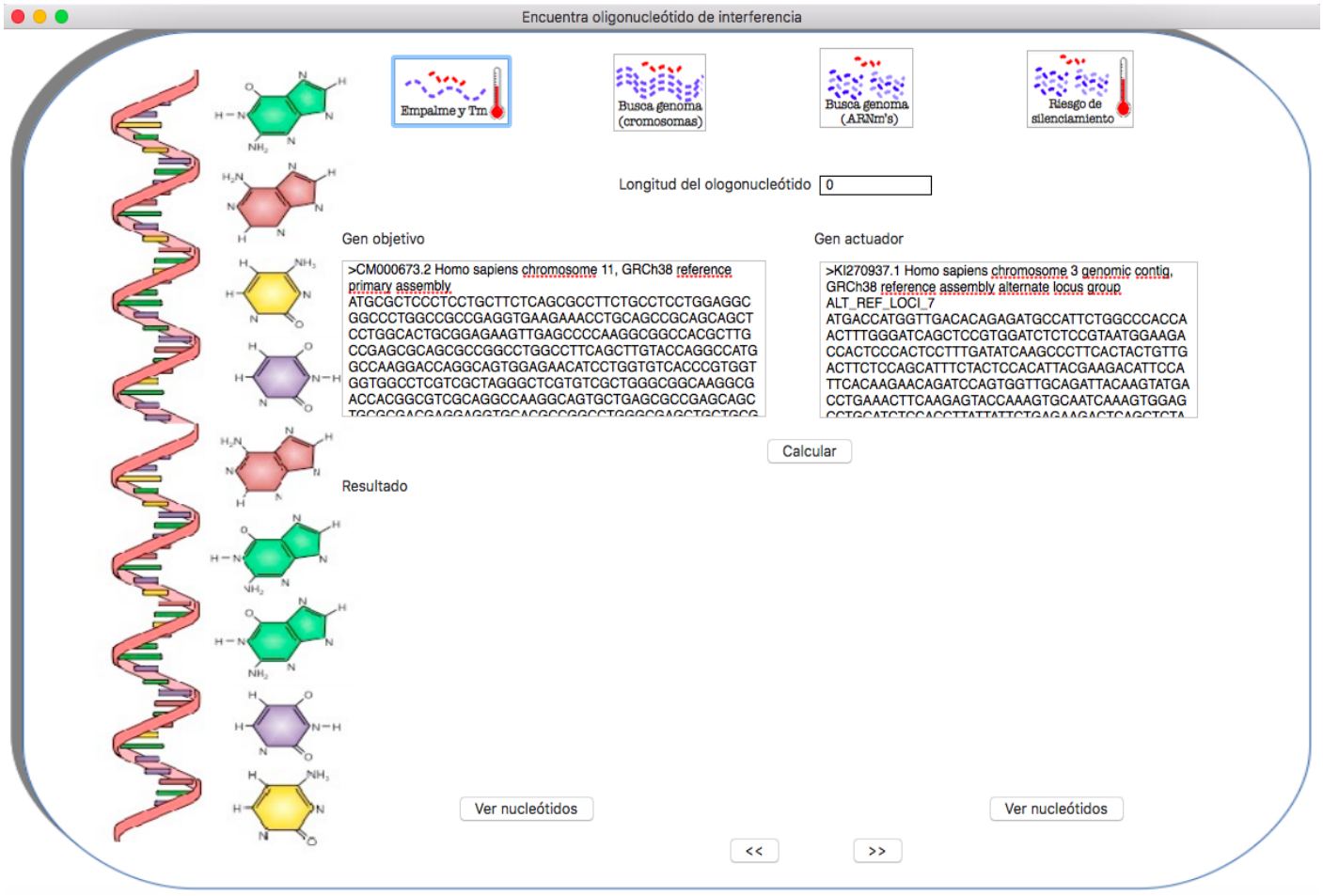

Figura 4.9: Ventana de la aplicación “Encuentra oligonucleótido de interferencia”

En la parte superior de la pantalla se encuentran botones que ayudan al usuario a acceder a los otros módulos de la aplicación como son: Encuentra empalme y Tm, Busca en el genoma (cromosomas), Busca en el genoma (ARNm maduro) y Riesgo de silenciamiento fuera del objetivo. Debajo de los botones, la pantalla esta compuesta por un recuadro cuya etiqueta es longitud del oligonucleótido, en el cual el usuario debe de colocar el número de letras que requiere para su oligonucleótido de interferencia.

Después se encuentran dos recuadros de mayor tamaño en uno se colocaran los nucleótidos, representados como letras ‘A’(adenina), ‘C’(citosina), ‘G’(guanina, ‘T’(Timina) del gen objetivo,

que es el gen que se desea silenciar, mientras que en el otro recuadro se colocarán los nucleótidos del gen actuador, este gen será acoplado al oligonucleótido sensor, el cual se podrá desnaturalizar en el cuerpo humano en presencia del gen objetivo. En la aplicación esos recuadros tienen los siguientes valores por omisión.

El gen objetivo es Hsp47:>CM000673.2 Homo sapiens chromosome 11, GRCh38 reference primary assembly / ATGCGCTCCCTCCTGCTTCTCAGCGCCTTCTGCCTCCTGGAGGCG
GCCCTGGCCGCCGAGGTGAAGAAACCTGCAGCCGCAGCAGCTCCTGGCACTGCG
GAGAAGTTGAGCCCCAAGGCGGCCACGCTTGCCGAGCGCAGCGCCGGCCTGGCC
TTCAGCTTGTACCAGGCCATGGCCAAGGACCAGGCAGTGGAGAACATCCTGGTGT
CACCCGTGGTGGTGGCCTCGTCGCTAGGGCTCGTGTCGCTGGGCGGCAAGGCGA
CCACGGCGTCGCAGGCCAAGGCAGTGCTGAGCGCCGAGCAGCTGCGCGACGAGG
AGGTGCACGCCGGCCTGGGCGAGCTGCTGCGCTCACTCAGCAACTCCACGGCGCG
CAACGTGACCTGGAAGCTGGGCAGCCGACTGTACGGACCCAGCTCAGTGAGCTTC
GCTGATGACTTCGTGCGCAGCAGCAAGCAGCACTACAACTGCGAGCACTCCAAGA
TCAACTTCCGCGACAAGCGCAGCGCGCTGCAGTCCATCAACGAGTGGGCCGCGCA
GACCACCGACGGCAAGCTGCCCGAGGTCACCAAGGACGTGGAGCGCACGGACGG
CGCCCTGCTAGTCAACGCCATGTTCTTCAAGCCACACTGGGATGAGAAATTCCAC
CACAAGATGGTGGACAACCGTGGCTTCATGGTGACTCGGTCCTATACCGTGGGTG
TCATGATGATGCACCGGACAGGCCTCTACAACTACTACGACGACGAGAAGGAAAA
GCTGCAAATCGTGGAGATGCCCCTGGCCCACAAGCTCTCCAGCCTCATCATCCTC
ATGCCCCATCACGTGGAGCCTCTCGAGCGCCTTGAAAAGCTGCTAACCAAAGAGC
AGCTGAAGATCTGGATGGGGAAGATGCAGAAGAAGGCTGTTGCCATCTCCTTGC
CCAAGGGTGTGGTGGAGGTGACCCATGACCTGCAGAAACACCTGGCTGGGCTGG
GCCTGACTGAGGCCATTGACAAGAACAAGGCCGACTTGTCACGCATGTCAGGCAA
GAAGGACCTGTACCTGGCCAGCGTGTTCCACGCCACCGCCTTTGAGTTGGACACA
GATGGCAACCCCTTTGACCAGGACATCTACGGGCGCGAGGAGCTGCGCAGCCCCA
AGCTGTTCTACGCCGACCACCCCTTCATCTTCCTAGTGCGGGACACCCAAAGCGG
CTCCCTGCTATTCATTGGGCGCCTGGTCCGGCCTAAGGGTGACAAGATGCGAGAC
GAGTTATAG

El gen actuador es PPAR γ :>KI270937.1 Homo sapiens chromosome 3 genomic contig, GRCh38 reference assembly alternate locus group ALT_REF_LOCI_7 / ATGACCATGGTTGACACA
GAGATGCCATTCTGGCCCACCAACTTTGGGATCAGCTCCGTGGATCTCTCCGTAA
TGGAAGACCACTCCCACTCCTTTGATATCAAGCCCTTCACTACTGTTGACTTCTC
CAGCATTTCTACTCCACATTACGAAGACATTCCATTCACAAGAACAGATCCAGTG
GTTGCAGATTACAAGTATGACCTGAAACTTCAAGAGTACCAAAGTGCAATCAAAG
TGGAGCCTGCATCTCCACCTTATTATTCTGAGAAGACTCAGCTCTACAATAAGCC
TCATGAAGAGCCTTCCAACTCCCTCATGGCAATTGAATGTCGTGTCTGTGGAGAT
AAAGCTTCTGGATTTCACTATGGAGTTCATGCTTGTGAAGGATGCAAGGGTTTCT
TCCGGAGAACAATCAGATTGAAGCTTATCTATGACAGATGTGATCTTAACTGTCG
GATCCACAAAAAAAGTAGAAATAAATGTCAGTACTGTCGGTTTCAGAAATGCCTT
GCAGTGGGGATGTCTCATAATGCCATCAGGTTTGGGCGGATGCCACAGGCCGAG
AAGGAGAAGCTGTTGGCGGAGATCTCCAGTGATATCGACCAGCTGAATCCAGAG
TCCGCTGACCTCCGGGCCCTGGCAAAACATTTGTATGACTCATACATAAAGTCCT
TCCCGCTGACCAAAGCAAAGGCGAGGGCGATCTTGACAGGAAAGACAACAGACA
AATCACCATTCGTTATCTATGACATGAATTCCTTAATGATGGGAGAAGATAAAAT
CAAGTTCAAACACATCACCCCCCTGCAGGAGCAGAGCAAAGAGGTGGCCATCCGC
ATCTTTCAGGGCTGCCAGTTTCGCTCCGTGGAGGCTGTGCAGGAGATCACAGAGT
ATGCCAAAAGCATTCCTGGTTTTGTAAATCTTGACTTGAACGACCAAGTAACTCT

CCTCAAATATGGAGTCCACGAGATCATTTACACAATGCTGGCCTCCTTGATGAAT
AAAGATGGGGTTCTCATATCCGAGGGCCAAGGCTTCATGACAAGGGAGTTTCTAA
AGAGCCTGCGAAAGCCTTTTGGTGACTTTATGGAGCCCAAGTTTGAGTTTGCTGT
GAAGTTCAATGCACTGGAATTAGATGACAGCGACTTGGCAATATTTATTGCTGTC
ATTATTCTCAGTGGAGACCGCCCAGGTTTGCTGAATGTGAAGCCCATTGAAGACA
TTCAAGACAACCTGCTACAAGCCCTGGAGCTCCAGCTGAAGCTGAACCACCCTGA
GTCCTCACAGCTGTTTGCCAAGCTGCTCCAGAAAATGACAGACCTCAGACAGATT
GTCACGGAACACGTGCAGCTACTGCAGGTGATCAAGAAGACGGAGACAGACATG
AGTCTTCACCCGCTCCTGCAGGAGATCTACAAGGACTTGTACTAG

Después de presionar el botón "calcular", en la parte inferior se muestra el resultado de la ejecución de la implementación del algoritmo, que se compone de un texto que contiene el oligonucleótido de interferencia óptimo, seguido de la Tm máxima entre él y el gen que funge como objetivo, y su Tm máxima entre él y el gen que funge como actuador. Seguido de esto, en la parte izquierda se muestra el gen objetivo, representando sus oligonucleótidos de distintos colores con el símbolo '|', donde el color rojo es el nucleótido adenina, el color negro es el nucleótido citocina, el color azul es el nucleótido guanina y el color amarillo es el nucleótido timina. El tramo que esta subrayado con verde fosforescente es la parte donde el oligonucleótido de interferencia es complementario al gen, es decir, donde va a empalmar el oligonucleótido y el gen. A la par en la parte derecha se muestra el gen que funge como actuador, representado de la misma manera que el gen que funge como objetivo, y las partes donde el gen y el oligonucleótido de interferencia se acoplan están subrayadas con verde fosforescente (véase figura 4.10).

