UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES

ACATLAN

COMPUTADORAS DE ADN

T E S I S

QUE PARA OBTENER EL TITULO DE:

LICENCIADA EN MATEMATICAS APLICADAS Y COMPUTACION

PRESENTA:

ANA DELIA GAYOSSO BECERRIL

ASESOR: **MTRO. VICTOR JOSE PALENCIA GOMEZ**

MARZO DEL 2005

FES ACATLAN U.N.A.M.
DEPTO. DE TÍTULOS
UNIDAD DE ADMINISTRACIÓN ESCOLAR
PROFESIONALES Y CERTIFICACIÓN

m.342477

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.
NOMBRE: Gamboa Becerril Ana Delia
FECHA: 01/04/05
FIRMA:

Agradecimientos

Si tuviera que hacer una lista de todos aquéllos a los que tengo algo que agradecer, cabrían casi todos los nombres que existen

Gracias a mi madre por su apoyo y su ayuda incansables, a mi hermano por ser el motivo que tantas veces ha dado sentido a mi vida, a mi padre por sus sabios consejos y a los tres por ser mi mayor inspiración para realizar este trabajo.

Gracias también al maestro Víctor Palencia por su ayuda que fue un importante incentivo para mi trabajo constante en este proyecto y a la maestra MariCarmen González por sus observaciones y por su calidad profesional y humana.

Igualmente, agradezco al maestro Alfonso Vilchis por su disposición para resolver hasta mis más elementales dudas de Biología y a la doctora Laura Ongay por sus amables explicaciones sobre la manipulación del ADN.

De la misma manera, mi agradecimiento para todos los maestros que he tenido a lo largo de mi vida académica por todas sus enseñanzas y consejos.

Asimismo, agradezco a todos los que han contribuido a hacer de MAC, de Acatlán y de la UNAM las comunidades entusiastas y comprometidas a las que me enorgullece sobremanera pertenecer.

En particular, a todos los amigos con los que compartí las aulas universitarias y que de una u otra manera contribuyeron a que este trabajo tomara forma con sus sugerencias, comentarios y apoyo, muchas gracias.

Índice general

Introducción

Este trabajo versa sobre las computadoras de ADN, cuya importancia radica en ser un tópico de actualidad en cuanto a las tendencias en computación, además de ser un tema interdisciplinario puesto que para la comprensión de su funcionamiento es necesario conocer los conceptos fundamentales de la genética y la computación

De manera más concreta, el objetivo de este trabajo es mostrar algunas ventajas de las computadoras de ADN con relación a las computadoras electrónicas, particularmente que la complejidad del tiempo de solución de un problema específico (el problema SAT o de la *satisfactibilidad*) se reduce de exponencial a polinomial utilizando computadoras de ADN en lugar de computadoras electrónicas, aun cuando la cantidad de moléculas de ADN necesarias para resolverlo crece de manera exponencial.

La importancia de este análisis se debe a la necesidad de justificar el esfuerzo y el costo implicados en la construcción de computadoras de ADN mediante afirmaciones objetivas acerca de las diferencias que favorecen a estos dispositivos sobre las computadoras electrónicas dado que las cantidades de dinero y tiempo necesarias para el desarrollo dichas computadoras no pueden considerarse despreciables como se observará posteriormente.

Este trabajo se sustenta en conceptos de lógica matemática, teoría de la computación, teoría de gráficas y probabilidad, y a través de su desarrollo se muestra la aplicabilidad de diversos conceptos matemáticos para el estudio de las ventajas ofrecidas por las computadoras de ADN.

De esta forma, en el Capítulo 1 se ubicará a dichas computadoras como un nuevo paradigma de computación, se describirán la estructura del ADN y las técnicas que permiten manipularlo para construir las computadoras en cuestión y se presentarán las características que debe tener un problema para que sea conveniente resolverlo mediante computadoras de ADN.

En el Capítulo 2 se analizarán diversos conceptos utilizados por la teoría de la complejidad para comparar diferentes paradigmas de computación, entre ellos los tipos de máquinas de Turing, la complejidad temporal, la notación de la "O grande", los problemas P y NP y la relación de estos últimos con las computadoras de ADN.

En el Capítulo 3 se discutirán las soluciones propuestas a dos problemas NP: la computadora de Adleman para resolver el problema del agente viajero y la propuesta de Lipton para resolver el problema SAT.

En el Capítulo 4 se presentarán y analizarán tres modelos; el primero para calcular el tiempo de solución del problema SAT en una computadora electrónica, el segundo para calcular dicho tiempo en una computadora de ADN que utilice el algoritmo propuesto por Lipton y el tercero para calcular el volumen mínimo de ADN requerido para implementar dicho algoritmo.

Finalmente, en el Capítulo 5 se analizarán las ventajas y desventajas de las computadoras de ADN con relación a las computadoras electrónicas, y se proporcionará un panorama general sobre el futuro de las computadoras de ADN.

Capítulo 1

ADN computacional

Hay un libro abierto siempre para todos los ojos: la naturaleza.

Jean Jacques Rousseau

1.1. Biología computacional

Las distintas ramas de la ciencia siempre han estado ligadas. Esta situación se manifiesta en el caso de la biología computacional, también conocida como biocomputación, computación molecular, computación biomolecular, bioinformática y computación biológica.

La biología computacional une a la biología con la computación y se ha utilizado en dos sentidos. El primero, para denominar al estudio del uso de los sistemas de cómputo actuales para almacenar y procesar información genómica sobre los seres vivos, y hacer inferencias sobre la misma. Y el segundo, para referirse a la construcción de computadoras moleculares o biológicas [19].

Una computadora de este tipo es un dispositivo en el que las entradas se codifican a través de moléculas biológicas, que son manipuladas con técnicas de biología molecular para provocar reacciones químicas con las que se obtengan ciertos resultados codificados también en moléculas biológicas.

Aunque estamos acostumbrados a pensar en las computadoras como dispositivos electrónicos con una pantalla, un teclado y un disco duro, una computadora es un dispositivo que acepta entradas, sigue instrucciones para

manipular dichas entradas y a partir de esta manipulación, obtiene resultados; sin intervención humana en al menos alguno de sus procesos[20].

En este sentido, las computadoras biológicas realmente pueden considerarse como tales porque las reacciones provocadas con técnicas de biología molecular sobre las moléculas de entrada se llevan a cabo sin intervención humana.

Richard Feynman fue el primero en considerar la posibilidad de construir computadoras a escala molecular; presentó su idea durante una conferencia en la reunión anual de la Sociedad Americana de Física del 29 de diciembre de 1959 en el Instituto Tecnológico de California (Caltech) [15].

Sin embargo, la factibilidad de llevar a cabo computaciones a nivel molecular se probó hasta 1994 cuando el desarrollo de la biología molecular permitió la construcción de la primera computadora biológica desarrollada por Leonard M. Adleman usando ADN[1] [1].

Además de ADN, las computadoras biológicas pueden utilizar como materiales de construcción algunas proteínas [7]. Así, las computadoras de ADN son un tipo especial de computadora biológica, a las que también se identifica con los nombres de ADN computacional, computación por ADN y cómputo molecular basado en ADN.

Su construcción y manipulación requiere de ADN sintético, enzimas y otros materiales, además de un laboratorio que cuente con los dispositivos necesarios para aplicar técnicas de biología molecular al ADN.

Estas computadoras pueden considerarse un nuevo paradigma de computación, es decir una forma diferente de analizar, diseñar e implementar sistemas computacionales, porque al utilizar nuevos materiales para codificar, almacenar y procesar la información, implican una nueva forma de análisis, diseño e implementación de sistemas computacionales.

Para entender el funcionamiento de las computadoras de ADN es necesario entender de manera general la estructura del ADN y las técnicas para manipular dicha molécula, tópicos que se analizarán en las siguientes dos secciones para finalizar este capítulo con un análisis de la naturaleza de los problemas adecuados para su solución en computadoras de ADN.

[1] ADN: ácido desoxirribonucleico

1.2. Estructura del ADN

El ADN es una molécula que en forma natural se encuentra dentro del núcleo de las células vivas (*in vivo*). Su importancia radica en que contiene la codificación para la producción de las proteínas necesarias para que las células se desarrollen y en ser la única molécula capaz de replicarse a sí misma, garantizando así que las particularidades de una célula pasen a sus descendientes y con ello que se transmitan las características hereditarias de generación en generación, además de realizar otras funciones celulares [21].

En cuanto a su estructura, el ADN es una macromolécula[2] constituida por unidades llamadas desoxirribonucleótidos o nucleótidos.

Cada desoxirribonucleótido está formado por un azúcar llamada desoxirribosa, un grupo[3] fosfato (P) y una base nitrogenada. La desoxirribosa tiene 5 átomos de carbono; numerándolos de 1' a 5', el grupo fosfato se une al carbono 5' y la base se une al carbono 1'. Además hay un grupo hidroxilo (OH) unido al carbono 3' [7].

Hay dos tipos de bases: purinas y pirimidinas. Las purinas, a su vez, pueden ser adenina (A) o guanina (G) y las pirimidinas pueden ser citosina (C) o timina (T). Así, a los nucleótidos se les llama A, G, C y T, dependiendo del tipo de base que tengan. En la figura 1.1, se representa, de manera simplificada, la estructura de un nucleótido con B como la base correspondiente, P el grupo fosfato, la línea que une los puntos 1' a 5' como el azúcar con sus 5 carbonos, y OH como el grupo hidroxilo.

Dos nucleótidos se pueden unir a través de un enlace químico conocido como *enlace fosfodiester*. Este tipo de enlace es un enlace covalente[4] que se forma entre el grupo hidroxilo de un nucleótido y el grupo fosfato de otro, de manera que une el carbono 5' de la desoxirribosa de un nucleótido con el carbono 3' de la desoxirribosa del otro nucleótido.

La molécula resultante, por tanto, tiene un grupo fosfato libre en el

[2]Una macromolécula es una molécula gigante formada por gran número de átomos [14]. En general, se considera que están formadas por más de 1000 átomos [9].

[3]En química, un grupo es un conjunto específico de átomos que le dan propiedades químicas particulares a los compuestos de los que forman parte. Los grupos también se conocen como *radicales*[14]. Siendo sinónimos *grupo* y *radical*, utilizaré el primer término porque este trabajo está dirigido principalmente a lectores que no están familiarizados con la terminología química.

[4]Dos de los principales enlaces químicos son el covalente y el iónico. En el enlace covalente los átomos que se unen comparten electrones, mientras que en el enlace iónico los electrones son cedidos por unos átomos y ganados por otros [8]

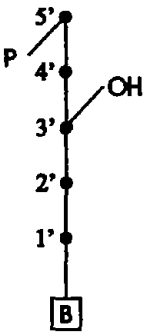

Figura 1.1: Representación de un nucleótido

carbono 5' y un grupo hidroxilo libre en el carbono 3'. Dependiendo del nucleótido que se tome como el primero y de si éste tiene libre el fosfato en el carbono 5' o el grupo hidroxilo en el carbono 3', la molécula tendrá una dirección 5'−3' ó 3'−5'. Con el enlace fosfodiester se forman cadenas simples de ADN, que pueden representarse mediante el número inicial de su dirección y una secuencia de letras que identifica a los nucleótidos que la conforman [7]. Así, la cadena de la figura 1.2 se puede identificar como 5'−ACG.

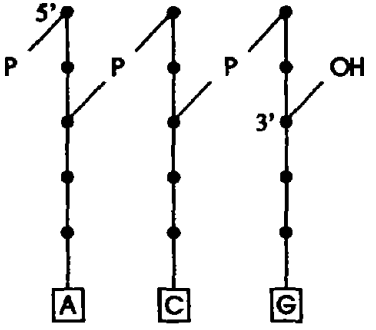

Figura 1.2: Representación de la cadena simple 5'−ACG

Los nucleótidos también pueden enlazarse cuando la base de uno de los nucleótidos se une a la base del otro, mediante *enlaces de hidrógeno*[5] que sólo pueden formarse entre la adenina y la timina o entre la citosina y la

[5]En un enlace o puente de hidrógeno dicho elemento sirve de vínculo entre dos átomos electronegativos al encontrarse unido a uno de ellos mediante un enlace covalente y ser atraído por el otro

guanina.