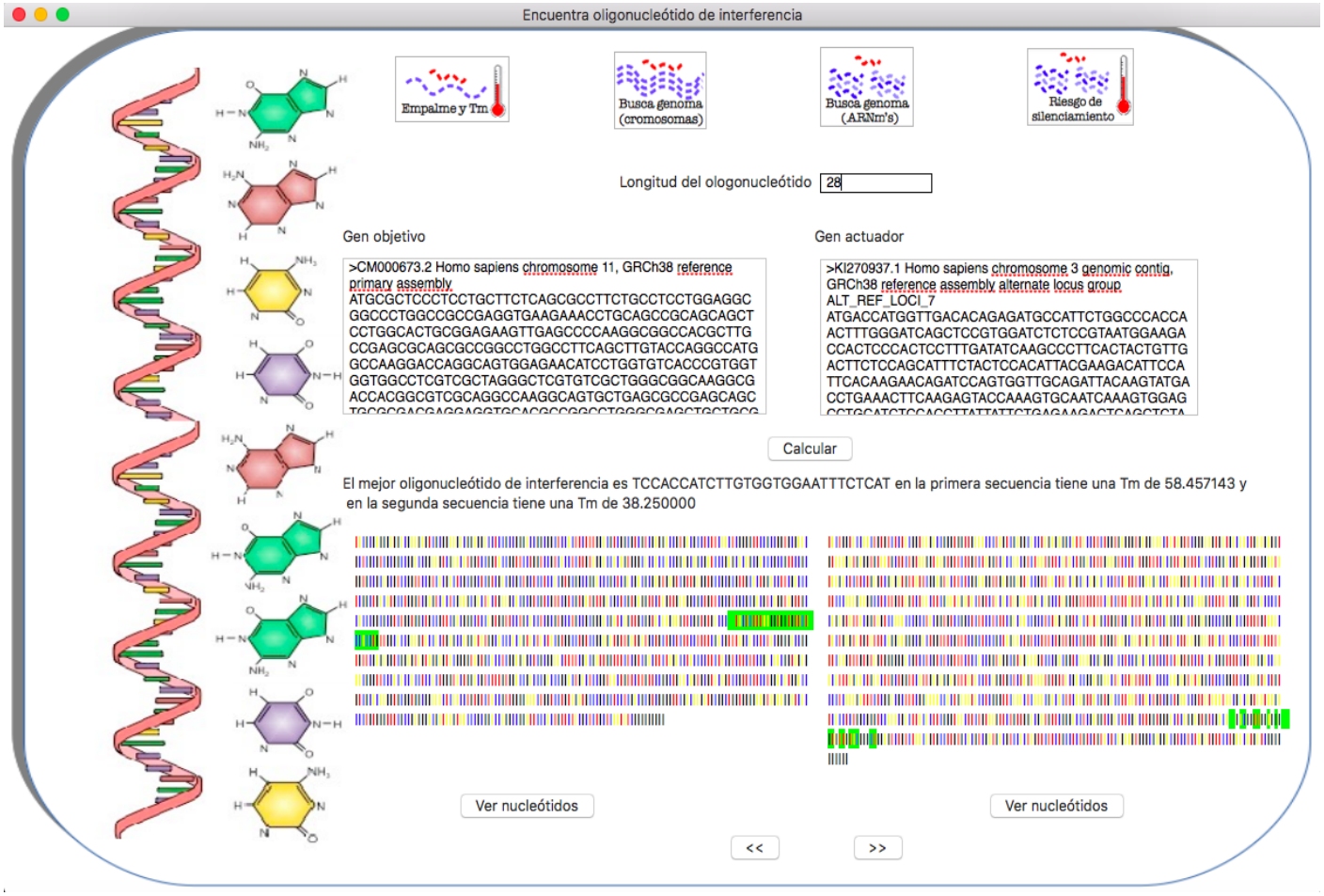

Figura 4.10: Ventana de la aplicación "Encuentra oligonucleótido de interferencia"

En la parte inferior de cada representación del gen tanto el que funge como objetivo, como el que funge como actuador, se encuentra el botón “ver nucleótidos”, el cual después de ser presionado muestra una nueva ventana que contiene los nucleótidos representados con letras ‘A’ para adenina (rojo), ‘C’ para citosina (negro), ‘G’ para guanina(azul) y ‘T’ para timina(amarillo), y la parte donde el oligonucleótido de interferencia es complementaria, la cual está subrayada con verde fosforescente, como se muestra en la figura 4.11.

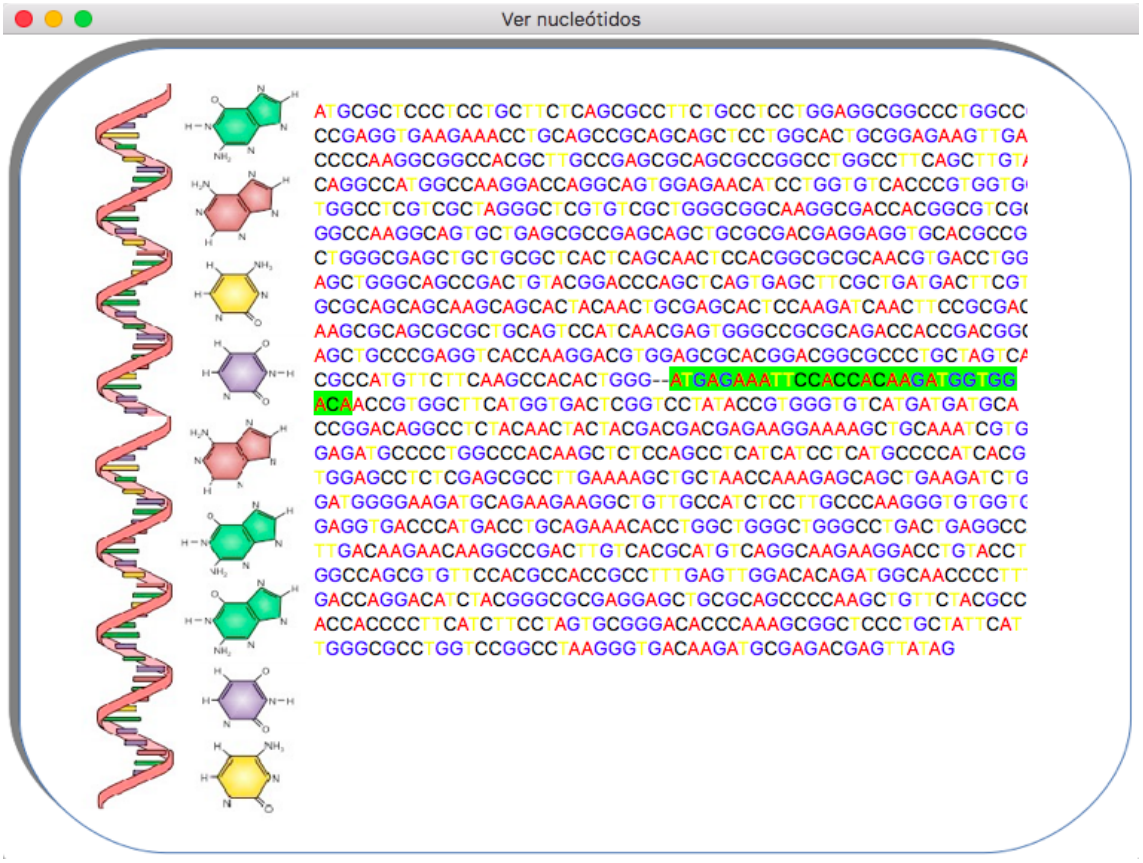

Figura 4.11: Ventana de la aplicación “Ver nucleótidos”

En la parte inferior de la pantalla “Encuentra oligonucleótido de interferencia” se encuentran dos botones que nos permiten conocer los otros candidatos a oligonucleótidos de interferencia ordenados por Tm, el botón >> permite ir hacia adelante en la lista de los oligonucleótidos, mientra que el botón << permite ir hacia atrás.

4.4.3.3. Encuentra empalme y Tm

Este módulo nos permite consultar en que parte de un gen un oligonucleótido se empalma y la Tm entre ambos. En la ventana de este módulo (véase figura 4.12) es necesario colocar el oligonucleótido y el gen representados como una cadena en el alfabeto Σ. Una vez que se presiona el botón calcular, en la parte inferior se muestra un texto que indica la Tm entre los dos oligonucleótidos y seguido se muestra el gen, cuyos nucleótidos están representados con el símbolo ‘|’ de distintos colores, donde el color rojo es el nucleótido adenina, el color negro es el nucleótido citocina, el color azul es el nucleótido guanina y el color amarillo es el nucleótido timina. Dentro de la representación del gen se encuentran uno o varios tramos que esta subrayados con verde fosforescente que representan la parte donde el oligonucleótido de interferencia es complementario con el gen.

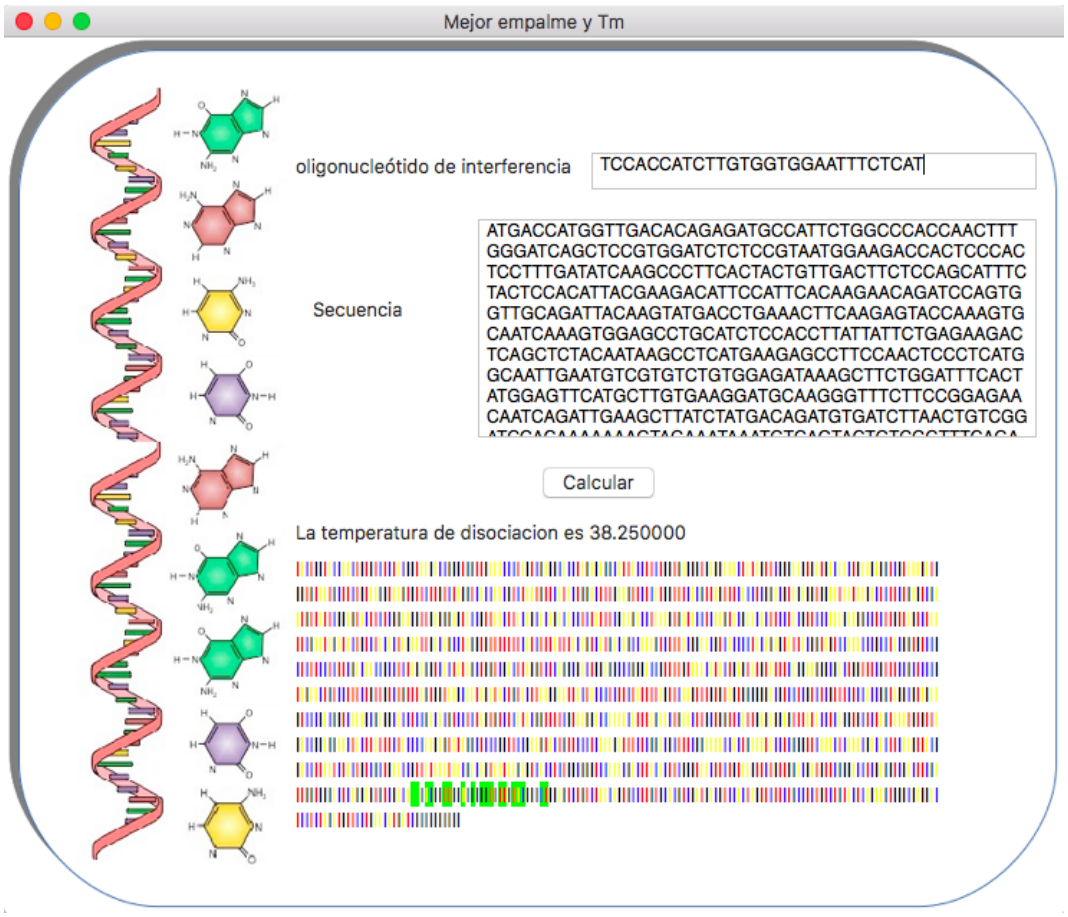

Figura 4.12: Ventana de la aplicación "Encuentra empalme y Tm"

4.4.3.4. Busca en el genoma (cromosomas)

Este módulo permite buscar si un oligonucleótido es totalmente complementario con algún segmento de un cromosoma del genoma humano, en caso de ser así, se proporciona en que posición y en que cromosoma el oligonucleótido es complementario.

Para este módulo, se tienen dos propuestas implementadas una donde se hace uso de programación concurrente, para reducir el tiempo de búsqueda de la solución, y otra donde la programación es secuencial. En este último caso se hace uso de un archivo con extensión .fna que contiene todos los cromosomas del genoma humano; fue descargado de [Ensemble, 2000]. Cada cromosoma del archivo tiene un segmento de descripción como el siguiente :>CM000663.2 Homo sapiens chromosome 1, GRCh38 reference primary assembly, donde:

- *CM000663.2*, (GenBank) es una base de datos de secuencias que es libre.
- *Homo sapiens*, se refiere a la especie; en este caso el ser humano.
- *chromosome*, indica el número de cromosoma de la secuencia.
- *GRCh38 reference primary assembly*, se refiere al consorcio del genoma humano construido número 38.

Optimización con programación concurrente

Analizando la estructura del archivo que contiene los ARNm maduros del genoma humano, se puede observar que se encuentra compuesta por varios segmentos: una línea descriptiva como la siguiente: '>CM000663.2 Homo sapiens chromosome 1, GRCh38 reference primary assembly', seguida de un segmento de letras que representan los nucleótidos (A, C, G, T), cada una de las líneas descriptivas se encuentran conformadas por el nombre del organismo en cuestión, el número de cromosoma y el banco de donde fue recaudada la información.

El algoritmo de búsqueda del oligonucleótido de interferencia en el genoma visto en el capítulo dos sólo hace uso de los segmentos que contienen las letras que hacen referencia a los nucleótidos, no toma en cuenta las líneas descriptivas, por lo que es posible fragmentar el archivo de los ARNm maduros del genoma humano en diferentes archivos a partir de sus líneas descriptivas, esto con el fin de ejecutar el algoritmo paralelamente para cada archivo, lo que reduce el coste en tiempo para hallar una solución.

Para poder llevar a cabo simultáneamente la ejecución del algoritmo para todos los archivos, se hizo uso de la programación concurrente. La programación concurrente permite ejecutar tareas simultáneamente, estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Los hilos de ejecución comparten los mismos recursos, tales como el espacio de memoria, los archivos abiertos, la situación de autenticación, etc.

Para la aplicación se hace uso de 24 archivos, 22 de los archivos corresponden a los oligonucleótidos que conforman los 22 pares cromosoma en el ser humano y los dos últimos archivos corresponden uno al cromosoma X y el otro al cromosoma Y. Éstos fueron descargados de [NCBI, 2017] y contienen una descripción de su respectivo cromosoma como la siguiente:

>gi|568815597|ref|NC_000001.11| Homo sapiens chromosome 1, GRCh38.p7 Primary Assembly, donde:

- *gi|568815597*, es el número GI (para GenInfo Identifier), una serie simple de dígitos que se asignan consecutivamente a cada registro de secuencia procesado por NCBI.
- *NC_000001.11*, (GenBank) es una base de datos de secuencias que es libre.
- *Homo sapiens* se refiere a la especie; en este caso el ser humano.
- *chromosome*, indica el número de cromosoma de la secuencia.
- *GRCh38 reference primary assembly*, se refiere al consorcio del genoma humano construido número 38.

Entre mayor sean los recursos de computo (memoria ram, núcleos de procesamiento, etc) con los que cuente una computadora, el número de hilos que se pueden ejecutar al mismo tiempo de manera eficiente será mayor y con esto el tiempo de ejecución de un programa diseñado para ejecutarse de manera concurrente disminuira. En caso de contar con recursos de computo limitados, habrá que hacer un análisis para averiguar la cantidad de hilos más adecuada para ejecutar dicho programa.

El tiempo de ejecución del programa esta dado por el tiempo de lectura de los datos más el tiempo de procesamiento de estos. Al contar con un solo disco duro, el tiempo de lectura de datos incrementa al intentar leerlos desde hilos diferentes, ya que la lectura se realiza serialmente, por lo que al tener más de un hilo estas tareas terminan por encolarse, volviendo este proceso más lento. Por el contrario el tiempo de procesamiento de los datos disminuye al distribuirse en diferentes hilos. Por lo que encontrar el tiempo de ejecución basado en el número de hilos

no tiene un comportamiento lineal. En este caso se hicieron pruebas con distintos números de hilos, hasta que la cantidad de minutos fuera la misma entre un número de hilos y otro, se obtuvieron los siguientes resultados: 5.9 minutos para 1 hilo, 3.31 minutos para 2 hilos, 3.09 minutos para 3 hilos, 3.0 minutos para 4 hilos, 2.80 minutos para 5 hilos, 2.62 minutos para 6 hilos y 2.62 para 7 hilos. Por lo tanto se concluye que se requieren aproximadamente máximo 6 hilos, para que la aplicación reduzca el tiempo de ejecución a 2.62 minutos , mientras que cuando no se usaron hilos el tiempo en el que la aplicación arrojó el resultado fue de 5.998427 minutos. Por lo que con programación concurrente el tiempo en el que la aplicación arroja un resultado se reduce en más de la mitad del tiempo que sin usar esta.

Sin embargo la aplicación cuenta con las dos opciones de implementación, ya que para el caso donde no se hace uso programación secuencial solo es necesario un archivo que contenga todos los cromosomas, mientras que para el caso donde se hace uso de programación concurrente se necesita un archivo por cada cromosoma.

En la ventana de este módulo se requiere escribir el oligonucleótido que se desea buscar en el genoma humano (véase la figura 4.13). Después de oprimir el botón de calcular, se realiza la búsqueda y en caso de que no se encuentre una coincidencia en el genoma humano se mostrara el siguiente texto “El oligonucleótido no se encuentra en el genoma”, en caso contrario se mostrara la descripción del o de los cromosomas donde el oligonucleótido es complementario totalmente. Pero se espera que como mínimo aparezca el gen objetivo

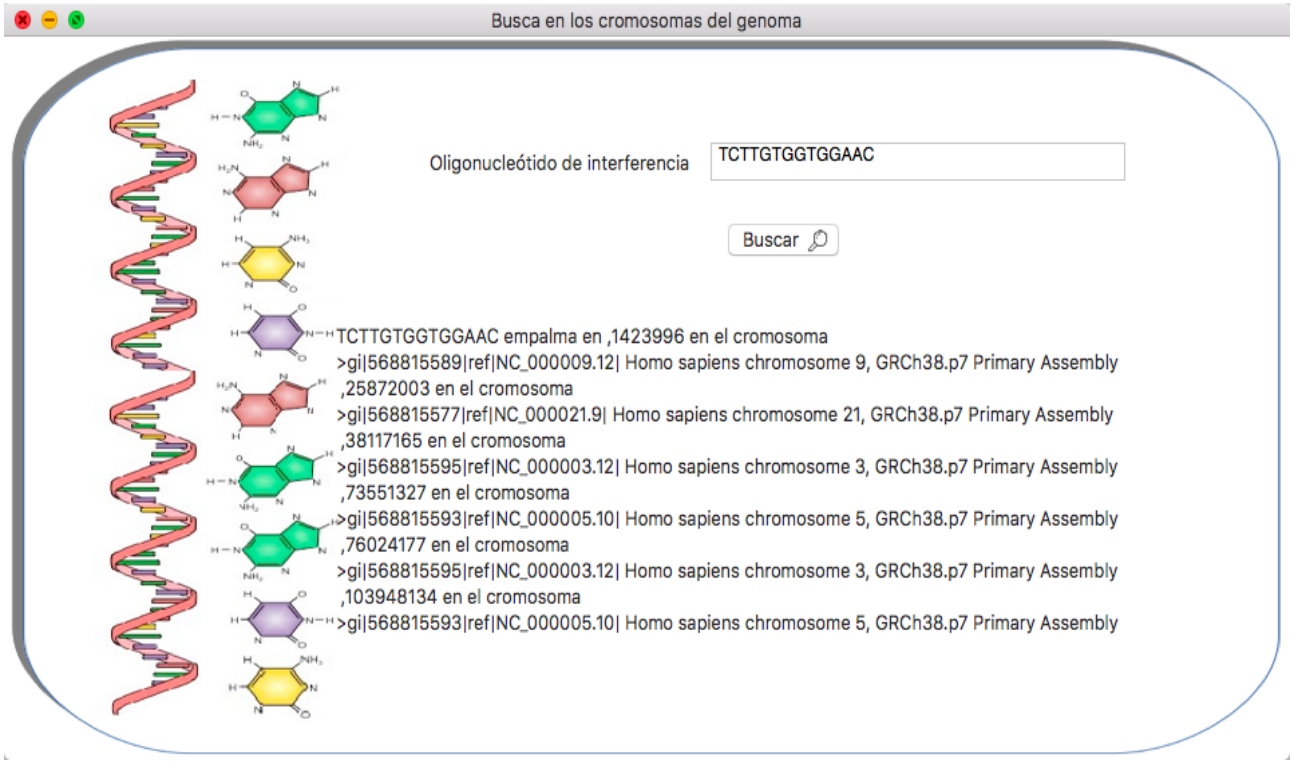

Figura 4.13: Ventana de la aplicación “Búsqueda del genoma(cromosomas)”

Este módulo al igual que BLAST puede buscar si un oligonucleótido es totalmente complementario en el genoma humano, pero es importante mencionar que BLAST usa un algoritmo heurístico por lo que no nos puede garantizar que se encuentren todas las soluciones.

4.4.3.5. Busca en el genoma (ARNm)

Este módulo permite buscar si un oligonucleótido es totalmente complementario con algún ARNm maduro del genoma humano, es decir, busca si el oligonucleótido propuesto es capaz de silenciar otro gen que no sea el objetivo, en caso de ser así, se proporciona en que posición y en que ARNm el oligonucleótido es complementario.

Para este módulo se hace uso de un sólo archivo con extensión .fasta que contiene todos los ARNm maduros del genoma humano, el cual fue descargado de [NCBI-AceView, 2012]. Este archivo contiene diferentes variantes de las ARNm que son distinguidas por las letras a, b, c, d, e, e-unspliced, f. Por ejemplo la variante 'a' se reconstruye a partir de 8 clones de cDNA.

En el caso de la aplicación sólo se hará uso de la variante 'a', para lo cual se realizó un programa, que crearía un nuevo archivo con los ARNm de la variante 'a' y eliminaría todos los ARNm con diferentes variantes.

En el archivo los ARNm contienen un segmento de descripción como el siguiente :

>ARNm:A1CF.aAug10|Gene|A1CF|GeneId29974. donde:

- *A1CF*, representa el gen
- *aAug10*, representa la variante junto con la fecha
- *GeneId29974*, representa el id del gen.

En la ventana de este módulo se requiere escribir el oligonucleótido que se desea buscar en los ARNm maduro del genoma humano (véase la figura 4.14). Cuando se oprime el botón de calcular, se realiza la búsqueda en los ARNm maduros del genoma humano, en caso de que no se encuentre una coincidencia en el genoma humano se mostrara el siguiente texto "El oligonucleótido no se encuentra en el genoma", en caso contrario se mostrara la descripción del o de los ARNm donde el oligonucleótido es complementario totalmente. Pero se espera que como mínimo aparezca el gen objetivo.

4.4.3.6. Riesgo de silenciamiento fuera del objetivo

Este módulo permite buscar si un oligonucleótido cumple con la tolerancia de incompatibilidad dentro de los ARNm del genoma humano. Si cumple con esta tolerancia es posible que el oligonucleótido silencie algún gen que no sea el gen objetivo, en tal caso, se proporciona la descripción del ARNm y en que posición del ARNm se encuentra.

Para este módulo se hace uso de un sólo archivo con extensión .fasta, el mismo archivo que se describió en la sección anterios que contiene los ARNm del genoma humano.

En la ventana de este módulo se requiere escribir el oligonucleótido que se desea buscar en los ARNm del genoma humano (véase la figura 4.15). Después de oprimir el botón de "calcular", se realiza la búsqueda en los ARNm del genoma, en caso de que no se encuentre una coincidencia se mostrara el siguiente texto "El oligonucleótido no se encuentra en el genoma", en caso contrario se mostrara la descripción del o de los ARNm donde el oligonucleótido cumple con la tolerancia de incompatibilidad y su Tm. Pero se espera que como mínimo aparezca el gen objetivo.

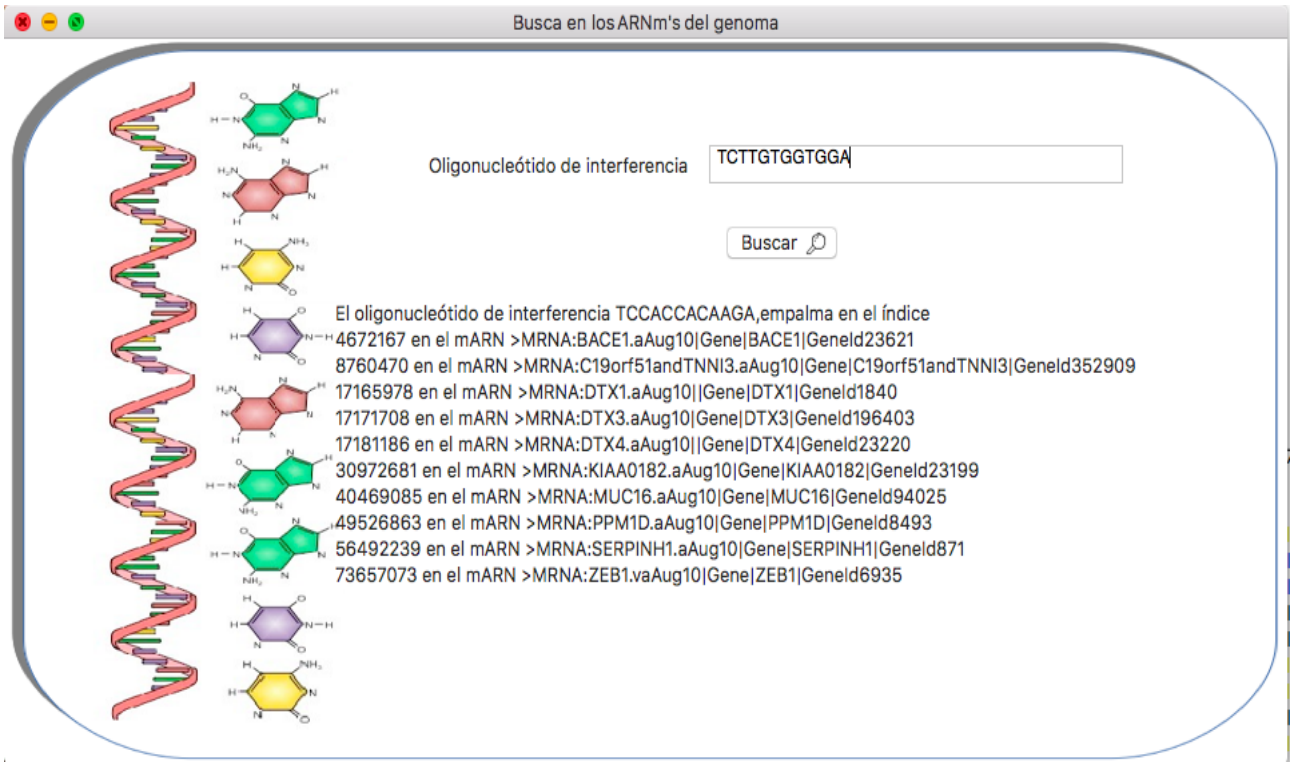

Figura 4.14: Ventana de la aplicación "Búsqueda del genoma(ARNm)"

4.4.3.7. Método Tm

Como se había comentado anteriormente existen distintas fórmulas para obtener la Tm entre dos oligonucleótidos. Está es indispensable para los algoritmos que se desarrollaron y es usada en todos los módulos de la aplicación para obtener resultados. Debido a que un usuario puede preferir el uso de una u otra fórmula de Tm se implemento en el menú una opción para que el usuario pueda elegir el método de su preferencia. El objetivo de que está opción se colocara en el menú, es que el usuario puede cambiar el método de Tm en el momento en que desee no importando en que módulo se encuentre, por lo que se tiene un marcador que indica cual de los dos métodos es el que se encuentra activado (véase figura 4.16).

En la aplicación se contemplan dos métodos para obtener la Tm:

El método de Wallace: se suele usar para secuencias de menos de 14 nucleótidos y se define de la siguiente manera:

$$Tm = (wA + xT) \cdot 2 + (yG + zC) \cdot 4$$

donde w, x, y, z es el número de las bases A, T, G, C en la secuencia, respectivamente.

El método de Marmur preferentemente se usa para secuencias de más de 14 nucleótidos y se define de la siguiente manera:

$$Tm = 64{,}9 + 41 \cdot (yG + zC - 16{,}4)/(wA + xT + yG + zC)$$

donde w, x, y, z es el número de las bases A, T, G, C en la secuencia, respectivamente. La aplicación tiene este último método habilitado por omisión.