Por esta razón se dice que la A y la T, al igual que la C y la G son complementarias y a este principio se le conoce como *complementariedad Watson-Crick* en honor a James D. Watson y Francis H. C. Crick, quienes dedujeron la estructura del ADN en 1953 [35], basándose en estudios de otros investigadores como Rosalind Franklin y Maurice Wilkins. Por este descubrimiento, en 1962 Watson, Crick y Wilkins ganaron el premio Nobel de Fisiología o Medicina.

El tipo de enlace formado entre nucleótidos complementarios es mucho más débil que el que se forma entre los grupos fosfato e hidroxilo y sólo se vuelve estable cuando ocurre entre cadenas largas de ADN; generalmente un tamaño de 18 a 20 nucleótidos es suficiente, aunque también influye la temperatura del medio en el que se encuentre el ADN y la cantidad de enlaces A−T con relación a la cantidad de enlaces G−C, ya que estos últimos involucran la formación de 3 enlaces de hidrógeno, mientras que los primeros sólo provocan la formación de dos enlaces de hidrógeno de manera que una mayor presencia de pares G−C hace que la molécula sea más estable[6]. Mediante este tipo de enlace se forman las cadenas dobles de ADN. En la figura 1.3 se representa la unión entre las cadenas 5'−ACG y 3'−TGC con las líneas dobles y triples entre los nucletidos simbolizando enlaces dobles y triples, respectivamente.

Por otra parte, en realidad las cadenas dobles de ADN sufren una especie de torsión, formando una doble espiral o doble hélice, especie de escalera, como la de la figura 1.4.

Una representación más simple de esta unión, que será utilizada en los siguientes capítulos, es:

5'-ACG
3'-TGC

En general, la secuencia de los nucléotidos, es decir, el orden en que se presentan en una molécula de ADN determina la información genética de un ser vivo. De esta forma, los genes tienen como principal componente fragmentos de ADN que determinan una característica del organismo, aunque también contienen secciones de ADN cuyas funciones todavía no se conocen del todo. Al compactarse, los genes forman los cromosomas [10].

[6]Alfonso Vilchis Peluyera. Laboratorio de Biología Molecular. Facultad de Ciencias, UNAM: Comunicación personal.

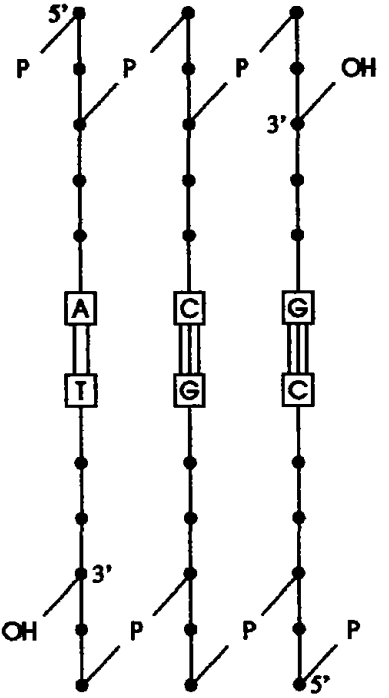

Figura 1.3: Representación de la molécula doble formada al unirse 5'−ACG con 3'−TGC

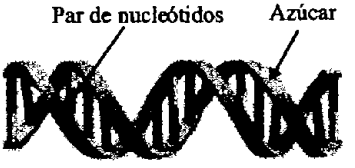

Figura 1.4: Doble hélice

1.3. Técnicas de manipulación

Para poder entender el funcionamiento de las computadoras de ADN es necesario conocer de manera general las técnicas utilizadas para manipular el ADN fuera de las células o *in vitro*. Con la intención de presentar un panorama de la manera en que estas técnicas funcionan, a continuación se describen algunas de las manipulaciones que se aplican a las moléculas de ADN cuando se utilizan para realizar cómputos, aunque como siempre existe más de una forma de lograr el mismo objetivo y las presentadas aquí están en constante mejoramiento; además puede haber otras que logren los mismos resultados con menos recursos, en menos tiempo o de manera más sencilla.

En términos generales, las técnicas que se utilizan corresponden al desacoplamiento de moléculas dobles en moléculas sencillas (*desnaturalización*), a la construcción de cadenas dobles (*renaturalización*), la localización de secuencias específicas (*búsqueda de moléculas*), su copiado (*PCR*), la síntesis, corte y ligado de cadenas, la determinación de su longitud (*electroforesis*) y la determinación exacta de su secuencia de nucleótidos (*secuenciación*).

Para aplicar estas técnicas, además de algunos dispositivos, se requieren enzimas. Las *enzimas* son proteínas que sirven como catalizadores de las reacciones químicas que ocurren en las células, aumentando la velocidad de dichas reacciones; generalmente actúan sobre un tipo específico de reacción, aunque pueden actuar sobre dos o más [14].

1.3.1. Desnaturalización y renaturalización

Dado que los enlaces entre dos cadenas complementarias son más débiles que los enlaces entre nucleótidos consecutivos en una cadena simple, se pueden romper los primeros manteniendo los segundos, mediante un proceso llamado desnaturalización que consiste en calentar la solución de ADN hasta que se separen las cadenas dobles, lo cual ocurre a temperaturas entre 85° C y 95° C, o a una temperatura menor si están presentes ciertos químicos que aceleren el proceso como es el caso de la formamida[7] [27].

El proceso inverso se conoce como renaturalización o hibridización, y se logra bajando la temperatura de la solución lentamente para dar tiempo a que los nucleótidos complementarios se encuentren.

[7] Líquido aceitoso e incoloro, soluble en agua y alcohol. Se usa como disolvente y en síntesis orgánica [14].

1.3.2. Corte

Para acortar cadenas de ADN se usan enzimas llamadas DNA *nucleasas*, que se dividen en *exonucleasas* y *endonucleasas* [27].

Las *exonucleasas* van quitando nucleótido por nucleótido de los extremos de las moléculas de ADN. Pueden hacerlo en la dirección 3'−5' ó 5'−3' o ambas y pueden ser para moléculas dobles, simples o ambas, dependiendo de la enzima de la que se trate. Por ejemplo, la enzima *Exonucleasa III* remueve los nucleótidos en la dirección 3'−5' en moléculas dobles, actuando como se observa en la figura 1.5, donde α representa una cadena doble de ADN.

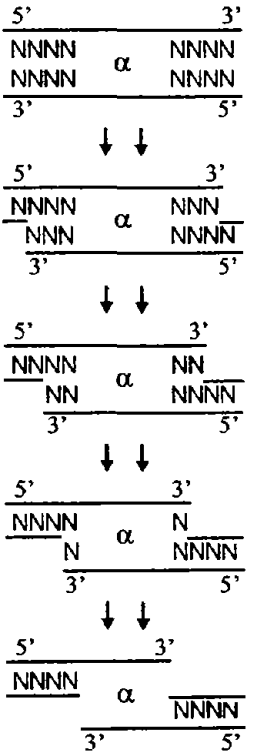

Figura 1.5: Acción de la enzima *Exonucleasa III*

Las *endonucleasas*, en cambio, destruyen los enlaces internos en las moléculas de ADN, dividiéndolas. Cada una de las enzimas de este tipo,

como en el caso anterior, puede ser para cadenas simples, dobles o ambas y en el caso de las enzimas que cortan cadenas dobles, pueden hacer cortes escalonados o rectos.

Además, las *endonucleasas* pueden ser *con reconocimiento* o *sin reconocimiento*. Cuando son sin reconocimiento pueden cortar en cualquier sitio, mientras que cuando son con reconocimiento solamente cortan cadenas dobles y se llaman así porque tienen una sección de reconocimiento; pueden ser de tipo I o tipo II, dependiendo de si cortan fuera o dentro de esta región de reconocimiento.

Las *endonucleasas* con reconocimiento tipo II son las más usadas porque se sabe exactamente dónde cortarán la cadena y generalmente sus sitios de reconocimiento son palíndromos, es decir, al leer una cadena en una dirección se obtiene la misma secuencia que al leer la otra cadena en la misma dirección, mientras que para el tipo I no se sabe con exactitud en qué lugar fuera de la sección de reconocimiento cortarán.

En las células vivas, las enzimas con reconocimiento defienden al organismo contra organismos invasores, cortando el ADN del invasor que tiene la sección de reconocimiento, pero como el organismo a proteger también puede tener secciones con la secuencia de reconocimiento, se protege a sí mismo a través de enzimas llamadas *modificadoras*, que tienen el mismo sitio de reconocimiento y cuando llegan a él lo modifican agregando o quitando algunos componentes químicos como el grupo fosfato o el grupo hidroxilo de los extremos de la cadena de ADN para que de esta forma la región modificada ya no sea identificable para la enzima con reconocimiento.

Algunos ejemplos de *endonucleasas* son:

S1: solamente puede cortar cadenas simples y no tiene una región de reconocimiento, es decir, puede cortar en cualquier sitio, como se ve en la figura 1.6.

XmaI: Es una endonucleasa con reconocimiento y de tipo II. La sección de reconocimiento es 5'−CCCGGG y el corte es escalonado como se ve en la figura 1.7.

SmaI: Tiene las mismas características que XmaI, pero el corte es recto como se ve en la figura 1.8.

HgaI: Es una endonucleasa con reconocimiento y de tipo I. La sección de reconocimiento es 5'−GACGC y el corte es escalonado como se ve en la figura 1.9.

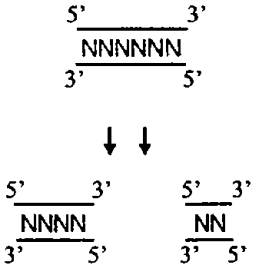

Figura 1.6: Acción de la enzima *S1*

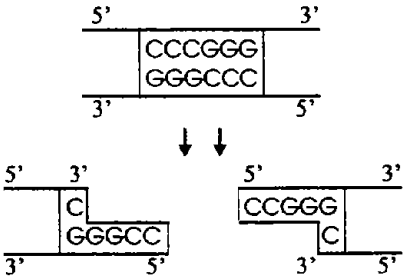

Figura 1.7: Acción de la enzima *XmaI*

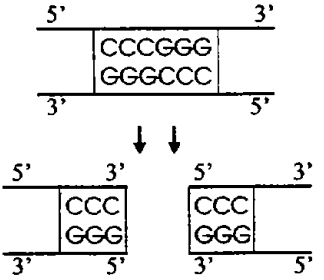

Figura 1.8: Acción de la enzima *SmaI*

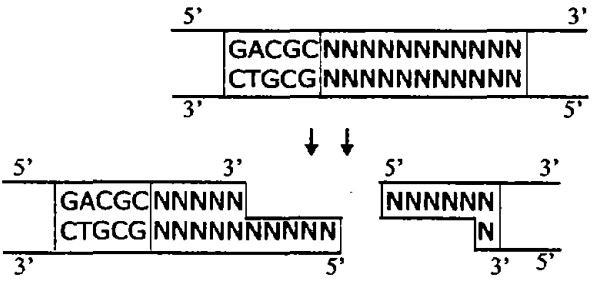

Figura 1.9: Acción de la enzima *HgaI*

1.3.3. Búsqueda de moléculas

Existen varios métodos para localizar secuencias específicas de ADN en una solución donde además haya otro tipo de secuencias. En esta sección se describirá uno de los más utilizados: el método de *Southern blot*[8].

Para aplicarlo, primero se debe garantizar que las cadenas se encuentren en forma simple, aplicando para ello la técnica de desnaturalización. Después a la solución que contiene todas las cadenas se le agregan secuencias complementarias a la *sonda*, nombre con el que se conoce a la secuencia que se está buscando.

Las cadenas complementarias a la *sonda* se marcan con una sustancia fluorescente bajo la acción de una luz ultravioleta, ya que de manera natural las moléculas de ADN son invisibles. En el proceso de marcado de las cadenas de ADN se utilizan dos enzimas modificadoras. La primera enzima modificadora utilizada, fosfatasa alcalina, remueve el grupo fosfato de las terminaciones 5' de la cadena de ADN y en su lugar coloca grupos OH. Trabaja en un medio alcalino, es decir, en un medio donde se encuentren presentes muchos grupos OH. Después se colocan grupos fosfato marcados previamente de manera radioactiva con bromuro de etidio que es fluorescente bajo la acción de una luz ultravioleta y que se intercala entre las bases del ADN, y la segunda enzima, quinasa polinucleótida, retira las terminaciones 5'−OH para colocar en su lugar los grupos fosfato marcados [27]. El proceso se observa en la figura 1.10 para una cadena doble de ADN representada por α.

De esta forma cuando las moléculas complementarias a la *sonda* se unan

[8]Alfonso Vikhis Peluyera. Laboratorio de Biología Molecular. Facultad de Ciencias, UNAM: Comunicación personal.

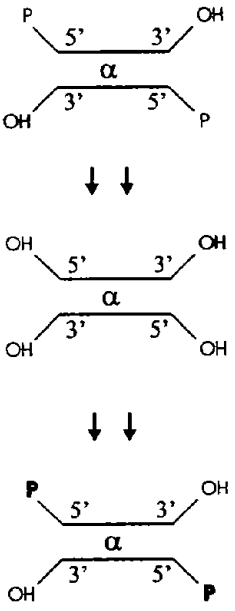

Figura 1.10: Proceso para marcar una cadena de ADN

a las cadenas que tengan esta secuencia, las cadenas formadas serán localizables bajo la acción de la luz ultravioleta para ser aisladas del resto y aplicarles un proceso de desnaturalización con el objeto de obtener sólo las moléculas que contienen las secuencias que se buscaban.

1.3.4. Síntesis

Para sintetizar una cadena simple de ADN se fija el extremo 3' del primer nucleótido de la cadena a un soporte fijo, puesto que la reacción química que añadirá el resto de los nucleótidos lo hará en la dirección 3'−5' de manera que al quedar libre el extremo 5', el extremo 3' del nuevo nucleótido formará un enlace fosfodiester con él.

La secuencia deseada se logra colocando la cadena en una solución que sólo contenga el tipo de nucleótido que se requiera en ese momento, esperando el tiempo que dura la síntesis[9] y repitiendo el proceso en otra solución que contenga sólo el tipo de nucleótido que se desea siga en la cadena.

A partir de esta cadena simple se puede sintetizar una cadena doble, sintetizando primero una secuencia complementaria a la cadena simple original y esperando a que se una a ésta. Para acelerar el proceso se puede sintetizar una cadena complementaria solamente en los primeros nucleótidos de la cadena simple a partir del extremo 5' para que una vez que se haya unido a ésta se utilice una enzima llamada *polimerasa* que agrega nucleótidos en la dirección 5'−3', uno por uno, a una cadena simple con una secuencia unida a parte de esta cadena y con el extremo 3' libre para extensión [27].

En la figura 1.11 se muestra un ejemplo de este proceso y en la figura 1.12 se observa uno de los tipos de equipo utilizado para sintetizar cadenas simples de ADN. Los cuatro frascos pequeños marcados como A, C, G y T contienen los nucleótidos correspondientes; el primer nucletido de la cadena a sintetizar se encuentra en el contenedor marcado como E y la cadena resultante se encuentra en el frasco marcado como S. Los frascos de mayor tamaño contienen soluciones necesarias para la formación de los enlaces fosfodiester.

[9]En el Instituto de Fisiología Celular de la UNAM tarda aproximadamente 5 minutos. Laura Ongay. Unidad de Biología Molecular. Instituto de Fisiologia Celular, UNAM: Comunicación personal.

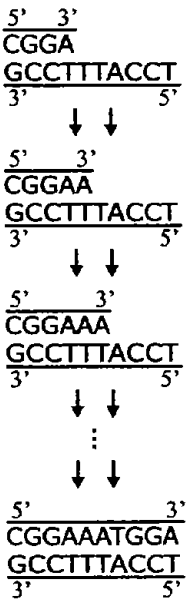

Figura 1.11: Acción de una enzima polimerasa

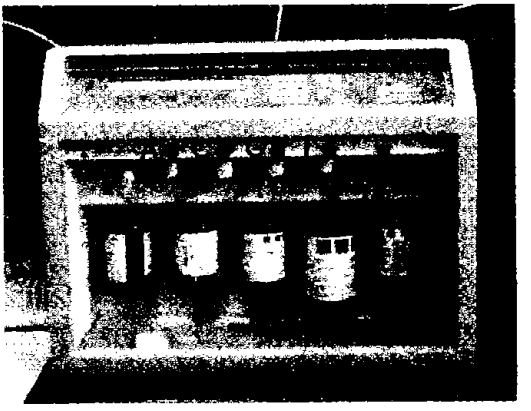

Figura 1.12: Equipo utilizado en la síntesis de cadenas simples de ADN

1.3.5. Ligado

La unión de moléculas separadas mediante cualquiera de los procesos descritos en la sección 1.3.2 se conoce como ligado y utiliza enzimas llamadas *ligasas* [27].

En el caso de las moléculas que resultan de cortes escalonados, si se mantienen suficientemente cerca, se formarán enlaces de hidrógeno entre los nucleótidos complementarios, y mediante el ligado se formará el enlace fosfodiester, faltante entre los nucleótidos consecutivos que se separaron con el corte.

El proceso se observa en la figura 1.13 y se puede usar para obtener moléculas híbridas, es decir, que una parte de la molécula resultante del ligado haya sido producida por un corte y otra parte por otro corte, siempre que sus extremos salientes sean complementarios.

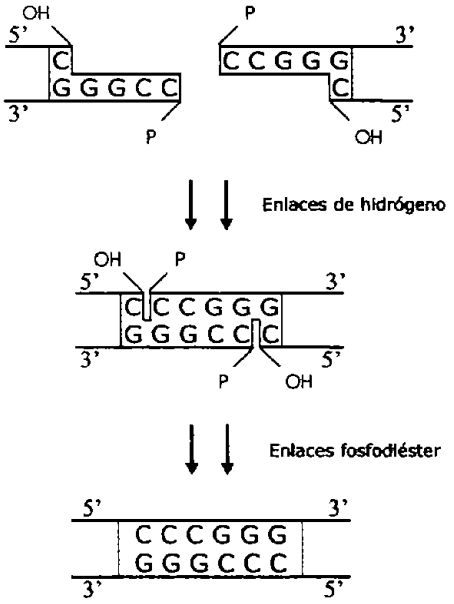

Figura 1.13: Proceso de ligado

Cuando las moléculas resultan de cortes rectos, el proceso de ligado es más difícil para las ligasas porque las moléculas a unir no están necesariamente tan próximas antes de que actúen las enzimas, como en el caso anterior en que esta proximidad está garantizada por los enlaces de hidrógeno entre

los extremos complementarios. La ventaja es que en este caso se pueden unir moléculas independientemente de sus secuencias terminales, ya que no requieren ser complementarias.

1.3.6. Electroforesis

La longitud de una cadena de ADN es el número de nucleótidos o el número de pares de bases o *bp* (base pair) que la componen dependiendo de si se trata de una cadena simple o doble. Una técnica para medir la longitud de una cadena de ADN es la electroforesis [27].

Para aplicar esta técnica se coloca gel de *agarosa*[10] o *poliacrilamida*[11] caliente en un contenedor de vidrio y se deja enfriar insertando una especie de peine como el que se observa en la figura 1.14 en una de las orillas para que al retirarlo se formen cavidades donde se coloquen las moléculas de ADN.

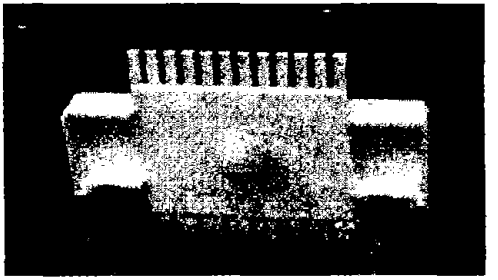

Figura 1.14: Instrumento utilizado en la técnica de electroforesis

Dado que en forma natural las moléculas de ADN son invisibles, antes de colocarlas en el gel se tiñen con bromuro de etidio. Para lograrlo se utiliza el proceso descrito en la sección de búsqueda de moléculas.

Después de colocar el ADN teñido en las cavidades, se activa un campo eléctrico con su electrodo positivo en la orilla opuesta a la de las cavidades y, dado que los fosfatos tienen carga negativa, las moléculas de ADN también tienen una carga negativa proporcional a su longitud, por lo que se mueven a través del gel hacia el electrodo positivo con una rapidez inversamente

[10] La agarosa es un azúcar que se obtiene de ciertas algas y se usa de manera común en la industria para formar geles [14].

[11] La poliacrilamida es un polímero, es decir, una macromolécula formada por unidades repetitivas llamadas monómeros, que forma geles con poros de menor tamaño que la agarosa [14].

proporcional a su longitud.

Así, para conocer la longitud de las cadenas, se colocan moléculas de ADN de diferentes longitudes, pero conocidas, en una de las cavidades (a esta mezcla de cadenas de longitud conocida se le llama *marcador*) y el campo eléctrico se detiene cuando las moléculas más cortas alcanzan el electrodo positivo, pudiendo comparar la posición en que quedaron las cadenas de longitud desconocida con la posición de las cadenas de longitud conocida para de esta forma determinar el tamaño de las primeras, sabiendo que cadenas del mismo tamaño se mueven a las mismas velocidades y quedan en la misma posición.

Finalmente, el ADN se puede recuperar, cortando la parte del gel donde se encuentra para después congelarlo con nitrógeno líquido, destruyendo así la estructura del gel para permitir que al derretirse se pueda centrifugar con un filtro especial de tal forma que se conserve solamente el ADN. La recuperación de las moléculas de ADN también se puede lograr con un equipo formado por columnas que tienen una resina con carga positiva a las que, por atracción, se pega el ADN .

Utilizando la concentración de gel y el equipo adecuados, con esta técnica es posible distinguir entre cadenas cuya longitud difiera solamente en un nucleótido.

1.3.7. PCR

Para hacer muchas copias de un fragmento específico de una cadena doble de ADN se utiliza la técnica llamada PCR (*Polymerase Chain Reaction*; reacción en cadena de la polimerasa) desarrollada por Kary Mullis en 1985 [24] y cuya importancia permitió que Mullis recibiera el Premio Nobel de Química en 1993.

Para llevar a cabo esta técnica es necesario que se conozcan las secuencias de los extremos opuesto de las cadenas simples que forman la molécula a amplificar o *cadena objetivo*.

Se prepara una solución conteniendo agua doblemente destilada (ddH2O), la molécula objetivo y cadenas simples que sean complementarias a los extremos conocidos de dicha cadena, nucleótidos de los 4 tipos y una polimerasa llamada Taq, que es aislada de una bacteria con necesidad de temperatura elevada para su desarrollo y que, por lo mismo, es resistente al calor.

Un ciclo de esta técnica consiste en aplicar desnaturalización para separar la cadena objetivo en cadenas simples y enfriar la solución hasta aproximadamente 55° C para que las secuencias complementarias a los extremos de la molécula objetivo se unan a dichos extremos de las cadenas simples resultantes de la desnaturalización [27].

Finalmente, un tipo especial de polimerasas extiende las cadenas que resultan de la unión anterior, produciendo dos moléculas idénticas a la molécula objetivo. Las cadenas complementarias a los extremos conocidos de la molécula objetivo se conocen como *primers*. El proceso completo se observa en la figura 1.15.

Por tanto, en el ciclo n se obtienen 2^n copias de la cadena en cuestión. Es decir, esta técnica es muy eficiente ya que permite producir millones de copias en un periodo muy corto de tiempo.