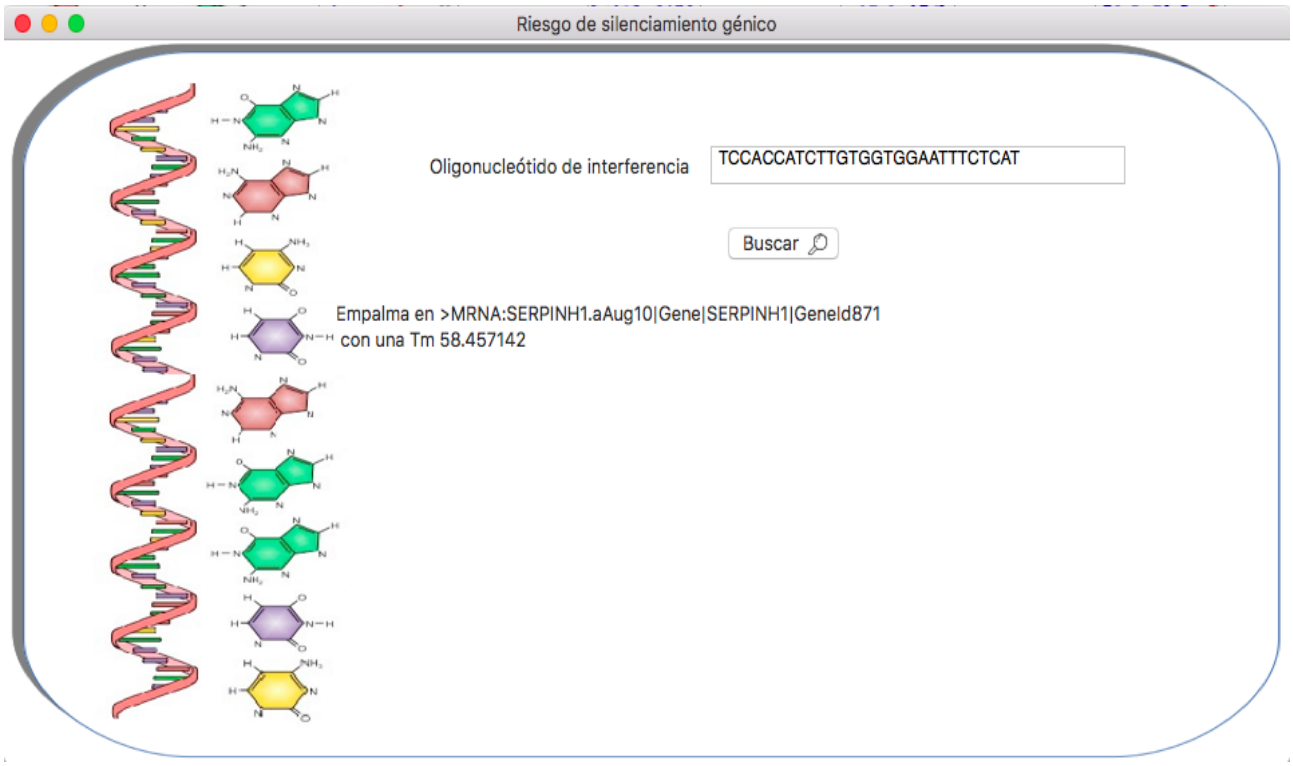

Figura 4.15: Ventana de la aplicación "Riesgo de silenciamiento fuera del objetivo"

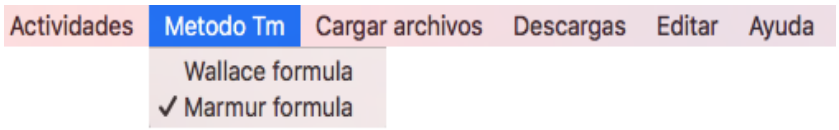

Figura 4.16: menú de la aplicación "Método Tm"

4.4.3.8. Cargar archivos

Debido que las bases de datos de los genomas se actualizan y son distribuidas por diferentes centros de información, en el menú se agregó una opción que permite al usuario cargar la base de datos de su preferencia, para hacer el análisis y obtener el mejor oligonucleótido de interferencia.

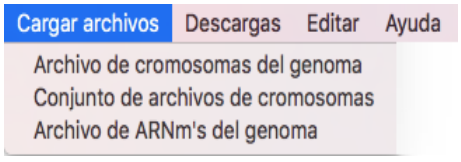

Figura 4.17: Menú de la aplicación "Cargar archivos"

Este menú tiene tres opciones diferentes como se muestra en la figura 4.17, al elegir una de esas opciones se abrirá una ventana como la que se muestra en la figura 4.18, la cual le permite al usuario elegir un archivo que se encuentre en su computadora y que contenga una base de datos del genoma, en este caso en la ventana solo se habilitan archivos con el siguiente tipo de extensiones .fasta, .fa .fna, que son las principales extensiones de las bases de datos biológicas.

La primera opción del menú "Archivo de cromosomas del genoma", permite al usuario cargar

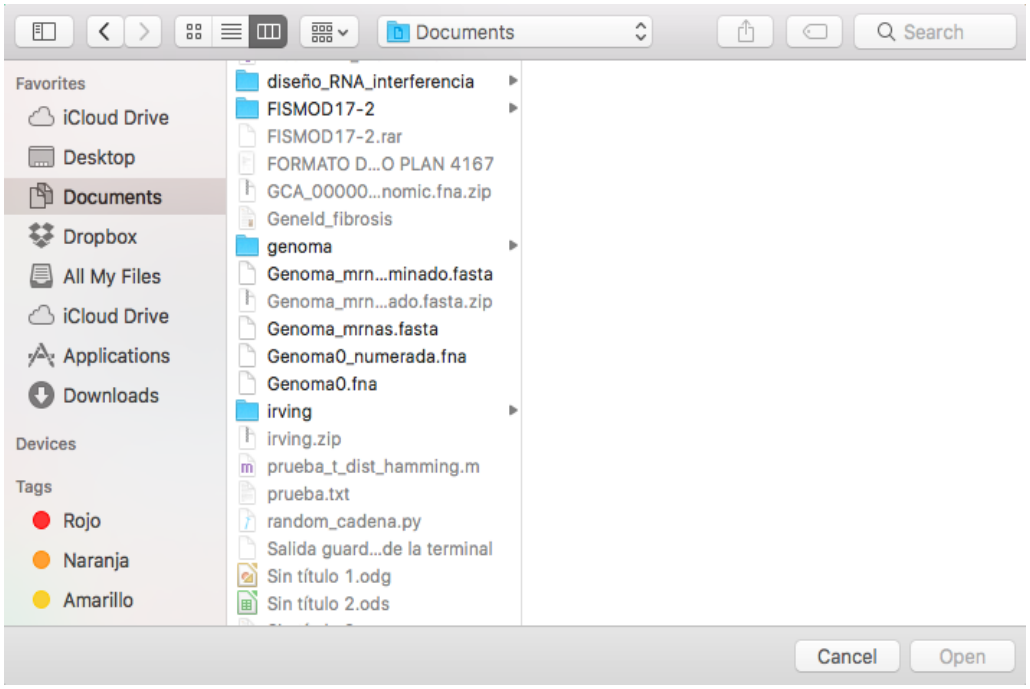

Figura 4.18: Ventana para cargar un archivo de base de datos

un archivo que contenga todos los cromosomas del genoma, al elegir está opción, se tiene un marcador el cual indica que todos los cromosomas se encuentran en un solo archivo y por lo tanto se ejecuta el algoritmo secuencial; por el contrario si el usuario elige la opción "Conjunto de archivos de cromosomas" el algoritmo que se ejecuta es el que se encuentra implementado con programación concurrente. En este caso es necesario que el usuario tenga todos los archivos de los cromosomas en la misma carpeta y con que solo elija el primer archivo con el primer cromosoma del conjunto de archivos de cromosomas del genoma, la aplicación podrá leer los demás. La última opción de este menú es "Archivo de ARNm del genoma", esta permite al usuario cargar un archivo que contenga los ARNm del genoma humano.

4.4.3.9. Descargas

La aplicación cuenta con un menú que permite al usuario descargar las bases de datos de los cromosomas del genoma o de los ARNm. El menú cuenta con tres opciones: "Archivo de cromosomas del genoma humano", "Conjunto de archivos de cromosomas del genoma humano", "Archivo de ARNm del genoma humano"(véase figura 4.19)

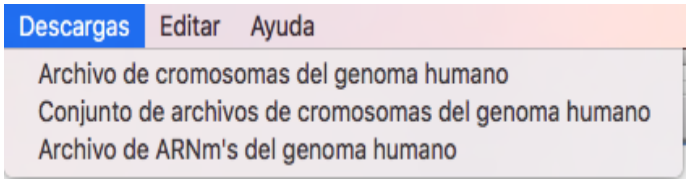

Figura 4.19: Menú de la aplicación "Descargas"

Cada una de las opciones abre una nueva ventana la cual le permite al usuario escribir o buscar la dirección en su computadora donde desea que se guarde la base de datos, para esta última opción se encuentra un botón del lado izquierdo llamado "Guardar en ". En la parte inferior del botón el usuario tiene la opción de ponerle el nombre que desee al archivo que va a descargar,

seguido de esto se encuentra una URL, que es la página ftp que se recomienda para descargar el archivo y la cual fue obtenida de la página HTML que se encuentra por debajo de la ventana '(como se muestra en la figura 4.20), la cual fue cargada en la aplicación en los casos de las ventanas de "Conjunto de archivos de cromosomas del genoma humanoz "Archivo de ARNm del genoma humano".

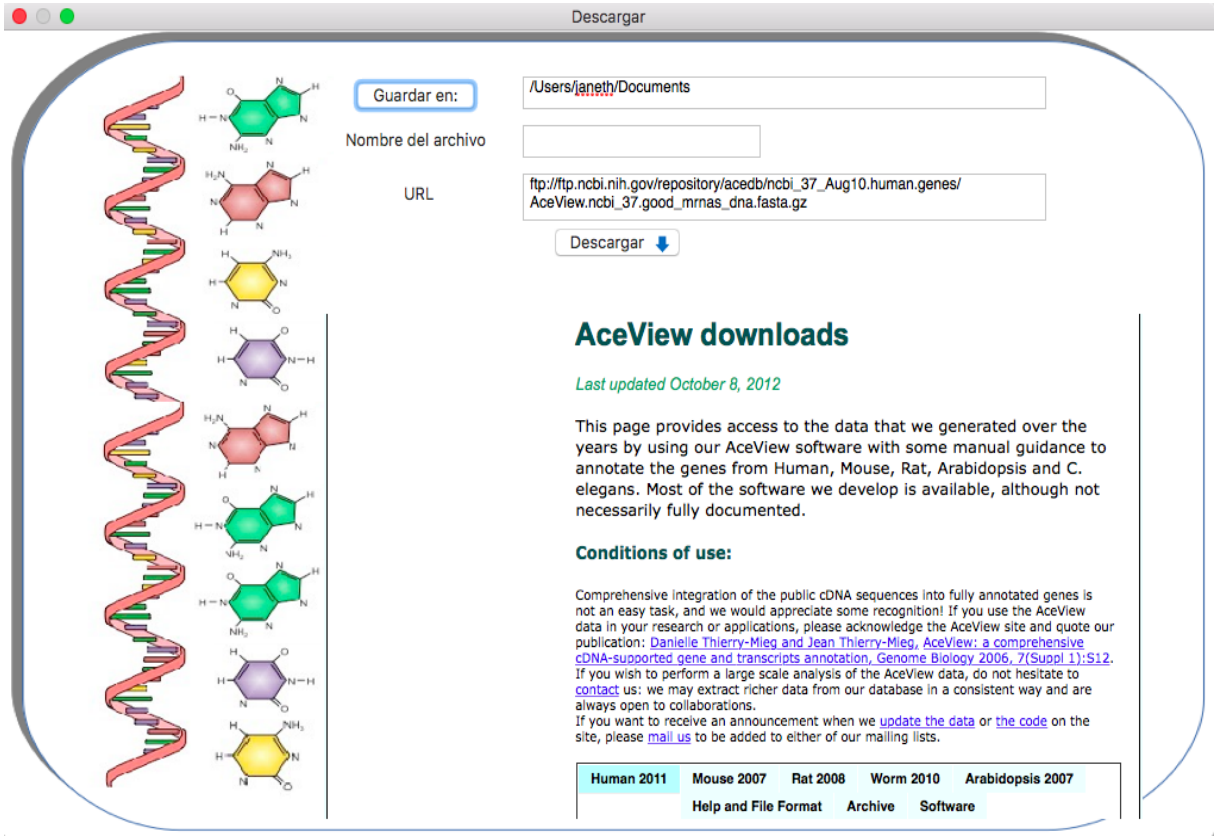

Figura 4.20: Pantalla de descargas del Archivo de los ARNm del genoma humano

4.4.4. Pruebas y resultados

El objetivo de la aplicación es encontrar un oligonucleótido de interferencia que sea capaz de inhibir la fibrosis hepática. Para comenzar se debe especificar la longitud del oligonucleótido.

Por la definición del problema, la Tm máxima entre el oligonucleótido y el gen de $PPAR\gamma$ debe ser menor o igual a 37°C y el oligonucleótido propuesto no debe ser subcadena de los ARNm del genoma humano. Se sabe que entre más pequeña sea una cadena de letras, es más probable que se encuentre como subcadena de otra cadena, por ejemplo, es más probable que en una cadena de tamaño 100, aparezca más veces una cadena de tamaño 3 que una de tamaño 50.