Una desventaja de esta técnica es que como un *primer* se une a una de las cadenas simples que resultan de la desnaturalización y el otro *primer* se une a otra de las cadenas, cuando la cadena objetivo se encuentra en la misma solución que otras cadenas de diferente secuencia pero que empiecen o terminen con la misma secuencia conocida de la molécula objetivo, de estas cadenas (que sólo cumplen con empezar en la misma secuencia o terminar en la misma secuencia, pero no con ambas) también se obtendrá una copia.

Es decir, no se amplificarán en el sentido de que se obtenga más de una copia de ellas, pero en cada ciclo se obtendrá por cada cadena doble, una cadena doble igual y una cadena simple que empieza con la misma secuencia, pero que no se amplificará la próxima vez; en el ciclo n se tendrán 1 cadena doble y n cadenas simples, como se observa en la figura 1.16.

1.3.8. Secuenciación

Para conocer la secuencia exacta de nucleótidos de una molécula de ADN se utiliza un método conocido como método de Sanger, en honor a su inventor [30], o método de dideoxigenina por usar nucleótidos llamados dideoxinucleótidos, en los que se ha cambiado el grupo hidroxilo unido al carbono 3' del azúcar por un átomo de hidrógeno. Los dideoxinucleótidos se denotan como ddA, ddT, ddC y ddG, dependiendo de su base nitrogenada.

Para aplicar este método, primero se extiende la cadena que se desea leer en la dirección 5'−3', añadiéndole una secuencia conocida, γ. Después se colocan secuencias complementarias a esta secuencia conocida que están marcadas radiactivamente, $\overline{\gamma}$, obteniendo moléculas como la que se obser-

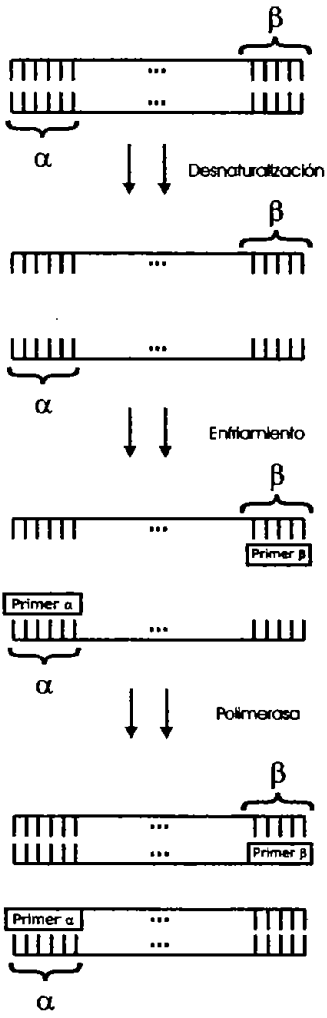

Figura 1.15: Un ciclo de la técnica PCR

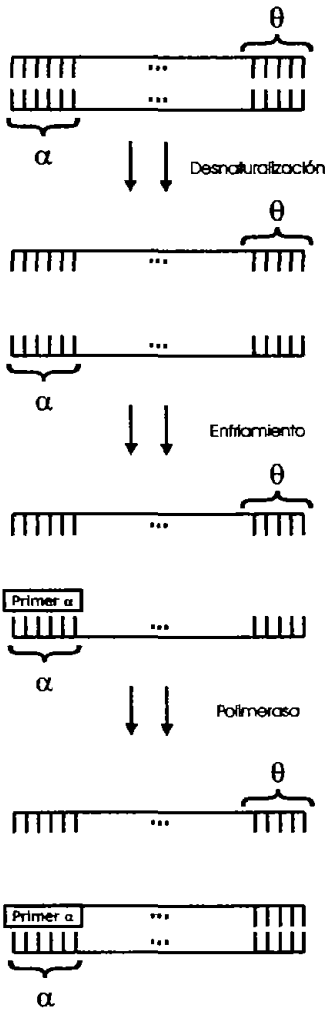

Figura 1.16: PCR en una molécula con una secuencia inicial conocida

va en la figura 1.17 donde la cadena cuya secuencia se quiere conocer es 3'−AGTACGGGACGC.

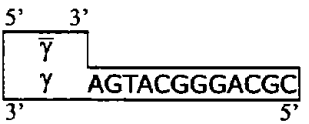

Figura 1.17: Primer paso en la secuenciación de una molécula

Después se preparan cuatro tubos (Tubo A, Tubo T, Tubo C y Tubo G), que contienen una cantidad determinada cuidadosamente de ddA, ddT, ddC y ddG, dependiendo del tubo del que se trate (ddA para el tubo A, ddT para el tubo T, etc.), polimerasa (generalmente la llamada fragmento Klenow de la polimerasa I de ADN) y nucleótidos A, T, C y G.

Así, la polimerasa extenderá la molécula en la dirección 5'−3', pero si en lugar de usar un nucleótido usa el correspondiente dideoxinucleótido su trabajo con esa molécula se detendrá porque la falta del grupo hidroxilo en el carbono 3' del azúcar del dideoxinucleótido impedirá que se forme un enlace fosfodiester con el que debería ser el siguiente nucleótido de la cadena. De esta forma, para la cadena de la figura 1.15 anterior, para el tubo A se formarán las siguientes moléculas:

5'−$\overline{\gamma}$ TCATGCACTGCG

5'−$\overline{\gamma}$ TCATGCA

5'−$\overline{\gamma}$ TCA

Para el tubo T:

5'−$\overline{\gamma}$ TCATGCACTGCG

5'−$\overline{\gamma}$ TCATGCACT

5'−$\overline{\gamma}$ TCAT

5'−$\overline{\gamma}$ T

Para el tubo C:

5'−$\overline{\gamma}$ TCATGCACTGCG

5'−$\overline{\gamma}$ TCATGCACTGC

5'−$\overline{\gamma}$ TCATGCAC

5'−$\overline{\gamma}$ TCATGC

5'−$\overline{\gamma}$ TC

Y para el tubo G:

5'−$\overline{\gamma}$ TCATGCACTGCG

5'−$\overline{\gamma}$ TCATGCACTG

5'−$\overline{\gamma}$ TCATG

Después se desnaturalizan las moléculas anteriores y se selecciona a aquéllas que inicien con la secuencia marcada radiactivamente para después aplicarles la técnica de electroforesis en gel, usando cuatro cavidades, una para cada tubo como se observa en la figura 1.18, donde la letra colocada bajo la representación de los fragmentos de ADN denota el nucleótido con el que termina dicho fragmento.

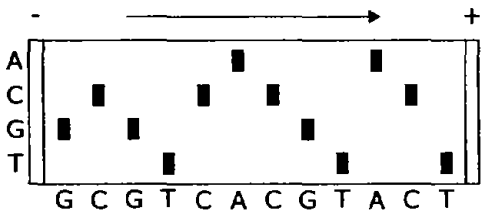

Figura 1.18: Secuenciación de la cadena 3'−AGTACGGGACGC

Dado que, a mayor longitud, menor velocidad en la técnica de electroforesis en gel, los fragmentos de derecha a izquierda son:

5'−$\overline{\gamma}$ T,

5'−$\overline{\gamma}$ TC,

5'−$\overline{\gamma}$ TCA,

5'−$\overline{\gamma}$ TCAT,

5'−$\overline{\gamma}$ TCATG,

5'−$\overline{\gamma}$ TCATGC,

5'−$\overline{\gamma}$ TCATGCA,

5'−$\overline{\gamma}$ TCATGCAC,

5'−$\overline{\gamma}$ TCATGCACT,

5'−$\overline{\gamma}$ TCATGCACTG,

5'−$\overline{\gamma}$ TCATGCACTGC, y

5'− TCATGCACTGCG.

Por tanto la molécula de la que se deseaba conocer la secuencia debe ser

complementaria a ésta, es decir debe ser 3'−AGTACGTGACGC.

La cantidad de nucleótidos ddA, ddT, ddC y ddG debe ser determinada cuidadosamente porque entre mayor cantidad de ellos haya, mayor será la probabilidad de que todas las moléculas de ADN obtenidas terminen en el primer nucleótido A, T, C ó G, de la cadena [27].

1.4. Problemas adecuados para las computadoras de ADN

Las computadoras de ADN se construyen utilizando las técnicas mencionadas en la sección anterior. En principio, podrían resolver cualquier tipo de problema computacional, pero su capacidad de operar sobre varias moléculas al mismo tiempo, las vuelve idóneas para resolver problemas en los que el propósito es decidir si las entradas tienen una propiedad particular y el método más seguro de solución conocido implica formar subconjuntos de un conjunto finito teniendo en cuenta el número y el orden de sus elementos para después realizar una búsqueda exhaustiva entre estas combinaciones hasta encontrar la que resuelva el problema, es decir, la que muestre que las entradas tienen la propiedad buscada.

Estos problemas se conocen como problemas combinatorios o de búsqueda por su forma de solución y como problemas de decisión por su propósito [3]. Un ejemplo de ellos es el siguiente: dado un conjunto finito S de enteros positivos y un objetivo t, también entero positivo, responder si existe un subconjunto de S cuya suma sea t [26].

La capacidad de las computadoras de ADN para resolver problemas combinatorios se debe a que pueden actuar sobre muchas combinaciones al mismo tiempo, a diferencia de las computadoras electrónicas secuenciales que primero deben intentar obtener todas las soluciones posibles y después comenzar a descartar las que no resuelvan el problema.

Por esta razón, la mayoría de los algoritmos propuestos o implementados en computadoras de ADN para resolver problemas de búsqueda consisten en generar todas las soluciones potenciales al problema y luego remover las que no sean soluciones reales del mismo. El paso que aprovecha el paralelismo es la generación de soluciones potenciales ya que que todas se generan al mismo tiempo.

Con el desarrollo de este trabajo se observará que las computadoras de ADN pueden competir con las computadoras electrónicas en paralelo en la

resolución de un problema en particular (el problema SAT). Los problemas combinatorios para los cuales una solución potencial puede ser probada con facilidad son llamados problemas NP y tratados por la teoría de la complejidad, tema que se abordará en el siguiente capítulo.

Capítulo 2

Teoría de la complejidad

La única posibilidad de descubrir los límites de lo posible es aventurarse un poco más allá de ellos.

Arthur C. Clarke

La superioridad de un paradigma de computación respecto a otro se puede analizar con tres enfoques diferentes [26]:

Universalidad: Estudia si una computadora del primer paradigma puede simular cualquier otra computadora del mismo paradigma o del paradigma con el que se está comparando.

Computabilidad: Estudia si una computadora del primer paradigma puede realizar computaciones que una computadora del otro paradigma no puede.

Complejidad: Estudia si una computadora representante del primer paradigma de computación puede realizar las mismas tareas que una computadora del otro paradigma de manera más eficiente, lo cual en este caso significa que requiere un cantidad significativamente menor de tiempo o espacio.

En este trabajo se compararán las computadoras de ADN con las computadoras clásicas[1] utilizando el enfoque de complejidad porque la eficiencia

[1]Las computadoras clásicas o convencionales son aquellas que funcionan aplicando algoritmos a información codificada en bits .

de un paradigma en estos términos es muy importante en la práctica, puesto que el hecho de que un problema sea computable, es decir, que exista un algoritmo que permita resolverlo en una cantidad finita de tiempo, no garantiza que se pueda resolver en la práctica si requiere de mucho tiempo de computación o de una gran cantidad de memoria.

Para comenzar se explicará brevemente el concepto de la máquina de Turing, un modelo matemático idealizado que tiene las características esenciales de una computadora y que ha permitido a los científicos de la computación probar teoremas acerca de los recursos utilizados por las computadoras y de los límites de sus capacidades.

2.1. La máquina de Turing

Un modelo de computación es una definición abstracta de computadora que permite analizar fácilmente los requerimientos de tiempo o espacio de un algoritmo ignorando detalles de implementación. Diferentes modelos de computación difieren en su capacidad de cómputo, es decir, algunos modelos pueden realizar computaciones imposibles en otros modelos y también pueden diferir en el costo de sus operaciones.