Si los ARNm del genoma humano tiene como alfabeto $\Sigma = `A', `C', `G', `T'$ y el número de letras de los ARNm del genoma humano es 60,324,144. Suponiendo que los ARNm del genoma humano tienen una distribución uniforme de letras, entonces se tienen 4^n posibilidades de palabras de tamaño n y la probabilidad de que estás aparezcan como subcadena de los ARNm del genoma humano es de $P(n) = (60,324,144 - n)/4^n$, entonces se tienen las siguientes probabilidades

$$P(21) = 0{,}0000137161$$

$$P(23) = 8{,}57210^{-7}$$

$$P(25) = 5{,}35710^{-8}$$

$$P(26) = 1{,}33910^{-8}$$

$$P(27) = 3{,}34810^{-9}$$

$$P(28) = 8{,}3710^{-10}$$

Observando las probabilidades, entre mayor sea el valor de n, menor es la probabilidad de que la cadena de tamaño n aparezca como subcadena en el genoma y sabiendo que el empalme entre el oligonucleótido propuesto y el gen $PPAR\gamma$ debe de cumplir con la Tm máxima menor o igual a 37°C, entonces el tamaño de la cadena no debe de ser tan grande, por lo que se realizaron pruebas con cadenas de longitudes 26, 27, 28.

Si se requieren oligonucleótidos de tamaño 28 la aplicación arroja que el único oligonucleótido posible tiene una Tm mínima de 38.25°C, la cual sobrepasa la Tm establecida en la definición del problema, por lo tanto, no tiene caso seguir verificando la Tm de las cadenas de mayor longitud, ya que cadenas de mayor longitud tendrán una mayor Tm. Si se requieren oligonucleótidos de longitud 27, la aplicación arroja como resultado el oligonucleótido 3'-CCACCATCTTGTGGTGGAATTTCTCAT-5' cuya Tm es 36.4733°C (véase figura 4.21).

Este resultado no sobrepasa los 37°C establecidos en la definición del problema y sólo empalma en una posición con el gen PPARγ, así que el siguiente paso es verificar si es capaz de silenciar otro gen, primero se verifica si esta cadena es totalmente complementaria con algún ARNm del genoma humano, para esto se puede hacer uso del módulo "Búsqueda en el genoma(ARNm)", el resultado que se obtiene es que este oligonucleótido se encuentra en el ARNm: "MRNA:SERPINH1.aAug10|Gene|SERPINH1|GeneId871", este GeneId en los bancos de datos pertenece a Hsp47, que es el gen objetivo.

El siguiente paso es averiguar si existe riesgo de silenciamiento fuera del objetivo tomando la tolerancia de incompatibilidad, para esto se hace uso del módulo "Riesgo de silenciamiento fuera del objetivo", en este caso el resultado que obtenemos es el mismo que el anterior, el único riesgo de silenciamiento es con "MRNA:SERPINH1.aAug10|Gene|SERPINH1|GeneId871" que es el gen que representa a Hsp47.

Verificamos si existen otros posibles oligonucleótidos que cumplan con la condición de Tm que no sobrepase los 37°C, para esto en la pantalla principal como ya se había mencionado anteriormente se encuentran dos botones en la parte inferior, que nos permiten ver los demás posibles oligonucleótidos de interferencia, sin embargo para el caso de oligonucleótidos de 27 letras no existen más candidatos.

Si se requieren oligonucleótidos de tamaño 26, la aplicación arroja como resultado principal la cadena 3'- CACCATCTTGTGGTGGAATTTCTCAT -5' que tiene una Tm de 35.253846°C y sólo empalma en una posición con el gen PPARγ (véase figura 4.22).

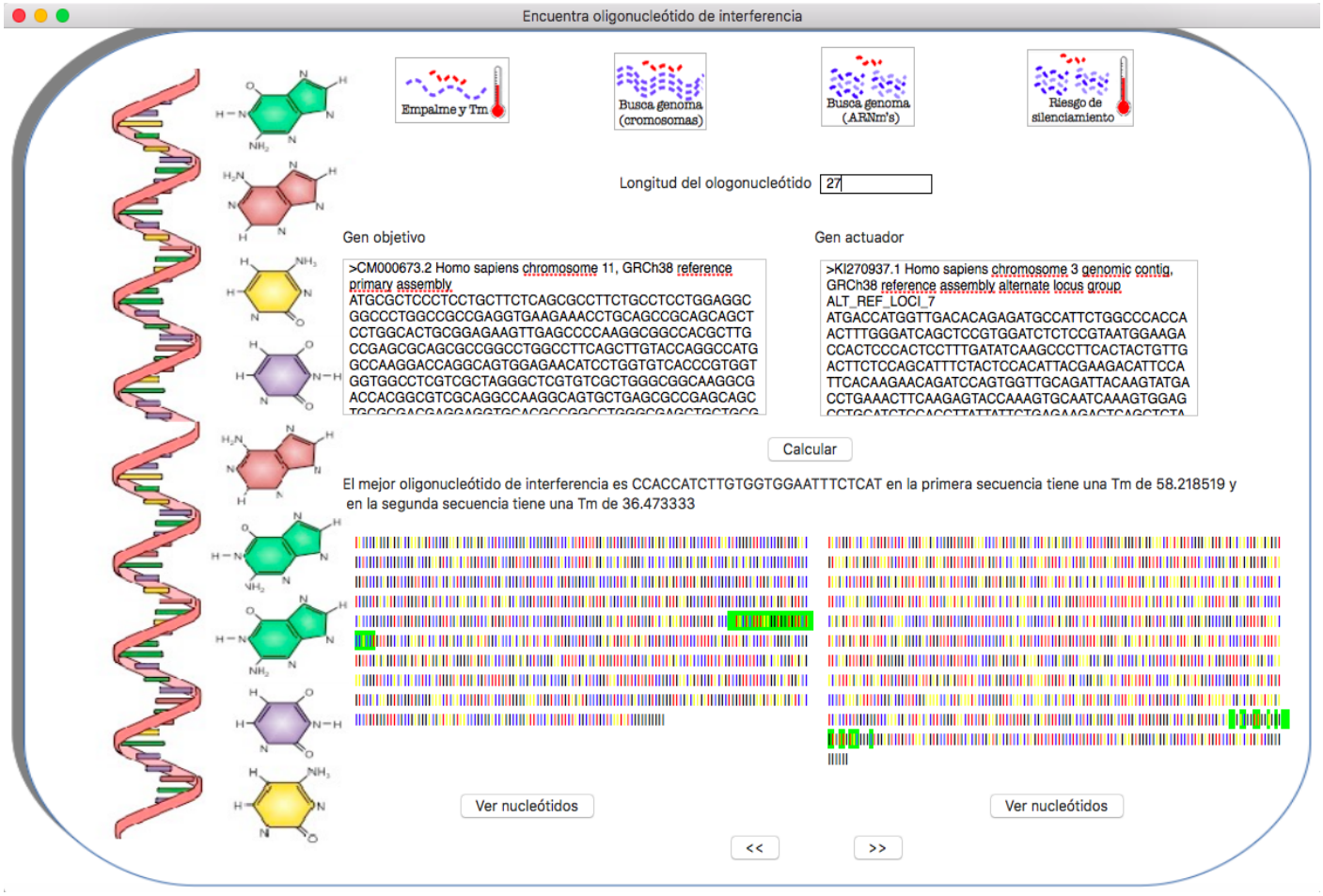

Figura 4.21: Resultado para cadena de longitud 27

Se realizan los mismos pasos que con el oligonucleótido obtenido de 27 letras, se verifica si es capaz de silenciar otro gen con el módulo 'Búsqueda en el genoma(ARNm)" y como el resultado se obtiene que se encuentra en el ARNm: "MRNA:SERPINH1.aAug10|Gene|SERPINH1|GeneId871", que pertenece a Hsp47, por lo que se busca si existe otro riesgo de silenciamiento génico y la respuesta es la misma que la anterior, no hay riesgo de silenciamiento fuera del gen objetivo.

Se verifica en la aplicación si existen otros posibles oligonucleótidos de interferencia, en este caso aparecen otras opciones entre las cuales están :

1. TTGATCTTGGAGTGCTCGCAGTTGTA, tiene la misma Tm 35.253846°C pero puede empalmar en 3 posiciones del gen PPARγ.
2. CTTGTCGCGGAAGTTGATCTTGGAGT tiene una Tm de 36.2°C, y empalma en una posición más cercana al inicio del gen PPARγ que el oligonucleótido del resultado principal (véase figura 4.23). Dadas estas características este puede ser un buen candidato, así que se hacen las pruebas para ver si hay riesgo de silenciamiento génico en algún lugar del genoma humano y el resultado es que sólo hay silenciamiento en Hsp47.
3. ACGGTATAGGACCGAGTCACCATGAA tiene una Tm de 36.2°C pero empalma en dos posiciones de PPARγ.
4. AATTTCTCATCCCAGTGTGGCTTGAA, CCACCATCTTGTGGTGGAATTTCTCA, TCCACCATCTTGTGGTGGAATTTCTC y TTGTCCACCATCTTGTGGTGGAATTT tienen una Tm de 36.47333°C , empalman en una sola posición de PPARγ pero sus po-

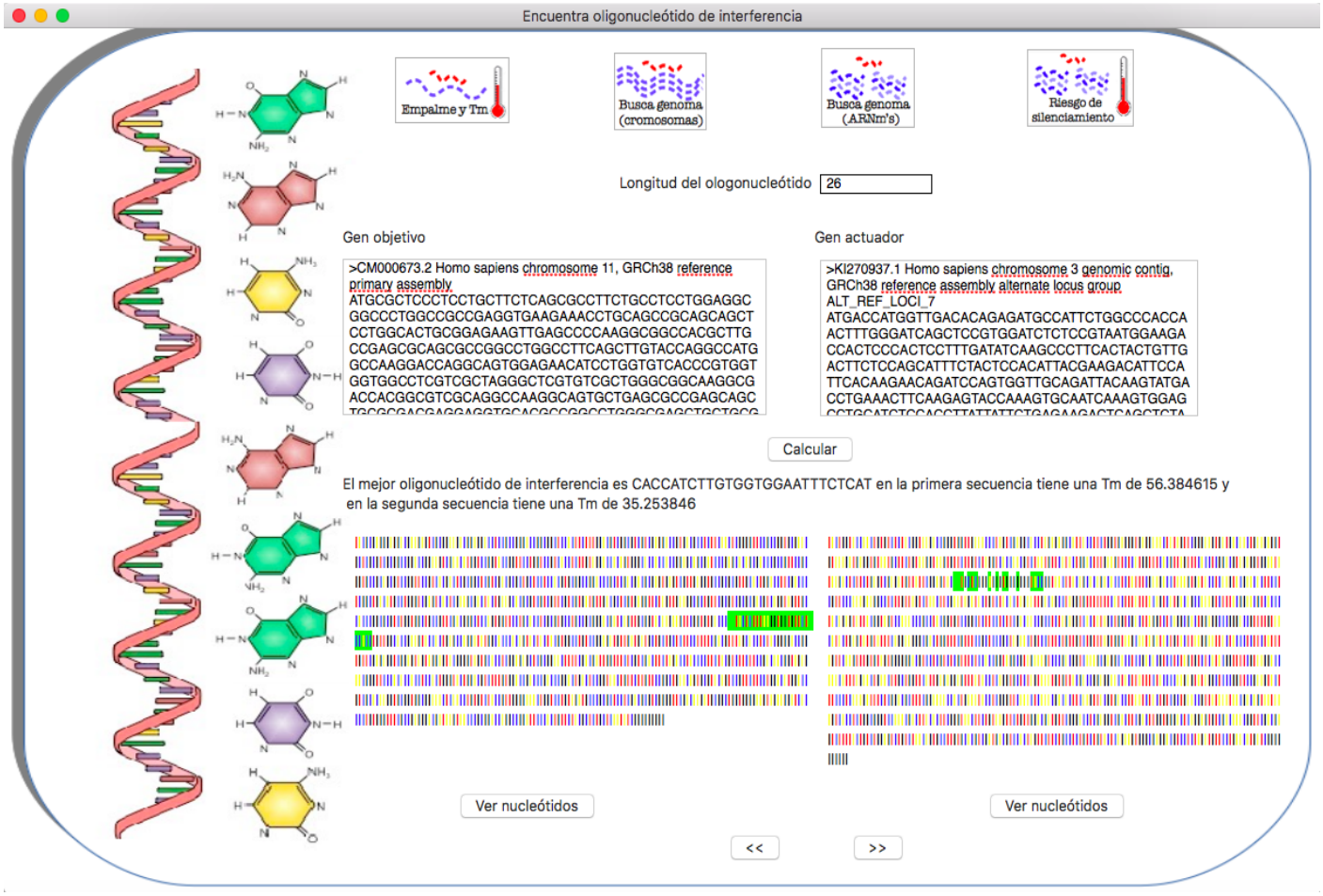

Figura 4.22: Resultado para cadena de longitud 26

siciones son más lejana que el oligonucleótido del número 2, CTTGTCGCGGAAGTTG-ATCTTGGAGT, por lo tanto este último es una mejor opción.

5. TCTTTGGTTAGCAGCTTTTCAAGGCG tiene una Tm de 36.47333 ° C y empalma en dos posiciones de PPARγ.

Dados estos resultados, tres de los mejores candidatos son los siguientes CCACCATCTTGT-GGTGGAATTTCTCAT, CACCATCTTGTGGTGGAATTTCTCAT y CTTGTCGCGGAA-GTTGATCTTGGAGT. Entre los dos últimos, CTTGTCGCGGAAGTTGATCTTGGAGT a pesar de que tiene su Tm más alta, la posición en la que empalma con PPARγ es más cercana al inicio. Por lo que se propone hacer pruebas de laboratorio con los oligonucleótidos CCACCATCTTGTGGTGGAATTTCTCAT y CTTGTCGCGGAAGTTGATCTTGGAGT.