Algunos de estos modelos son la máquina determinista de Turing, el modelo formal más simple de computación y una de las bases de la computación clásica, algunas otras variantes de la máquina de Turing y la computación en paralelo, que es un sistema con dos o más procesadores internos capaces de procesar un cómputo de manera colectiva.

Los principios de la computación fueron establecidos en los años 30 del siglo pasado por Alonzo Church y Alan Turing, entre otros, como respuesta al problema conocido como *entscheidungsproblem* (problema de la decisión o de la *decidibilidad*) que fue propuesto por el matemático alemán David Hilbert en 1900 en el Congreso Internacional de Matemáticas llevado a cabo en París, junto con otros 22 problemas.

El nombre de *entscheidungsproblem* viene de que en alemán la palabra para decisión es *entscheidung* y el problema pregunta si hay un procedimiento mecánico por el cual pueda decidirse sobre la verdad o falsedad de cualquier conjetura o cuestión matemática [26].

Para resolver este problema Turing modeló el proceso de prueba que sigue un matemático cuando intenta probar una conjetura. Encontró que se requiere un conjunto de reglas de transformación para llevar un enunciado

matemático a otro, un método para registrar cada paso del proceso, la capacidad de combinar las últimas inferencias con las anteriores y un mecanismo para decidir qué regla aplicar en un momento dado.

Turing simplificó este proceso de manera que se pudiera construir una máquina para imitarlo. Para ello, sustituyó la lista de símbolos matemáticos por cadenas de símbolos de un alfabeto que contiene un número finito de elementos.

La prueba sería escrita en una larga tira de papel o cinta dividida en celdas idénticas donde sólo se pudiera escribir un símbolo o, si fuera el caso, se dejaría en blanco. El movimiento a lo largo de la cinta se lograría a través de una cabeza de lectura y escritura.

Además, la máquina podría encontrarse en diferentes estados, que simularían el contexto en el que el conjunto de símbolos es visto por el matemático que intenta probar una conjetura. La combinación del símbolo que estuviera siendo leída por la cabeza y el estado de la máquina determinarían el símbolo a escribir en la cinta, la dirección en que se movería la cabeza y el estado en que se encontraría.

Con esta abstracción, Turing demostró que no hay un procedimiento mecánico por el cual la verdad o falsedad de cualquier conjetura o cuestión matemática pueda ser decidida. Church también llegó a la misma conclusión mediante otro modelo equivalente[2], pero el de mayor influencia fue el desarrollado por Turing, que se conoce en su honor como máquina de Turing o máquina determinista de Turing.

Así, Turing y Church colocaron las bases de la ciencia de la computación, ya que la máquina de Turing es un modelo del proceso de cómputo independiente de la forma física en que las computadoras son implementadas en realidad.

En resumen, una máquina determinista de Turing consiste en una cinta de longitud infinita equivalente a la memoria de una computadora digital y que está dividida en celdas (bits) en las cuales se pude escribir 0 ó 1 o dejar la celda en blanco, en una cabeza de lectura y escritura que se puede mover a lo largo de la cinta leyendo el contenido de una celda a la vez, un control finito de estados (que es un conjunto finito de estados internos en que se puede encontrar la máquina) y un programa o conjunto de instrucciones que especifica, dado el estado interno actual y el bit que está siendo leído,

[2]Los modelos de Turing y Church son equivalentes puesto que cada modelo computacional es capaz de simular al otro.

cómo debe cambiar el estado, si el bit debe ser cambiado y la dirección en que se debe mover la cabeza.

El control finito de estados está formado por los estados $q_1, q_2, \ldots, q_m$. Además están los estados q_s y q_h, que indican el inicio de la computación o la terminación de ésta, respectivamente.

La cinta de la máquina de Turing se extiende de manera infinita en una dirección. Inicialmente, la cinta tiene un símbolo como ∇ que indica su inicio en la primera celda, una secuencia de los caracteres 0 y 1 que representan al programa y, ocasionalmente, algunos datos iniciales mientras que el resto de la cinta está en blanco.

Después se siguen las instrucciones del programa y cuando el estado actual es q_h la salida de la computación se encuentra en las celdas no vacías de la cinta [6]. En la figura 2.1 se muestra un esquema de la máquina.

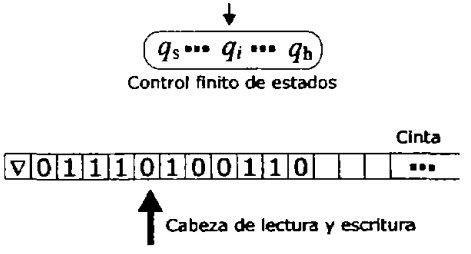

Figura 2.1: Esquema de una máquina determinista de Turing

Las instrucciones de un programa para la máquina de Turing son de la forma (q, x, q', x', s), donde q es un estado del conjunto de estados internos de la máquina, x es uno de los símbolos del alfabeto que pueden aparecer en la cinta y s es un indicador que orientará el movimiento de la cabeza de lectura y escritura.

Así, en cada paso, la máquina de Turing busca a través de las instrucciones del programa una línea cuyo primer elemento sea el estado actual de la máquina y cuyo segundo elemento sea el símbolo que está siendo leído en la cinta. Si no existe esa instrucción, el estado interno de la máquina cambia a q_h, pero si la instrucción se encuentra, el estado interno de la máquina cambia a q', el símbolo x es sobrescrito con el símbolo x' y la cabeza se mueve a la derecha, a la izquierda o no se mueve, dependiendo del valor de s: -1, $+1$, ó 0, respectivamente.

Además, existe una Máquina Universal de Turing que puede simular

cualquier otra máquina de Turing. La importancia de esta variante de la máquina de Turing deriva de una tesis establecida independientemente por Church y Turing y que se conoce como la tesis Church-Turing la cual establece que la clase de funciones computables por una Máquina Universal de Turing corresponde exactamente a la clase de funciones que vemos como computables por un algoritmo. Desde el establecimiento de esta tesis hacia 1936 [34, 11] no se ha encontrado evidencia de lo contrario y ha servido como base a la ciencia de la computación por darle una definición rigurosa al concepto de algoritmo.

También existen las Máquinas Probabilísticas de Turing, adecuadas para resolver problemas computacionales de búsqueda que tienen un gran número de soluciones potenciales, pero sólo una solución verdadera, los cuales en la mayoría de los casos serían resueltos tomando mucho tiempo en una máquina determinista porque ésta tendería a examinar casi todas las soluciones potenciales antes de encontrar la verdadera.

Para evitar esto se introduce la aleatoriedad en las máquinas de Turing originando la máquina probabilística o no determinista, que sólo difiere de la máquina determinista de Turing en que le da la posibilidad de hacer una elección aleatoria sobre el estado interno al que cambiará. Esta máquina puede simularse por la máquina Universal de Turing eligiendo de manera secuencial todos los estados entre los que se hace la elección aleatoria en una máquina probabilística de Turing, pero generalmente llegar al resultado tomará más tiempo.

El concepto de la máquina de Turing es fundamental para la ciencia de la computación en general y en particular para la teoría de la complejidad porque le permite tener un modelo universal para el cual se calculan los recursos requeridos en la solución de problemas particulares

2.2. Enfoque de complejidad

La complejidad computacional, teoría de la complejidad o, simplemente, complejidad estudia la eficiencia con que los problemas computacionales pueden ser resueltos con relación al poder computacional que requieren sus algoritmos de solución.

Los dos tipos de complejidad son la complejidad temporal que mide la complejidad de los algoritmos en términos de su tiempo de solución y la complejidad espacial que mide sus requerimientos de memoria. En ambos

casos los problemas y sus correspondientes algoritmos son clasificados de acuerdo a los recursos necesarios para resolverlos.

De esta forma, la complejidad de un algoritmo está dada por sus requerimientos de tiempo y espacio, mientras la complejidad de un problema se define como la complejidad mínima de todos los algoritmos que resuelven el problema, por lo que la complejidad de un algoritmo también puede definirse como un límite superior para la complejidad del problema resuelto por ese algoritmo, ya que siempre existir la posibilidad de encontrar un algoritmo más eficiente.

Diferentes algoritmos para resolver un mismo problema pueden compararse para elegir al más eficiente, es decir, al que aproveche mejor los recursos para resolver el problema en cuestión. De la misma forma pueden compararse diferentes problemas con relación a sus mejores algoritmos de solución para saber cuáles requieren menores recursos al resolverse.

El tiempo de ejecución de un algoritmo depende de la naturaleza de las entradas, del modelo computacional utilizado, de la computadora en que se resolverá y del tamaño de las entradas.

En cuanto a la naturaleza de los datos, la estrategia seguida es la de analizar la eficiencia del algoritmo en el peor caso, es decir, la eficiencia de aquellos ejemplares de un tamaño fijo de entrada en los que el algoritmo debe realizar mayor número de operaciones y, por tanto, emplea más tiempo. De esta forma se obtiene una cota superior del tiempo de ejecución para cualquier tipo de entradas.

Además, diferentes modelos computacionales requieren diferentes recursos para resolver los mismos problemas como se puede observar en las diferencias de tiempo para resolver problemas de búsqueda entre las máquinas determinista y probabilística de Turing.

En cuanto a la dependencia de la computadora utilizada, se sigue el criterio de ignorar todo efecto que pudiera tener porque si se mantiene constante el tamaño de las entradas y la naturaleza de los datos, garantizando así que el número de operaciones básicas que se realizarán es el mismo independientemente de la computadora utilizada, el tiempo de ejecución del algoritmo será el número de operaciones multiplicado por el tiempo utilizado para realizar cada operación por la computadora en que se implemente, de manera que diferentes implementaciones de un mismo algoritmo diferirán solamente en constantes multiplicativas positivas.

Así, los requerimientos de tiempo y memoria no se miden en cantida-

des absolutas para evitar las variaciones en rendimiento provocadas por las diferencias en cantidades de memoria RAM y velocidades de procesamiento de diferentes computadoras, características que además varían año con año. De esta forma, la complejidad de un algoritmo es una medida intrínseca del problema que se pretende resolver y no de la computadora en que se resolverá pues si éste fuera el caso esta medida tendría que actualizarse constantemente.

Para evitar estas dependencias, en lugar de medir el tiempo preciso que tomará resolver el problema se mide el número de operaciones básicas necesarias para resolverlo, ya que el tiempo necesario para resolverlo depende de manera directa del número de operaciones que deban realizarse.

La selección de las operaciones básicas a considerar se debe hacer de tal manera que el número total de operaciones sea aproximadamente proporcional al número de operaciones básicas, con lo cual aunque con cierta imprecisión, se podrá distinguir a los algoritmos que efectúan un número de operaciones drásticamente diferente con entradas grandes.

Se recomienda descartar las instrucciones de inicialización, el establecimiento de apuntadores, el incremento de índices en los ciclos, entre otras [3]. Por ejemplo, para encontrar un número x en un arreglo, se puede considerar como operación básica la comparación de x con un elemento del arreglo, ignorando la lectura del número de elementos del arreglo y la de los elementos en sí, así como la actualización de índices en el ciclo utilizado para comparar x con cada elemento del arreglo.

Así como el tiempo de ejecución depende directamente del número de operaciones básicas, éste último estará determinado por el tamaño del problema. Este aspecto se considera importante porque a diferencia de la dependencia sobre la computadora que se elija para resolver el problema, un aumento en el tamaño no aumenta el número de operaciones, y por ende el tiempo, en un factor constante. Por ejemplo, el doble de tamaño no requiere necesariamente el doble de operaciones.

Más bien, la dependencia puede ser lineal, si es proporcional a n, cuadrática si es proporcional a n^2, etc., siendo n el tamaño del problema. El tamaño del problema se determina con base en la naturaleza del mismo, aunque generalmente es tomado como el número de bits necesarios para especificarlo en la computadora.

Las mismas consideraciones son aplicables al consumo de memoria o de otros recursos que utilizan los algoritmos.

La teoría de la complejidad provee de límites para las cantidades de tiempo y espacio que requiere el mejor algoritmo posible para resolver un problema computacional dado. Utilizando la información sobre estos límites, la teoría de la complejidad clasifica diferentes problemas y los algoritmos que los solucionan de acuerdo a su complejidad, en general, en términos de la tasa de crecimiento del tiempo de corrida o la memoria utilizados por el algoritmo para resolver un problema cuando el tamaño de éste se incrementa.

La clasificación básica de los algoritmos en complejidad computacional está fundamentada en complejidad temporal, es decir, la complejidad está medida en términos del tiempo y los clasifica en aquéllos que utilizan una cantidad de tiempo limitada por una función polinomial en n y en aquellos algoritmos que utilizan recursos que crecen más rápido que cualquier función polinomial en n.

En este último caso, las funciones tienen como límite inferior una función que no es polinomial y se llaman funciones super-polinomiales, pero como el tiempo requerido generalmente está limitado por una función exponencial se dice que los problemas requieren tiempo exponencial, aunque en este concepto también se incluyen funciones como $n!$ y n^n que tienen un crecimiento de orden mayor y funciones como $n^{\log n}$ que no crecen tan rápido como una función exponencial, pero sí más rápido que una función polinomial [18].

Los problemas cuyo mejor algoritmo conocido para el peor caso es de tiempo polinomial son vistos como problemas fáciles, tratables o factibles porque generalmente pueden ser resueltos en una cantidad viable de tiempo, mientras que los que requieren un tiempo exponencial se ven como difíciles, intratables o no factibles porque intentar calcular su solución se vuelve poco factible rápidamente con el incremento en el tamaño del problema independientemente de lo rápida que sea la computadora donde se resolverá o de la cantidad de procesadores en paralelo que se utilice.

En la mayoría de los casos se piensa que son intratables porque los algoritmos conocidos utilizan cantidades de tiempo que crecen más rápido que cualquier función polinomial, aunque nunca haya sido probado que no existe un mejor algoritmo, es decir, no se ha probado que el tiempo de solución del problema tenga un límite inferior exponencial.

Sin embargo, la factibilidad de un algoritmo en términos prácticos no se decide siempre con base en esta clasificación. Por ejemplo, si hay dos algoritmos que resuelven un mismo problema, uno es exponencial requiriendo $2^{n/1000}$ operaciones y el otro polinomial requiriendo n^{1000} operaciones y el tamaño del problema es 1000 bits , resultará mucho más eficiente el algoritmo

exponencial que el polinomial, puesto que el primero requerirá 2 operaciones, mientras que el segundo requerirá $1000^{1000} = 10^{3000}$, habiendo una diferencia radical entre ambos que favorece al algoritmo que en la clasificación de teoría de la complejidad es visto como difícil, intratable o no factible.

En este caso, el algoritmo polinomial sólo se vería favorecido para tamaños mayores a 10^8 y aunque, casi siempre, la clasificación de los algoritmos interesa para entradas grandes del problema, en raras ocasiones se utilizan algoritmos polinomiales de grado 1000 y, casi siempre, los recursos están limitados por algoritmos de grados bajos, por lo que en la práctica los algoritmos polinomiales se comportan generalmente de manera más eficiente que los exponenciales aun para problemas de tamaño pequeño [26].

Además, la importancia de la clasificación de los problemas computacionales en polinomiales y exponenciales tiene su fundamento en la tesis fuerte de Church-Turing, una versión más detallada de la tesis Church-Turing, la cual establece que cualquier función computada en un modelo de computación diferente a la máquina probabilística de Turing podría computarse en dicha máquina con un incremento a lo más polinomial en el número de operaciones requeridas. Es decir, cualquier modelo de computación puede ser simulado en una máquina probabilística de Turing con un incremento a lo más polinomial en el número de operaciones requeridas [4].

Algunas razones que dan soporte a la tesis fuerte de Church-Turing son que parece difícil imaginar otro método que se encuentre fuera de la descripción de la máquina probabilística de Turing, que para cada noción matemática de computabilidad que ha sido propuesta se ha probado su equivalencia a la definición de computabilidad establecida por Turing y que no hay ejemplos de dispositivos de computación que no puedan ser simulados por una máquina probabilística de Turing [3].

Sin embargo, esta tesis ha sido desafiada por científicos de la computación, físicos y lógicos, en el sentido de que podría encontrarse algún dispositivo que fuera capaz de computar alguna función que no puede ser computada por ninguna de las variantes de la máquina de Turing, es decir que puede existir algún dispositivo que pueda computar una función considerada incomputable por no poder ser computada en una máquina de Turing.

Si este dispositivo fuera encontrado se podría aprovechar para realizar nuevas computaciones que no se han podido realizar hasta ahora, aunque con ello sería necesario revisar la definición de computabilidad y de ciencia de la computación [26].

Por otra parte, la tesis fuerte de Church-Turing es importante porque la

máquina probabilística de Turing es, en muchos casos, mucho más eficiente que la máquina determinista, por lo que si un problema no tiene una solución con recursos polinomiales en una máquina probabilística de Turing, no tiene solución en ningún dispositivo de cómputo.

Además, si se encuentra una solución polinomial para cualquier modelo de computación, sólo se requerirá un incremento polinomial para llevar a cabo la computación en la máquina probabilista de Turing. De esta forma se identifica a una solución eficiente con una solución que utiliza recursos polinomiales considerando cualquier modelo de computación, lo que convierte a la clasificación polinomial *vs.* intratable independiente del modelo con el que se esté trabajando.

Sin embargo, no es sencillo probar que un problema dado requiere cantidades de tiempo polinomiales o exponenciales y, más que pruebas, se tienen conjeturas sobre la clase a la que pertenecen muchos problemas. Una clase de complejidad está definida por un conjunto de problemas que tienen en esencia los mismos límites para los recursos que requiere su solución. Se conocen varias clases de complejidad y se conocen o sospechan algunas relaciones importantes entre ellas. A continuación se dará una explicación más profunda de las clases P y NP, dado que los problemas computacionales que se abordan en este trabajo son problemas NP.

2.3. Clases de complejidad

Los límites establecidos por la teoría de la complejidad para los recursos utilizados por un algoritmo que resuelve un problema se especifican a través de una notación asintótica que se utiliza para estudiar el peor comportamiento de un algoritmo en particular y que conserva el comportamiento esencial de la cantidad de recursos necesarios para resolver un problema con relación a su tamaño.

Un caso de esta notación es la notación O, que establece límites superiores en el comportamiento de una función. Así, se dice que $f(n)$, que va de los naturales a los reales, está en la clase de las funciones $O(g(n))$ o que $f(n)$ es $O(g(n))$ si existen constantes c y n_0 mayores que 0, tales que para todos los valores de n mayores que n_0, $|f(n)| \leq c|g(n)|$ [7].

$O(g(n))$ es el conjunto de funciones que cumplen con la condición anterior y se llama "O grande" u "O de g"; el representante del conjunto se elige generalmente como la función más sencilla del mismo; por ejemplo, el

conjunto de complejidad lineal será el $O(n)$, el de complejidad cuadrática $O(n^2)$, etc. Si una función $f(n)$ pertenece a la clase $O(g(n))$, se dice que "f es O de $g(n)$" [26].

En el estudio de la complejidad de un algoritmo las funciones comparadas mediante esta notación representan el crecimiento de los recursos en función del tamaño de las entradas, por lo que $f(n)$ y $g(n)$ se consideran positivas.

De esta forma, una función $f(n)$ pertenece a la clase $O(g(n))$ si está acotada superiormente por algún múltiplo constante de $g(n)$ para valores de n suficientemente grandes y sin considerar posibles constantes multiplicativas.

Por tanto, una función $f \in O(g)$ si:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty \tag{2.1}$$

Por ejemplo, la función $3n^3 + n$ está en la clase $O(n^3)$ porque $3n^3 + n \leq 4n^3$ para todo n entero positivo. En general se cumple que para todo polinomio en n de grado m, $P(n^m)$, existen c y n_0, tales que se cumple que $a_m n^m + \ldots + a_1 n + a_0 \leq cn^m$ para $n > n_0$ donde c siempre puede elegirse como $|a_m| + \ldots + |a_1| + |a_0|$ ya que:

$$a_m + \ldots + a_1 \frac{1}{n^{m-1}} + a_0 \frac{1}{n^m} \leq |a_m| + \ldots + |a_1| + |a_0|$$

Esta notación es completamente independiente del sistema en que pretenda resolverse el problema puesto que permite ver cómo afectan los cambios en el tamaño del problema, los requerimientos de tiempo y espacio sin necesidad de conocer el tiempo requerido para realizar una instrucción o el número de bits usados para representar diferentes variables, que sólo afectarán como factores constantes los cambios en los requerimientos.

Además, como diferentes implementaciones de un mismo algoritmo sólo varían en una constante multiplicativa, si el tiempo de ejecución de una implementación de algún algoritmo es de complejidad $g(n)$, el tiempo empleado por cualquier otra implementación también será de complejidad $g(n)$.

Se dice que un problema se encuentra en la clase TIME($f(n)$) si existe un algoritmo que pueda resolverlo en cualquier variante de la máquina de Turing en un tiempo $O(f(n))$, donde n es el tamaño del problema.

En particular, se dice que un problema se puede resolver en tiempo polinomial si está en TIME(n^k) para algún valor finito de k, es decir, para que un problema esté en TIME(n^k) debe existir un algoritmo que lo resuelva

en cualquier variante de la máquina de Turing en un tiempo $O(n^k)$, lo cual implica que para valores del problema mayores a una constante n_0, el tiempo requerido para solucionar dicho problema es menor o igual que cn^k, donde c es una constante.

Todos los problemas que están en TIME(n^k) para algún valor de k se incluyen en la clase de complejidad P, donde P se refiere a polinomial [31]. Así, los problemas que pueden ser resueltos en tiempo polinomial además de considerarse fáciles, tratables o factibles, son colocados en la clase de complejidad P y también se conocen como problemas P.

En el caso de la clase de complejidad NP, el nombre NP viene de *non-deterministic polynomial time* o tiempo polinomial no determinista porque esta clase agrupa a todos aquellos problemas para los que, al menos en principio, existe un algoritmo no determinista eficiente para resolver el problema. Esto se debe a que una vez que se tienen una solución potencial para ellos, la exactitud de dicha solución puede ser verificada eficientemente, es decir, en tiempo polinomial, por lo que el problema podría ser resuelto eficientemente si se encontrara la solución correcta por medios aleatorios. Sin embargo, los problemas NP requieren una cantidad de tiempo exponencial si se resuelven en una máquina determinista de Turing ya que tienen una cantidad exponencial de soluciones potenciales y en el peor caso habría que probarlas todas para establecer que el problema no tiene solución o que su solución es la última que se probó.

Dicho de otra forma, la clase NP consiste en todos los problemas que pueden ser resueltos en tiempo polinomial en una máquina no determinista de Turing, que adivine la solución al problema. Por tanto, los problemas P se encuentran dentro de los problemas NP dado que para los problemas P la solución es encontrada y verificada en tiempo polinomial en una máquina determinista de Turing, lo cual implica que pueden resolverse en tiempo polinomial en una máquina no determinista de Turing, pues la primera solución que se encuentre aleatoriamente será de hecho la correcta [31]. Un problema se encuentra dentro de la clase NP si puede ser formulado como un problema de búsqueda.

El problema que aún no se resuelve es si P$\neq$NP, es decir, si existen problemas que se encuentren en la clase NP y no en la clase P. Este problema fue establecido en 1971 y la mayoría de los científicos de la computación cree que así es, pero nadie ha probado todavía que los problemas en la clase NP requieran un tiempo super-polinomial. De hecho, los límites inferiores probados para la mayoría de los problemas en esta clase son $O(n)$, mientras

que sus límites superiores son exponenciales, por lo que no se puede establecer si son problemas tratables o intratables. Para que sean clasificados como problemas tratables es necesario que se demuestre que tienen un límite superior polinomial y para que sean clasificados como intratables es necesario que se demuestre que tienen un límite inferior exponencial [18].