En el capítulo 3 se menciono que los motifs 5'-GUGU-3' podían activar la respuesta inmune innata por la presencia de un iARN y eliminarlo, por lo que haciendo una analogía con los oligonucleótidos propuestos, se puede pensar que quizá el mismo motif 5'-GUGU-3', pero intercambiando uracilo (U) por timina (T) ya que no es una cadena de ARN, puede activar la respuesta inmune innata. Analizando si el motif 5'-GTGT-3', o bien 3'-TGTG-5' se encontraba en los oligonucleótidos de interferencia propuestos: en el primer caso el oligonucleótido 3'- CCACCATCT**TGTG**GTGGAATTTCTCAT -5', tiene este motif, mientras que 3'- CT-TGTCGCGGAAGTTGATCTTGGAGT -5' no cuenta con este motif, así que si 3'- CCAC-CATCTTGTGGTGGAATTTCTCAT -5' en la experimentación no llega a silenciar Hsp47 y 3'- CTTGTCGCGGAAGTTGATCTTGGAGT -5' sí, es muy probable que el motif 5'-GTGT-

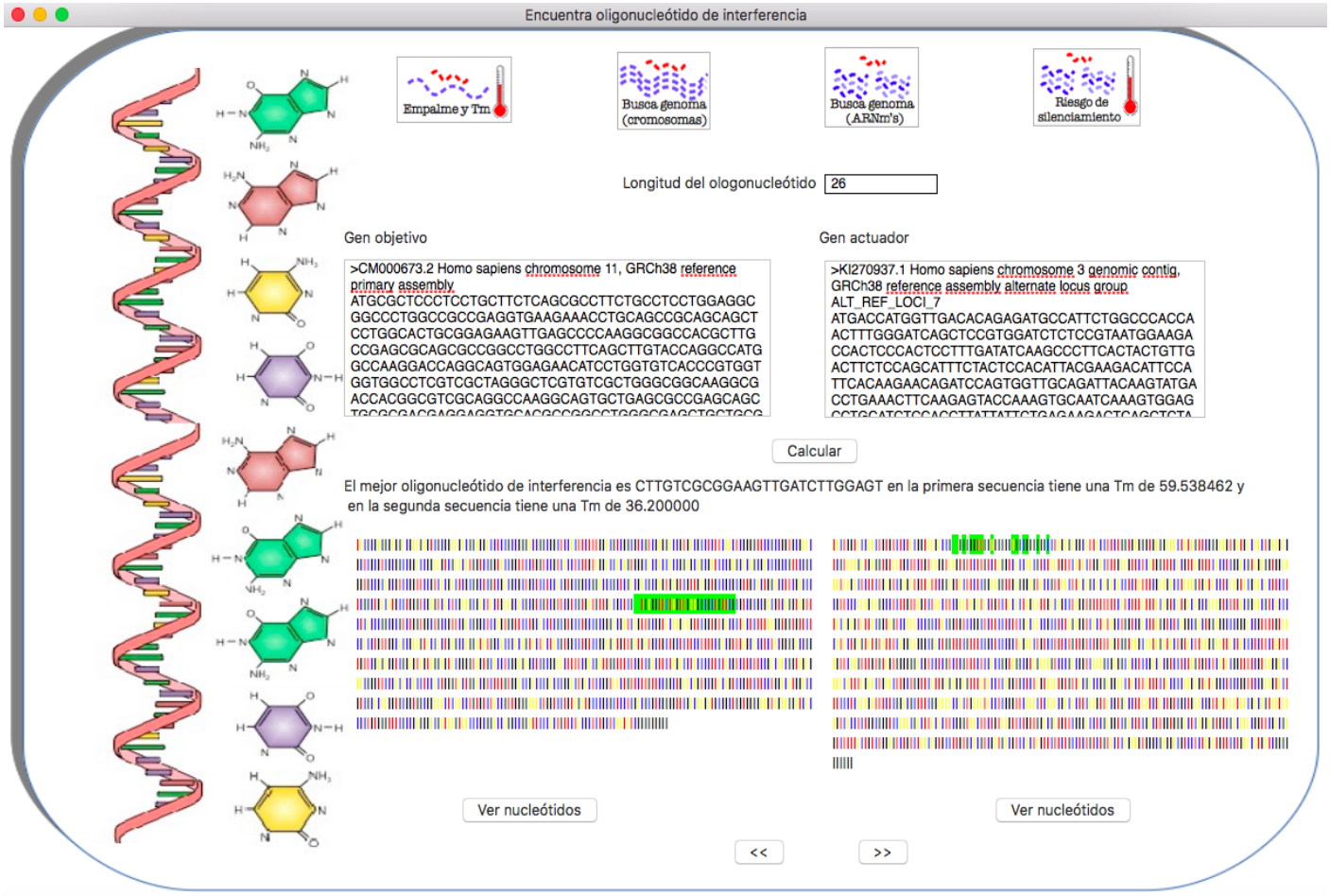

Figura 4.23: Mejor candidato como oligonucleótido de interferencia

3' quizá sea capaz de activar la respuesta inmune innata del cuerpo humano eliminando los oligonucleótidos que cuentan con éste.

4.5. Enfoque geométrico

Al intentar resolver el problema, surgió una idea que podría ayudar a reducir el tiempo de procesamiento para hallar la solución. En esta sección se expone y describen los problemas de aplicar esta idea y los resultados obtenidos.

4.5.1. Descripción

La idea es colocar en un espacio geométrico las subcadenas de las bases nitrogenadas A, C, G, T de la cadena O y las subcadenas de la cadena A (figura 4.24)

El objetivo es saber cuales de las subcadenas de A y O son más "parecidas", cuando se habla de que tan "parecidas" son las subcadenas, en realidad se desea saber entre que subcadenas existe la Tm mayor. Para obtener la Tm de la manera más sencilla, se hace uso de ciertas coordenadas cuyos valores fueron escogidos convenientemente para que al aplicar el producto punto entre dos puntos (que representan subcadenas) en el espacio geométrico el resultado sea

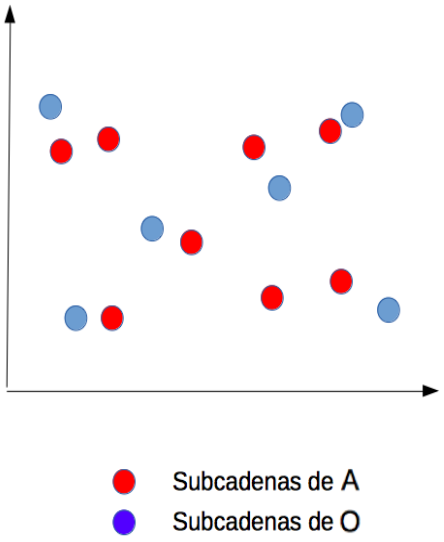

Figura 4.24: Ubicación de subcadenas

un valor equivalente al resultado de aplicar la función Tm entre dos subcadenas.

Si la función Tm está dada de la siguiente manera:

$$Tm = (wA + xT) \cdot 2 + (yG + zC) \cdot 4$$

Se tiene que por cada letra A ó T se agrega el valor de 2 al resultado de la Tm, mientras que por cada letra G ó C se agrega el valor de 4. Por lo tanto para que el producto punto de dos coordenadas que representan la letra A den como resultado el valor de 2, cada letra A debe de tener el valor de $\sqrt{2}$. Pero existe un problema por que para el producto punto entre dos coordenadas que representan la letra T también el resultado debe de ser el valor de 2, entonces si se sugiere el valor de $\sqrt{2}$ para la coordenada que representa la T, el producto punto entre las coordenadas de A y T será 2. Para solucionar el problema se agrega una dimensión a las coordenadas, es decir $A = (\sqrt{2}, 0)$ y $T = (0, \sqrt{2})$, asegurándonos que el producto punto entre las coordenadas A y T sea 0, esto significa que no hay Tm entre esas dos letras. Para dos letras que se comparen entre si y no sean las mismas, se debe de tener una Tm igual a cero, y debido a que el alfabeto en el que se esta trabajando es de 4 letras, el espacio geométrico que represente cada letra estará en $\mathbb{R}^4$.

Así las coordenadas que representan cada letra se encuentran en $\mathbb{R}^4$ y son las siguientes:

$A = (\sqrt{2}, 0, 0, 0)$

$T = (0, \sqrt{2}, 0, 0)$

$C = (0, 0, 2, 0)$

$G = (0, 0, 0, 2)$

Si se desea formar una cadena de letras entonces los puntos se encontraran en un espacio $\mathbb{R}^{4 \times \#letras}$. Las coordenadas de cada punto, el cual representa una cadena, están compuestas por la concatenación de las coordenadas de cada letra que conforma la cadena. Por ejemplo

para representar la cadena AGTT en el espacio geométrico, se concatenan las coordenadas de las letras A, G, T, T, como se muestra a continuación:

$$AGTT = (\sqrt{2}, 0, 0, 0, 0, 0, 0, 2, 0, \sqrt{2}, 0, 0, 0, \sqrt{2}, 0, 0)$$

La cadena GGTT se representa como el siguiente punto en el espacio geométrico :

$$GGTT = (0, 0, 0, 2, 0, 0, 0, 2, 0, \sqrt{2}, 0, 0, 0, \sqrt{2}, 0, 0)$$

Dadas las características mencionadas anteriormente se puede obtener el valor de la función de Tm para dos cadenas cualesquiera de misma longitud, haciendo uso del producto punto. Por ejemplo, tomando las coordenadas descritas anteriormente que representan las cadenas AGTT y GGTT, se tiene, que su producto punto es:

$AGTT \cdot GGTT = 0+0+0+0+0+0+0+4+0+2+0+0+0+2+0+0 =$**8**

Que es el mismo resultado de evaluar la función Tm con estas dos cadenas:

$Tm(AGTT, GGTT) = 0+4+2+2 =$**8**

A continuación se muestran otros dos ejemplos:

$AGTT \cdot AGTT =$**10** (Producto punto de las coordenadas que representan $AGTT$ y $AGTT$)

$Tm(AGTT, AGTT) =$**10** (Evaluación de la función Tm de $AGTT$ y $AGTT$)

$AGTT \cdot TACC =$**0** (Producto punto de las coordenadas que representan $AGTT$ y $TACC$)

$Tm(AGTT, TACC) =$**0** (Evaluación de la función Tm de $AGTT$ y $TACC$)

Siguiendo la definición del problema, la idea principal es encontrar la Tm_{max}. Se obtiene el producto punto de cada punto que represente cada subcadena de O con todos los puntos que representan las subcadenas de A (figura 4.25).

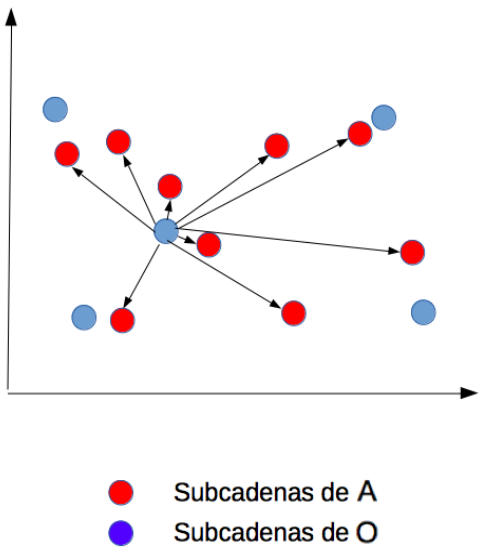

Figura 4.25: Distancias de una subcadena O a cada subcadena de A

Se toman los pares de puntos(subcadena O, subcadena A) más cercanos, es decir, el valor más alto al evaluar el producto punto entre dos puntos. Dadas las características del problema se sabe que el valor máximo de Tm tiene un límite, a partir del cual la subcadena de O ya no es una cadena que pueda ser viable para funcionar como oligonucleótido de interferencia, por lo que cuando el producto punto entre estos dos puntos sobrepase el límite ya no es necesario seguir realizando el producto punto con los puntos restantes que representan las subcadenas de A (figura 4.26).

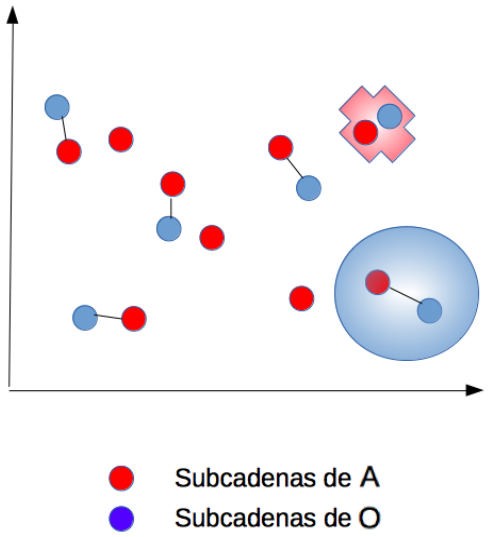

Figura 4.26: En color rojo se muestran las subcadenas que se eliminarán debido a su distancia casi nula entre ambas. En azul se muestran las subcadenas con mayor distancia entre ellas.

Una vez obtenidos los pares de puntos que tienen Tm_{max}, se toma el que cuyo producto punto tenga el valor mínimo y se verifica cuantas veces se repitió ese valor al hacer el producto punto entre el punto que representa esa subcadena de O y los puntos de las subcadenas de A. El mejor oligonucleótido de interferencia será el que tenga el menor número de repeticiones de Tm_{max}.

4.5.2. Complejidad

Para la implementación se realiza un preprocesamiento para las cadenas A y O, cambiando cada letra por las coordenadas que les correspondan, para los casos de las letras A y T, las coordenadas propuestas son: $A = \sqrt{2}000$, $T = 0\sqrt{2}00$, pero debido a que en la implementación en la computadora trabajar con irracionales es complicado, entonces se proponen las coordenadas $A = 1000$, $T = 0100$. Para G y C se tienen las coordenadas: $C = 0002$, $G = 0020$, si estas se cambian por $C = 0001$, $G = 0010$ se puede tener una representación binaria y una vez que se haga el producto punto de cada letra, si el resultado es ≤ 2 (corresponde a las letras A y T que coinciden entre dos cadenas) entonces este se cambia por el valor de 2, y si el resultado es > 2(corresponde a las letras G y C que coinciden entre dos cadenas) este se cambia por 4, con el fin de que sea consistente la definición del problema.

Se describe el algoritmo 10 para el preprocesamiento de la cadena O, el cual es analogo para el preprocesamiento de la cadena A.