Dentro de los problemas NP se encuentran los problemas NP completos, que son los problemas más difíciles de resolver dentro de esta clase, en el sentido de que si se puede resolver un problema NP completo en un tiempo $t(n)$, entonces cualquier otro problema en NP puede ser resuelto en tiempo $O((t(n))^k)$, lo cual implica que para valores mayores del problema a una constante n_0, el tiempo requerido para solucionar dicho problema es menor o igual que $c(t(n))^k$, donde c es una constante [26].

Es decir, si el tiempo de solución $t(n)$ de cualquier problema NP completo es polinomial, esto es $t(n)$ está en $O(n^r)$ para algún valor de r ($t(n) \leq c_1 n^r$ para $n > n1$), entonces el tiempo máximo de solución para dicho problema es $c_1 n^r$ y cualquier otro problema en NP puede ser resuelto en tiempo $O((c_1 n^r)^k)$ u $O({c_1}^k n^{rk})$, que como fue establecido antes se expresa generalmente como $O(n^{rk})$, por lo que también es un tiempo polinomial.

Por tanto, si se encuentra un algoritmo polinomial para cualquier problema NP completo, se tendría un algoritmo polinomial para todos los problemas NP, mientras que si se encuentra un límite inferior exponencial para el tiempo de solución de cualquier problema NP completo, se habría probado que ningún problema NP completo puede resolverse en tiempo polinomial [18].

La siguiente clase en la jerarquía de complejidad es PSPACE, que agrupa a los problemas que pueden ser resueltos en un espacio polinomial, pero no necesariamente en tiempo polinomial. PSPACE incluye a los problemas NP, ya que si un problema de tamaño n es NP, sus posibles soluciones tienen un tamaño máximo polinomial $p(n)$ y para determinar si el problema tiene o no solución es necesario probar secuencialmente las $2^{p(n)}$ soluciones potenciales; cada prueba puede ser realizada en tiempo polinomial por hipótesis y por tanto en espacio polinomial, de manera que si se borran los resultados intermedios entre cada prueba, se pueden verificar todas las soluciones usando espacio polinomial [31].

Sin embargo, no se ha probado que las clases sean diferentes, ni siquiera se ha probado que PSPACE sea diferente a P, es decir, que existan problemas en PSPACE que no se encuentren en P.

La última clase es la EXPTIME que engloba los problemas que se pueden

resolver en tiempo exponencial en una máquina de Turing, es decir, los problemas que se pueden resolver en un tiempo $O(t^{f(n)})$ donde t es una constante mayor que 1 y $f(n)$ es una función polinomial de n. Los problemas de esta clase corresponden a los problemas intratables. Se ha probado que estos problemas no se pueden resolver en tiempo polinomial determinista y se ha mostrado que P≠EXPTIME, es decir, que P es un subconjunto estricto de EXPTIME [31]. En resumen, se tiene la siguiente relación entre las principales clases de complejidad:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$$

y dado que P≠EXPTIME, al menos alguna de las inclusiones debe ser estricta, sin que se haya probado que ninguna de ellas en particular lo sea.

A continuación se verán las posibilidades ofrecidas por la computación en paralelo, y en particular por las computadoras de ADN, para resolver problemas que pertenecen a una de estas clases, los problemas NP.

2.4. Clase NP y computación en paralelo

Como se ha mencionado previamente, un cambio en el modelo de computación implica cambios en los recursos utilizados para resolver los problemas computacionales.

En el caso de las computadoras secuenciales, las instrucciones son procesadas una a la vez en una unidad central de proceso. En cambio, las computadoras en paralelo pueden procesar más de una instrucción a la vez produciendo un ahorro en tiempo. Sin embargo, las computadoras en paralelo pueden ser simuladas en una máquina de Turing con recursos físicos totales (tiempo y espacio) diferentes a lo más en cantidades polinomiales.

Es decir, la disminución en el tiempo de procesamiento de las computadoras en paralelo sobre las computadoras secuenciales se compensa con el aumento en los recursos espaciales requeridos para llevar a cabo la computación, por lo que no hay un cambio esencial en el poder de las computadoras en paralelo sobre las computadoras secuenciales.

Las computadoras de ADN son un caso especial de computación en paralelo. Su ventaja sobre las computadoras en paralelo digitales es su tamaño. Así, dado el pequeño tamaño de las cadenas de ADN con relación a las computadoras digitales, aunque la cantidad de recursos espaciales, en este

caso de cadenas de ADN, aumente al disminuir el tiempo de solución de los problemas, este aumento resulta aceptable en términos prácticos al menos hasta cierto tamaño del problema en cuestión, cosa que no ocurre con las computadoras digitales donde el mencionado límite para el tamaño del problema se alcanzaría más rápidamente.

De esta forma, para algunos problemas NP completos se han planteado algoritmos de solución en computadoras de ADN que reducen su tiempo de exponencial a polinomial gracias al paralelismo logrado con la capacidad de las computadoras de ADN de operar sobre varias moléculas al mismo tiempo.

En el siguiente capítulo se expondrán las soluciones para dos problemas NP completos en computadoras de ADN: el problema del camino hamiltoniano y el problema SAT.

Capítulo 3

Resolviendo problemas NP con computadoras de ADN

> *Manipular cosas átomo por átomo ... es algo que en principio se puede hacer, pero en la práctica no se ha hecho porque somos demasiado grandes.*
>
> Richard Feynman

En este capítulo se presentarán dos problemas NP y sus soluciones en computadoras de ADN: el problema del agente viajero, resuelto por Adleman a través de la primera computadora de ADN construida y el problema de la *satisfactibilidad*, cuya solución fue propuesta por Lipton.

3.1. La computadora de Adleman

El desarrollo práctico de las computadoras de ADN inició en 1994 con la construcción de la primera computadora de ADN por Leonard M. Adleman en la Universidad del Sur de California. Su hallazgo fue publicado en la revista *Science* en noviembre de ese año en un artículo titulado *Molecular computation of solutions to combinatorial problem* [1].

3.1.1. El problema del camino hamiltoniano

La computadora de Adleman resuelve una variante del problema del agente viajero: el problema del camino hamiltoniano dirigido o HP (*Hamiltonian*

Path). Un camino hamiltoniano es una secuencia de vértices y líneas de una gráfica que inicia en un vértice dado (vértice inicial) y termina en otro (vértice final) y que además visita cada vértice en la gráfica exactamente una vez [16]. En la figura 3.1, por ejemplo, hay dos caminos hamiltonianos entre el vértice A y el vértice C: A→B→D→C y A→D→B→C.

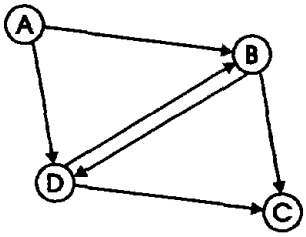

Figura 3.1: Gráfica con dos caminos hamiltonianos

El problema del camino hamiltoniano dirigido consiste en determinar si una gráfica dirigida dada contiene un camino hamiltoniano o no y es una variante del problema del agente viajero, que consiste en encontrar el camino de menor tiempo, distancia o costo para visitar una serie de ciudades partiendo y finalizando el viaje en la misma ciudad.

Este problema tiene aplicaciones en problemas de ruteo [3], aunque no se conoce ningún algoritmo eficiente (de tiempo polinomial) para resolverlo en computadoras electrónicas. Sin embargo, la exactitud de cualquier solución potencial puede ser verificada en tiempo polinomial, por lo que el problema es un problema NP y se puede resolver mediante búsqueda exhaustiva. Esta búsqueda termina una vez que se ha encontrado algún camino o que se han analizado todas las posibles soluciones. Además, ha sido probado que se encuentra en la clase de los problemas NP-completos, por lo que a partir de cierto tamaño del problema, tamaño que depende de las características de la computadora donde se pretenda solucionarlo, se requieren cantidades imprácticas de tiempo de cómputo para resolverlo.

Adleman resolvió una instancia del problema del camino hamiltoniano dirigido para 7 vértices que, como se observa en la gráfica de la figura 3.2, a la que se hará referencia como la gráfica G, sólo tiene un camino hamiltoniano con vértice inicial 0 y vértice final 6 (0→1→2→3→4→5→6).

Aunque en este caso la solución puede obtenerse rápidamente al inspeccionar visualmente la gráfica, el método empleado podría implementarse para resolver problemas de mayor escala. Además, el desarrollo de la compu-

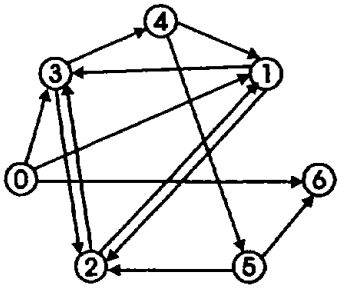

Figura 3.2: Gráfica del problema de Adleman

tadora de Adleman es importante porque demuestra la factibilidad de llevar a cabo computaciones a nivel molecular.

3.1.2. Algoritmo e implementación

Las entradas están conformadas por una gráfica G dirigida con n vértices, entre los cuales uno es designado como vértice inicial y otro como vértice final y la salida es un camino hamiltoniano, en caso de que exista al menos uno o la indicación de que no existe ninguno.

El algoritmo utilizado por Adleman para resolver el problema del camino hamiltoniano es el siguiente:

1. Generar caminos aleatorios a través de la gráfica.
2. Mantener solamente los caminos que comiencen con el vértice inicial y terminen con el vértice final.
3. Mantener solamente los caminos que tengan exactamente el número de vértices que tenga la gráfica.
4. Mantener sólo los caminos que visiten todos los vértices de la gráfica al menos una vez.
5. Si algún camino sobrevive, es la solución. Si no, la gráfica no tiene un camino hamiltoniano.

Este algoritmo fue implementado a nivel molecular en la computadora de Adleman de la siguiente forma.

1. *Generar caminos aleatorios a través de la gráfica.*

A cada vértice i $(0 \leq i \leq 6)$ en la gráfica se asoció una cadena simple de ADN con una secuencia aleatoria de 20 nucleótidos, denotada como s_i. Por ejemplo, Adleman utilizó, junto con las cadenas correspondientes a los vértices 0, 1, 5 y 6, las siguientes:

s_2 = 5'−TATCGGATCGGTATATCCGA
s_3 = 5'−GCTATTCGAGCTTAAAGCTA
s_4 = 5'−GGCTAGGTACCAGCATGCTT

Se considera que estas cadenas tienen la orientación 5' a 3', mientras que sus cadenas complementarias tienen la orientación 3'−5'. Así, si se define la función h como aquélla que mapea una cadena de ADN a su cadena complementaria, se tiene:

$h(s_2)$ = 3'−ATAGCCTAGCCATATAGGCT
$h(s_3)$ = 3'−CGATAAGCTCGAATTTCGAT
$h(s_4)$ = 3'−CCGATCCATGGTCGTACGAA

Cada cadena s_i se puede considerar formada por dos cadenas de longitud 10, llamadas s_i' y s_i", siendo s_i' la cadena formado por los 10 primeros nucleótidos y s_i" la cadena formada por los últimos 10 nucleótidos, lo cual se puede representar como $s_i = s_i's_i"$. Por ejemplo,

s_2'= 5'−TATCGGATCG
s_2"= 5'−GTATATCCGA

A continuación, cada arista existente en la gráfica del vértice i al vértice j se codificó con la cadena de 20 nucleótidos $e_{i\to j} = h(s_i" s_j') = h(s_i")h(s_j')$, es decir, se codificó como una cadena obtenida al ligar el complemento de la segunda mitad de la cadena que representa al vértice i con el complemento de la primera mitad de la cadena que representa al vértice j. Por ejemplo,

$e_{1\to 3} = h(s_2" s_3') = 3'$−CATATAGGCTCGATAAGCTC
$e_{3\to 2} = h(s_3" s_2') = 3'$−GAATTTCGATATAGCCTAGC

$$e_{3\to 4} = h(s_3'' s_4') = 3'\text{–GAATTTCGATCCGATCCATG}$$

Para cada vértice i y para cada arista de i a j se colocaron, aproximadamente, 3×10^{13} copias de cada molécula y se mezclaron en una reacción simple de ligado con T4 ADN ligasa en un buffer de ligasa y ddH2O para formar un volumen total de 100 μl, que fue incubado por 4 horas a temperatura ambiente.