Algoritmo 10 Algoritmo para cambiar a coordenadas las subcadenas de la cadena O

```
1: procedure ALGORITMO(O)                                   ▷ O=gen objetivo
2:     for i=1:i≤ ℓ_O:i++ do                                ▷ i es la i-ésima letra de O
3:         switch O_i do
4:             case A
5:                 binario_O = binario_O1000                ▷ Concatena el valor binario de A
6:             end case
7:             case C
8:                 binario_O = binario_O0001                ▷ Concatena el valor binario de C
9:             end case
10:            case G
11:                binario_O = binario_O0010                ▷ Concatena el valor binario de G
12:            end case
13:            case T
14:                binario_O = binario_O0100                ▷ Concatena el valor binario de T
15:            end case
16:        end switch
17:        if i ≥ ℓ then
18:            binariosO[i − ℓ] = binario_O
19:            binario_O << 4                               ▷ Corrimiento en 4 de la cadena binaria
20:        end if
21:    end for
22: end procedure
```

Entonces se lee una letra y se verifica si es A , C , G ó T, en este caso son 3 operaciones por cada letra, más otra de la condición $i \geq \ell$, que a su vez tiene el corrimiento y otra para agragarla a un arreglo, esto toma para la cadena O, $6\ell_O$ operaciones y para la cadena A, $6\ell_A$ operaciones.

El siguiente paso es obtener el producto punto de dos subcadenas (una de A y otra de O), con el fin de conocer su Tm. Si la resta entre está temperatura y la Tm de la subcadena O consigo misma resulta ser menor a 20°C, ya no tiene caso que seguir realizando el producto punto con las demás subcadenas de A porque esa subcadena de O ya no es un buen candidato para ser oligonucleótido de interferencia, ya que se requieren mínimo 20°C para que el oligonucleótido de interferencia pueda separarse del oligonucleótido de A y unirse al gen $Hsp47$.

Se muestra la descripción del algoritmo 11 para la búsqueda del oligonucleótido de interferencia óptimo , su correspondiente tiempo de ejecución y complejidad.

Algoritmo 11 Algoritmo para búsqueda del oligonucleótido de interferencia óptimo

```
1: procedure ALGORITMO(O, A)                                ▷ O=gen objetivo, A=gen actuador
2:     Tm_Oi = 0
3:     Tm_OAj = 0
4:     for i=1:i≤ ℓ_O − ℓ&Tm_Oi − Tm_OAj > 20:i++ do         ▷ i es la i-ésima subcadena de O
5:         Tm_Oi = 0
6:         for j=1:j≤ ℓ:j++ do          ▷ Producto punto de la i-ésima subcadena de O consigo
   misma
7:             multiplica = binarioO_j[i]&biarioO_j[i]   ▷ multiplicalos 4 dígitos binarios de una
   letra de la subcadena de O
```

```
            if multiplica ≤ 2 then
                Tm_Oi = Tm_Oi + 2
            else if multiplica > 2 then
                Tm_Oi = Tm_Oi + 4
            end if
        end for
        for j=1:j≤ ℓ_A − ℓ:j++ do                    ▷ i es la i-ésima subcadena de A
            Tm_OAj = 0
            for k=1:k≤ ℓ:k++ do  ▷ Producto punto de la i-ésima subcadena de O consigo misma
                multiplica = binarioA_k[j]&biarioA_k[j]   ▷ multiplicalos 4 dígitos binarios de una letra de la subcadena de O
                if multiplica ≤ 2 then
                    Tm_OAj = Tm_OAj + 2
                else if multiplica > 2 then
                    Tm_OAj = Tm_OAj + 4
                end if
            end for
            if Tm_OAj > Tm_max then
                Tm_max = Tm_OAj
            end if
        end for
        if Tm_max < Tm_min then
            Tm_min = Tm_max
        end if
    end for
end procedure
```

Analizando lel tiempo de ejecución del algoritmo, se tiene que para calcular la Tm con el producto punto de las cadenas se realizan 4ℓ operaciones. Así mismo se tiene un ciclo de $6\ell(\ell_A-\ell)$ operaciones anidado dentro de otro ciclo, entonces se tienen $[6(\ell_A-\ell)]+[6(\ell_O-\ell)]+[4\ell(\ell_O-\ell)(6\ell(\ell_A-\ell)+2)] = 6\ell_A-6\ell+6\ell_O-6\ell+(4\ell\ell_O-4\ell^2)(6\ell\ell_A-6\ell^2+2)$

$$6\ell_A+6\ell_O-12\ell+(4\ell\ell_O-4\ell^2)(6\ell\ell_A-6\ell^2+2)$$

. Para calcular la cota de complejidad tenemos que $ell_A >> \ell$ y $ell_O >> \ell$, entonces la expresión $[6(\ell_A-\ell)]+[6(\ell_O-\ell)]+[4\ell(\ell_O-\ell)(6\ell(\ell_A-\ell)+2)]$ se reduce a $6\ell_A+6\ell_O+(4\ell\ell_O)(6\ell\ell_A+2)$. Diremos que $\ell = n$, además $\ell < \ell_A$ y $\ell < \ell_O$, entonces $ell_A = k_1 n$, para alguna constante k_1 y $ell_A = k_2 n$, para alguna constante k_2, por lo tanto la expresión escrita anteriormente se puede ver como una función de n como la siguiente $f(n) = 6k_1n + 6k_2n + (4nk_2n)(6nk_1n+2) = 6k_1n+6k_2n+(4k_2n^2)(6k_1n^2+2)$

$$f(n) = 6k_1n+6k_2n+24k_2k_1n^4+8k_2n^2$$

, como $24k_2k_1n^4 > 6k_1n$, $24k_2k_1n^4 > 6k_2n$ y $24k_2k_1n^4 > 8k_2n^2$ $6k_1n+6k_2n+24k_2k_1n^4+8k_2n^2 < 3(24k_2k_1n^4)$, entonces $f(n) < 3(24k_2k_1n^4)$ y como $3(24k_2k_1)$ es una constante entonces la cota superior es

$$O(n^4)$$

Como se puede observar la cota de complejidad es muy grande, pero se espera que se discriminen muchas operaciones y con esto se reduzca el tiempo de ejecución. En la sección de pruebas y resultados se analizará si este algoritmo logro discriminar una buena cantidad de datos.

4.5.3. Lenguaje de programación

Debido a que en algoritmo propuesto se hace uso del producto punto se hizo uso de Octave. Que es un lenguaje de programación de alto nivel, destinado principalmente para cálculos numéricos, provee de una línea de comando para resolver problemas lineales y no lineales numéricamente y para hacer otros experimentos numéricos, además tiene una gran cantidad de herramientas para resolver problemas comunes del álgebra lineal.

4.5.4. Pruebas y resultados

Para saber cuan mejor era el algoritmo propuesto, se comparo el número de operaciones que se realizaron hasta encontrar la solución del problema usando la implementación del algoritmo propuesto, contra con el número de operaciones que se realizaban con la implementación del algoritmo de fuerza bruta. En el algoritmo de fuerza bruta, se verifica la función de Tm evaluada para cada subcadena de O con todas las subcadenas de A, sin hacer ninguna discriminación de datos. Una vez implementados ambos algoritmos se obtuvo que el número de operaciones que se realizan en el algoritmo de fuerza bruta son 1,733,248 y en el algoritmo propuesto se realizan 1,733,248 , que es el mismo número de operaciones, lo que nos indica que no hay discriminación de puntos con la condición propuesta de discriminación, por lo que se analizaron otras propuestas de algoritmos.

4.5.5. Otros algoritmos

Se analizó nuevamente el problema y otra de las ideas que se propuso fue colocar en orden ascendente las subcadenas de la cadena O dependiendo del resultado del producto punto entre la primer subcadena de O y las demás subcadenas de O. Así mismo se ordenan ascendentemente las subcadenas de la cadena A dependiendo del valor del producto punto entre la primer subcadena de O y las subcadenas A. Por lo tanto la primera subcadena de de este conjunto contiene la Tm más grande entre la primera subcadena de O y todas las subcadenas de A. Existen dos límites: límite máximo=$O_1 + A_1$ y límite mínimo=$\mid O_1 - A_1 \mid$. Debido a que están ordenados descendentemente por su distancia se tomara el límite máximo fijo como $O_1 + A_1$. Seguido se compara O_2, cuyo límite mínimo=$\mid O_2 - A_j \mid$, si este valor es menor al límite máximo fijo se pueden seguir comparando los valores de Tm evaluados con las otras subcadenas $A_{j+1}, ...$ hasta que ese valor mínimo sobrepase el límite máximo fijo, en ese momento ya no tiene caso seguir comparando con los demás valores de subcadenas A_j ya que su distancia será más grande y por lo tanto su Tm más pequeño.

En el ejemplo de la figura 4.27, se tienen a dos subcadenas de Hsp47 representadas con puntos, a tres subcadenas de $PPAR\gamma$ representadas con 'X'. La primera subcadena de Hsp47 se encuentra en el centro y a una distancia de las subcadenas de PPARγ (ordenadas ascendentemente) de: 2,5,7. La segunda subcadena de Hsp47 se encuentra a una distancia 1 de la primer subcadena de Hsp47, sin saber la distancia a la que se encuentra la primer subcadena de PPARγ de esta segunda subcadena de Hsp47, se puede obtener que a lo más puede estar a 3 de distancia (1 de la distancia de la primer subcadena de Hsp47 a la segunda subcadena de Hsp47 más 2 de la distancia de la primer subcadena de Hsp47 a la primer subcadena de PPARγ) y a lo menos a 1 de distancia (2 de la distancia de la primer subcadena de Hsp47 a la segunda subcadena de PPARγ menos 1 de la distancia de la primer subcadena de Hsp47 a la primer subcadena de Hsp47), para la segunda subcadena de PPARγ, a lo más puede estar a 6 de distancia y

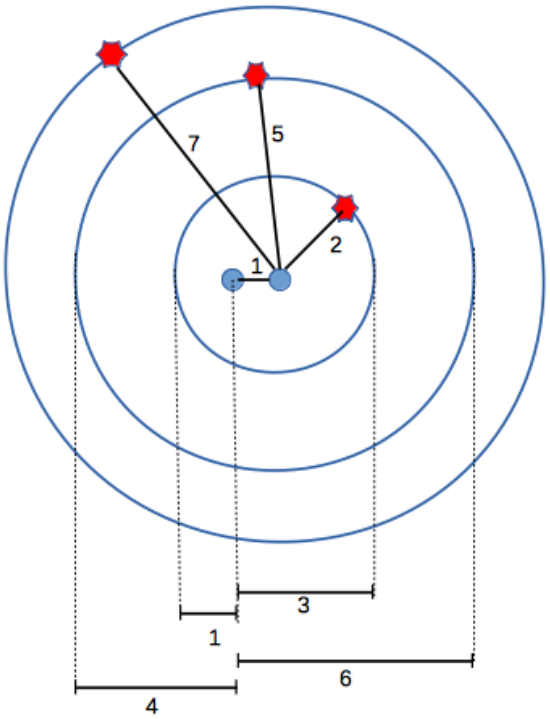

Figura 4.27: Ejemplo.
Representación de las distancias entre subcadenas, dos subcadenas de Hsp47 representadas con puntos y tres subcadenas de $PPAR\gamma$ representadas con 'X'.

a lo menos a 4, 4 es mayor que la distancia al punto que representa la primer subcadena de PPARγ que es 3, por lo tanto, el punto que representa a la segunda subcadena de PPARγ como los siguientes puntos tendrán una distancia mayor a 3, por lo que no tiene caso analizar estos puntos.

4.5.5.1. Pruebas y resultados

Analizando este algoritmo, la cota de complejidad incrementa en $\ell_O \log O + \ell_A \log A$ debido al ordenamiento de los puntos que representan las subcadenas de O y de A. Suponiendo que habrá una gran discriminación de puntos se implemento el algoritmo y como resultado se obtuvo que el número de operaciones que se realizaban eran 1,731,840, comparado con el número de operaciones que se realizaban con el algoritmo de fuerza bruta que eran 1,733,248, se podía concluir que se realizaban 1400 operaciones menos, muy pocas operaciones comparado con el aumento en la cota de la complejidad.

Hasta ahora no se han tenido mejores resultados en cuanto a las cotas de complejidad del enfoque geométrico, la razón es que la diferencia entre los puntos no es mucha, debido a que se propuso la representación de las cadenas con una gran cantidad de dimensiones, para que al realizar el producto punto diera como resultado el valor de la Tm. Esto conlleva a que las distancias que se deseen medir entre un punto con otros puntos van a ser muy similares, por lo que va a ser más difícil discriminar puntos.

Conclusiones

- El algoritmo desarrollado para la búsqueda de una subcadena en una cadena más grande, tiene una cota de complejidad $O(n)$ donde n representa la longitud de la cadena G. Este algoritmo no es aproximado por lo que la solución de encontrar una subcadena en una cadena es exacta. Mejora las cotas de complejidad y el tiempo de proccesamiento de algoritmos usados con fines bioinformáticos como la del algoritmo de Boyer-Moore que en el peor de los casos toma una complejidad de $O(nm)$, donde m es la longitud de la cadena a ser buscada; a pesar de que iguala la cota de complejidad del algoritmo de Knuth-Morris-Pratt, el algoritmo desarrollado es más sencillo de entender y de implementar que el de Knuth-Morris-Pratt. El algoritmo propuesto puede ser usado en la resolución de otros problemas con diferentes fines no bioinformáticos realizando ligeras modificaciones.

- El algoritmo desarrollado para la generación de oligonucleótidos de interferencia competitiva óptimos tiene una cota de complejidad de $O(n^2)$ que mejora la cota de complejidad del algoritmo de fuerza bruta, la cual es $O(n^3)$. Se logro disminuir la complejidad haciendo modificaciones al algoritmo de Smith-Waterman descrito en la literatura.

- Se desarrolló una nueva herramienta para minimizar el riesgo de silenciamiento génico fuera del objetivo en el genoma humano, la cual disminuye el tiempo del cálculo de la temperatura de fusión del oligonucleótido de interferencia con todas las subcadenas del genoma humano; esto gracias a un análisis con la fórmula de la Tm con el que se pudo desarrollar un algoritmo más eficiente, el cual implementa la búsqueda de una cadena que con cierto número de letras distintas podría ser subcadena de una segunda cadena. Gracias a esto, este algoritmo disminuye el tiempo en el que encuentra la respuesta a comparación de un algoritmo de fuerza bruta.

- Se implemento un nuevo enfoque para solucionar el problema con el fin de que la búsqueda del oligonucleótido fuera más rápida, para el cual se analizaron e implementaron distintos algoritmos. Pero debido a que en este enfoque que era geométrico se plantearon una gran cantidad de dimensiones, y debido a la cercanía de los puntos se pudo observar que las distancias que se deseaban medir entre un punto con otros puntos iban a resultar ser muy similares, por lo que la discriminación de puntos era casi nula. Por lo que se concluye que este enfoque hasta ahora no ayuda a disminuir la complejidad del algoritmo propuesto en esta tesis cuya cota de complejidad es de $O(nm)$

- Se desarrolló una aplicación, que agrupa los algoritmos expuestos anteriormente, con el fin de que se utilice para la búsqueda de oligonucleótidos de interferencia, donde el parámetro de temperatura de disociación (Tm) puede ser elegido por el usuario. Con la aplicación se pueden realizar búsquedas de olignucleótidos en la base de datos de preferencia del usuario, ya que la aplicación tiene la capacidad de descargar y usar la base de datos que le sea más útil en ese momento.

- Se proponen para experimentación los siguientes oligonucleótidos de interferencia para inhibir la fibrosis hepática 3'- CCACCATCT**TGTG**GTGGAATTTCTCAT -5' y 3'- CTTGTCGCGGAAGTTGATCTTGGAGT -5'; en dado caso que el primero de ellos no sea funcional, se propone que el motif 3'- TGTG -5' activa la respuesta innata del cuerpo humano y por lo tanto termina desechandolo.

Índice de figuras

Bibliografía

[Abrahamson, 1987] Abrahamson, K. (1987). Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051.

[Adleman, 1994] Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024.

[Albanis et al., 2003] Albanis, E., Safadi, R., and Friedman, S. L. (2003). Treatment of hepatic fibrosis: almost there. *Curr Gastroenterol Rep.*, 5(1):48–56.

[Altschul et al., 1990] Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Mol. Biol.*, 215(3):403–410.

[Amarzguioui and Prydz, 2004] Amarzguioui, M. and Prydz, H. (2004). An algorithm for selection of functional sirna sequences. *Biochem Biophys Res Commun*, 316(4):1050–1058.

[Benenson et al., 2004] Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., and Shapiro, E. (2004). An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429.

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

[Bustos and Poblete, 2002] Bustos, B. and Poblete, P. (2002). Algoritmos y estructuras de datos. `https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/BusqTexto`.

[Chalk et al., 2005] Chalk, A., Wahlestedt, C., and Sonnhammer, E. (2005). Improved and automated prediction of effective sirna. *Bioinformatics.*, 319(1):264–274.

[Dafforn et al., 2001] Dafforn, T., Della, M., and Miller, A. (2001). The molecular interactions of heat shock protein 47 (hsp47) and their implications for collagen biosynthesis. *J Biol Chem*, 276(52):49310–49319.

[Devijver and Kittler, 1982] Devijver, P. H. and Kittler, J. (1982.). Pattern recognition: a statistical approach. *Prentice Hall*.

[Elsharkawy et al., 2005] Elsharkawy, A. M., Oakley, F., and Mann, D. A. (2005). The role and regulation of hepatic stellate cell apoptosis in reversal of liver fibrosis. *Apoptosis*, 10(5):927–939.

[Ensemble, 2000] Ensemble (2000). Human (grch38.p10). `http://www.ensembl.org/Homo_sapiens/Info/Index/`.

[Farfan et al., 2010] Farfan, B., Strausz, R., Kershenobich, D., Mercado, G., and Hernández Urbina, V. (2010). Applying dna computing to diagnose-and-interfere hepatic fibrosis. *Proceedings - 2010 6th International Conference on Natural Computation, ICNC 2010*, 7:3839–3841.

[Fischer and Patrick, 1970] Fischer, F. P. and Patrick, E. A. (1970.). A preprocessing algorithm for nearest neighbor decision rules. *Proceedings of the National Electronic Conference.*, pages 481–485.

[Fortune, 1986] Fortune, S. (1986). A sweepline algorithm for voronoi diagrams. *Proceedings of the second annual symposium on Computational geometry.*, 2:153–174.

[Friedman, 2000] Friedman, S. (2000). Molecular regulation of hepatic fibrosis, an integrated cellular response to tissue injury. *The Journal of biological chemistry*, 275(4):2247–2250.

[Friedman, 2004] Friedman, S. (2004). Mechanisms of disease: mechanisms of hepatic fibrosis and therapeutic implications. *Nat. Clin. Pract. Gastroenterol. Hepatol*, 1(2):98–105.

[Fukunaga and Narendra, 1975] Fukunaga, K. and Narendra, M. (1975). A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers.*, 24(7):750 – 753.

[Hart, 1968] Hart, P. (1968.). The condensed nearest neighbor rule. *IEEE Transactions Information Theory.*, 14(3):515–516.

[Hazra et al., 2004] Hazra, S., Miyahara, T., Rippe, R. A., and Tsukamoto, H. (2004). Ppar gamma and hepatic stellate cells. *Comparative hepatology*, 3(Suppl 1):S7.

[Hernández and Friedman, 2011] Hernández, G. V. and Friedman, S. L. (2011). Pathogenesis of liver fibrosis. *Annu Rev Pathol*, 92(2):103–112.

[Judge et al., 2005] Judge, A., Sood, V., Shaw, J., Fang, D., McClintock, K., and MacLachlan, I. (2005). Sequence dependent stimulation of the mammalian innate immune response by synthetic sirna. *Nat Biotechnol*, 23(4):457–462.

[Kaderali and Schliep, 2002] Kaderali, L. and Schliep, A. (2002). Selecting signature oligonucleotides to identify organisms using dna arrays. *Bioinformatics.*, 18(10):1340–1349.

[Kalantari and McDonald, 1983] Kalantari, I. and McDonald, G. (1983.). A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering.*, SE-9(5):631 – 634.

[King, 2015] King, M. W. (2015). Proliferador de peroxisoma activados los receptores: (ppar). `http://themedicalbiochemistrypage.org/es/ppar-sp.php`.

[Kisseleva and Brenner, 2013] Kisseleva, T. and Brenner, D. A. (2013). Inactivation of myofibroblasts during regression of liver fibrosis. *Cell Cycle*, 12(3):381–382.

[Knuth et al., 1977] Knuth, D., Morris, J. H., and Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350.

[Levenkova et al., 2004] Levenkova, N., Gu, Q., and Rux, J. (2004). Gene specific sirna selector. *Bioinformatics.*, 20(3):30–32.

[López et al., 2007] López, T., Silva, D., López, S., and Arias, C. (2007). Rna de interferencia: el silencio de los genes. *Una ventana al quehacer científico*, 14(CS3):109–118.

[Marmur and Doty, 1962] Marmur, J. and Doty, P. (1962). Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. *J Mol Biol*, 5(1):109–118.

[Masuda et al., 1994] Masuda, H., Fukumoto, M., Hirayoshi, K., and Nagata, K. (1994). Coexpression of the collagen-binding stress protein hsp47gene and the alpha 1(i) and alpha 1(iii)

collagen genes incarbon tetrachloride-induced rat liver fibrosis. *J Clin Invest*, 94(6):2481–2488.

[Miyahara et al., 2000] Miyahara, T., Schrum, L., Rippe, R., Xiong, S., Yee, H. F., Motomura, K., Anania, F. A., Willson, T. M., and Tsukamoto, H. (2000). Peroxisome proliferator-activated receptors and hepatic stellate cell activation. *Journal of Biological Chemistry*, 275(46):35715–35722.

[Nagata, 2003] Nagata, K. (2003). Therapeutic strategy for fibrotic diseasesby regulating the expression of collagen-specific molecularchaperone hsp47. *Nippon Yakurigaku Zasshi*, 121(1):4–14.

[NCBI, 2017] NCBI (2017). How to: Download the complete genome for an organism. `https://www.ncbi.nlm.nih.gov/guide/howto/dwn-genome/`.

[NCBI-AceView, 2012] NCBI-AceView (2012). Aceview downloads. `https://www.ncbi.nlm.nih.gov/IEB/Research/Acembly/Download/Downloads.html`.

[Nishino et al., 2003] Nishino, T., Miyazaki M, Abe, K., Furusu, A., Mishima, Y., Harada, T., Ozono, Y., Koji, T., and Kohno, S. (2003). Antisense oligonucleotides against collagen-binding stressprotein hsp47 suppress peritoneal fibrosis in rats. *KidneyInt*, 64(3):887–896.

[Páramo Hernández et al., 2010] Páramo Hernández, D. B., Otero Regino, W., and Pineda Ovalle, L. F. (2010). Fibrogénesis hepática. *SciELO Analytics*, 25(2):187–197.

[Reynolds et al., 2004] Reynolds, A., Leake, D., Boese, Q., Scaringe, S., Marshall, W., and Khvorova, A. (2004). Rational sirna design for rna interference. *Nat Biotechnol*, 22(3):326–330.

[Sato et al., 2008] Sato, Y., Murase, K., Kato, J., Kobune, M., Sato, T., Kawano, Y., Takimoto, R., Takada, K., Miyanishi, K., Matsunaga, T., Takayama, T., and Niitsu, Y. (2008). Resolution of liver cirrhosis using vitamin a-coupled liposomes to deliver sirna against a collagen-specific chaperone. *Nat Biotechnol.*, 26(4):431–442.

[Shabalina et al., 2006] Shabalina, S. A., Spiridonov, A. N., and Ogurtsov, A. Y. (2006). Computational models with thermodynamic and composition features improve sirna design. *Bioinformatics*, 7:65.

[She et al., 2005] She, H., Xiong, S., Hazra, S., and Tsukamoto, H. (2005). Adipogenic transcriptional regulation of hepatic stellate cells. *The Journal of Biological Chemistry*, 280(6):4959–4967.

[Smith and Waterman, 1981] Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Mol. Biol.*, 147(1):195–197.

[Suárez Cuenca et al., 2008] Suárez Cuenca, J. A., Chagoya de Sánchez, V., Aranda Fraustro, A., Sánchez Sevilla, L., Martínez Pérez, L., and Hernández Muñoz, R. (2008). Partial hepatectomy-iduced regeneration accelerates reversion of liver fibrosis involving participation of hepatic stellated cells. *Exp Biol Med (Maywood)*, 233(7):827–839.

[Sunamoto et al., 1998] Sunamoto, M., Kuze, K., Tsuji, H., Ohishi, N., Yagi, K., Nagata, K., Kita, T., and Doi, T. (1998). Antisenseoligonucleotides against collagen-binding stress protein hsp47 suppress collagen accumulation in experimental glomerulonephritis. *Lab Invest*, 78(8):967–972.

[Sung and Lee, 2003] Sung, W. and Lee, W. (2003). Fast and accurate probe selection algorithm for large genomes. *IEEE Computer Society Bioinformatics Conference (CSB)*, 2:65–74.

[Tou and Gonzalez, 1974] Tou, J. T. and Gonzalez, R. C. (1974.). Pattern recognition principles. *Addison Wesley.*

[Trautwein et al., 2015] Trautwein, C., Friedman, S. L., Schuppan, D., and Pinzani, M. (2015). Hepatic fibrosis: Concept to treatment. *Journal of Hepatology*, 62(1):S15–S24.

[Ukkonen, 1995] Ukkonen, E. (1995). Online construction of suffix trees. *Algorithmica*, 14(3):249–260.

[Vidal, 1986] Vidal, E. (1986). An algorithm for finding nearest neighbours in (approximately) constant average time complexity. *Pattern Recognition Letters*, 4:145–157.

[Vidal, 1994] Vidal, E. (1994). New formulation and improvements of the nearest-neighbour approxi- mating and eliminating search algorithm (aesa). *Pattern Recognition Letters*, 15(1):1–7.

[Vinuesa, 2007] Vinuesa, P. (2007). Dinámica para alns. globales y locales. `http://www.ccg.unam.mx/~vinuesa/Cursos2RMBF/PDFs/C1/Material_supl1_programacion_dinamica.pdf`.

[Wallace et al., 1979] Wallace, R. B., Shaffer, J., Murphy, R., Bonner, J., Hirose, T., and Itakura, K. (1979). Hybridization of synthetic oligodeoxyribonucleotides to phi chi 174 dna the effect of single base pair mismatch. *Nucleic Acids Res*, 6(11):3543–3557.

[Wang and Mu, 2004] Wang, L. and Mu, F. (2004). A web-based design center for vector-based sirna and sirna cassette. *Bioinformatics.*, 20(11):118–120.

[Wang et al., 2003] Wang, Z., Inokuchi, T., Nemoto, T., Uehara, M., and Baba, T. (2003). Antisense oligonucleotide against collagen-specific molecular chaperone 47-kda heat shock proteinsuppresses scar formation in rat wounds. *Plast ReconstrSurg*, 111(6):1980–1987.

[Yamada and Morishita, 2005] Yamada, T. and Morishita, S. (2005). Accelerated off-target search algorithm for sirna. *Bioinformatics*, 21(8):1316–1324.

[Zheng et al., 2003] Zheng, J., Close, T., Jiang, T., and Lonardi, S. (2003). Efficient selection of unique and popular oligos for large est databases. *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM2003) Springer*, 20(13):2101–2112.