Como las moléculas representantes de la arista de i a j ($e_{i\to j}$) eran complementarias a la segunda mitad de la molécula que representaba al vértice i (s_i) y a la primera mitad de la molécula que representaba al vértice j (s_j), al colocarlas juntas en la reacción de ligado, s_i y s_j se unieron a través de $e_{i\to j}$, lo mismo que s_j y s_k se unieron a través de $e_{j\to k}$. Por tanto, a través de la reacción de ligado se formaron moléculas dobles de ADN que codificaban los caminos aleatorios a través de la gráfica. Por ejemplo, las moléculas s_2 y s_3 quedaron unidas a través de la molécula $e_{2\to 3}$, como se observa en la figura 3.3.

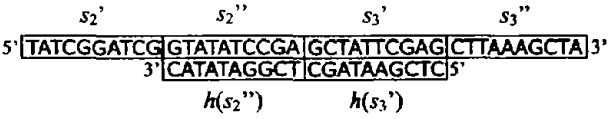

Figura 3.3: Molécula representante del camino 2→3

Éste es uno de los caminos aleatorios generados, pero como s_3 también podía unirse a s_4 a través de $e_{3\to 4}$, a la molécula anterior podía unirse s_4 a través de $e_{3\to 4}$, quedando el camino 2→3→4 codificado como se observa en la figura 3.4.

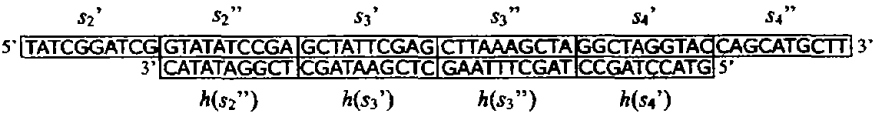

Figura 3.4: Molécula representante del camino 2→3→4

Y de esta forma se codificaron los caminos aleatorios a través de la gráfica. Como las cantidades de moléculas presentes son iguales para todos los vértices y aristas, la probabilidad p de que se forme cualquiera de los caminos de longitud 1 es la misma y es la probabilidad de que dos moléculas

que representan a los vértices se unan mediante la molécula que representa a la arista que los conecta.

Como además el número de moléculas es muy grande (3×10^{13} copias de cada vértice y cada arista), la probabilidad de que se formen caminos de longitud 2 es p^2, y en general la probabilidad de que se forman caminos de tamaño n es p^n. Por tanto, la probabilidad de que se formen caminos de tamaño n disminuye conforme n aumenta. Por ello, para verificar que la cantidad de ADN utilizada fue suficiente, sólo se calculará el número de caminos de tamaño menor o igual que 6, ya que además ésta es la longitud del camino hamiltoniano que nos interesa saber si existe o no.

Además, puesto que la formación de los enlaces de hidrógeno y fosfodiester que mantendrán unidas las cadenas complementarias implicarán un periodo de tiempo, se forman primero los caminos de menor tamaño. Además si se conoce el tiempo que dura la formación de cada enlace, podría limitarse el tiempo de la reacción para permitir sólo la formación de caminos de tamaño menor o igual que 6.

Como los elementos ${x_{ij}}^n$ de las potencias sucesivas de la matriz de adyacencia de la gráfica G, $\mathbf{X}(G)$, representan el número de paseos posibles de longitud n del vértice i al vértice j [17], para calcular el número de caminos de longitud i se suman todos los elementos de las gráficas $[\mathbf{X}(G)]^i$.

La matriz de adyacencia para la gráfica G es:

$$\mathbf{X}(G) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Por tanto, hay 13 caminos de longitud 1 en la gráfica.

Para $n = 2$,

$$[\mathbf{X}(G)]^2 = \begin{pmatrix} 0 & 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Hay 22 caminos de longitud 2.

Para $n = 3$,

$$[\mathbf{X}(G)]^3 = \begin{pmatrix} 0 & 3 & 1 & 2 & 1 & 1 & 0 \\ 0 & 2 & 2 & 2 & 1 & 1 & 0 \\ 0 & 3 & 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 4 & 2 & 1 & 0 & 1 \\ 0 & 2 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Hay 42 caminos de longitud 3.

Para $n = 4$,

$$[\mathbf{X}(G)]^4 = \begin{pmatrix} 0 & 2 & 6 & 4 & 2 & 1 & 1 \\ 0 & 3 & 5 & 4 & 2 & 1 & 1 \\ 0 & 2 & 6 & 4 & 2 & 1 & 1 \\ 0 & 5 & 2 & 4 & 2 & 1 & 0 \\ 0 & 2 & 4 & 3 & 2 & 1 & 0 \\ 0 & 3 & 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Hay 82 caminos de longitud 4.

Para $n = 5$,

$$[\mathbf{X}(G)]^5 = \begin{pmatrix} 0 & 8 & 7 & 8 & 4 & 2 & 1 \\ 0 & 7 & 8 & 8 & 4 & 2 & 1 \\ 0 & 8 & 7 & 8 & 4 & 2 & 1 \\ 0 & 4 & 10 & 7 & 4 & 2 & 1 \\ 0 & 6 & 6 & 6 & 3 & 2 & 1 \\ 0 & 2 & 6 & 4 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Hay 158 caminos de longitud 5.

Para $n = 6$,

$$[\mathbf{X}(G)]^6 = \begin{pmatrix} 0 & 11 & 18 & 15 & 8 & 4 & 2 \\ 0 & 12 & 17 & 15 & 8 & 4 & 2 \\ 0 & 11 & 18 & 15 & 8 & 4 & 2 \\ 0 & 14 & 13 & 14 & 7 & 4 & 2 \\ 0 & 9 & 14 & 12 & 6 & 3 & 2 \\ 0 & 8 & 7 & 8 & 4 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Hay 303 caminos de longitud 6.

Para cada camino de longitud n se requieren $2n + 1$ moléculas (n para codificar las líneas y $n + 1$ para codificar los vértices). En la tabla 3.1 se muestra el número de moléculas requeridas para las diferentes longitudes de camino.

Longitud (n)	Caminos (l)	Moléculas por camino ($2n + 1$)	Total de moléculas $l \times (2n + 1)$
1	13	3	39
2	22	5	110
3	42	7	294
4	82	9	738
5	158	11	1738
6	303	13	3939

Tabla 3.1: Cantidad de moléculas requeridas para diferentes longitudes de caminos

Por tanto, en total se requieren $39 + 110 + 294 + 738 + 1,738 + 3,939 = 6,858$ moléculas para codificar todos los caminos de longitud menor o igual a 6 que existen en la gráfica. Como estas moléculas tienen una secuencia diferente de nucleótidos dependiendo del camino que estén codificando, la cantidad de moléculas de tamaño 20 necesarias para cada vértice y para cada arista es menor que 6,858, pero tanto para los vértices como para las aristas se colocaron aproximadamente 3×10^{13} copias, por lo que la probabilidad de que se formaran varias copias de todos los caminos es alta, aunque, en teoría, habría sido suficiente con que se formara una molécula codificando el único camino hamiltoniano de la gráfica porque en los pasos siguientes sólo es necesario que exista al menos una copia de dicha molécula.

2. *Mantener solamente los caminos que comiencen con el vértice inicial y terminen con el vértice final.*

Para implementar este paso se amplificaron todas las moléculas que comenzaran con la molécula s_0 y que terminaran con la molécula s_6, aplicando la técnica de PCR con los *primers* como $h(s_0)$ y s_6. Se colocaron aproximadamente 6×10^{13} copias de cada primer, Taq ADN polimerasa y se procesaron en 35 ciclos de 15 segundos a 94° C, seguidos de 60 segundos a 30° C y 25 ciclos de 15 segundos a 94° C, seguidos de 60 segundos a 40° C.

Como en total se aplicaron 60 ciclos, se obtuvieron $2^{60} \approx 10^{18}$ copias de cada una de las moléculas que comenzaban con s_0 y terminaban con s_6.

Por otra parte, como para cada vértice y para cada línea se colocaron 3×10^{13} copias, el máximo de moléculas era el total de secuencias de 20 nucleótidos: 6×10^{14} (2.1×10^{14} moléculas para los vértices y 3.9×10^{14} para las aristas). Se ignoran las 60 cadenas simples para cada uno de los 61 caminos que empezara en s_0 o terminara en s_6, pero no cumplieran con ambas condiciones y que se obtienen como resultado de la aplicación de la técnica de PCR porque para cantidades de orden 10^{14}, 60×61 moléculas no son significativas.

Por tanto, el número de copias obtenidas mediante PCR de las moléculas codificando los caminos que comiencen con el vértice inicial y terminen con el vértice final es muy grande comparado con el resto de las moléculas presente en la solución y el proceso prácticamente eliminó a estas últimas moléculas.

Para ver que la cantidad de *primers* era suficiente, se considerarán sólo los caminos de longitud igual a 6 porque son éstos los que interesan y porque el paso siguiente, que consiste en mantener sólo los caminos de longitud 6, bien podría implementarse antes que éste, lo cual tendría la ventaja de no amplificar las moléculas de menor o mayor tamaño del que nos interesa con la consecuente disminución de recursos necesarios.

En cambio, si se lleva a cabo el proceso en el orden sugerido por Adleman, en el siguiente paso se tendría que trabajar con todas las cadenas generadas en el paso 1 y además con todas las moléculas obtenidas de la amplificación en el paso 2, en el que pueden amplificarse moléculas de tamaño menor a 6 que empiecen en s_0 y terminen en s_6 aun cuando éstas no interesan para lograr el objetivo final, complicando el manejo de las moléculas por utilizarse una cantidad mayor de ellas, innecesariamente.

En la gráfica y a través de la potencia 6 de la matriz de adyacencia, $[\mathbf{X}(G)]^6$, se observa que existen dos caminos que van de 0 a 6 de longitud 6:

$0 \rightarrow 3 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$ y $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$.

En la figura 3.5 se muestra el primer camino, antes y después de la separación por desnaturalización. Posteriormente, al unirse $h(s_0)$ a una de las moléculas obtenidas por desnaturalización y s_6 a la otra se obtienen la cadenas que se muestran en la figura 3.6. Lo mismo ocurriría para el segundo camino $(0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6)$.

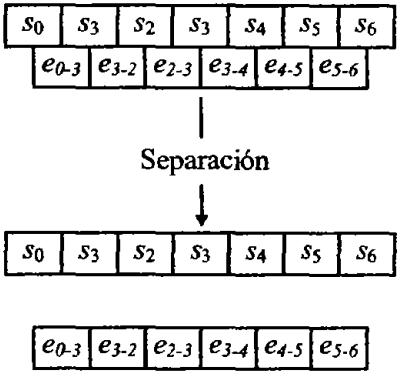

Figura 3.5: Desnaturalización en la computadora de Adleman

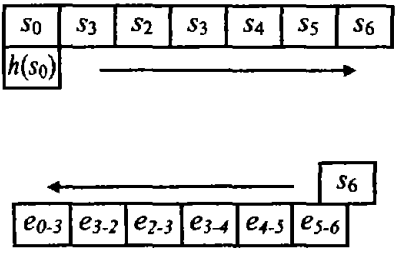

Figura 3.6: Amplificación de la molécula en la computadora de Adleman

También se consideran los caminos de longitud 6 que comienzan en s_0 o terminan en s_6, pero no ambos, porque para cada uno de ellos se requerirá un *primer* extra en cada ciclo, según lo explicado en la sección correspondiente al PCR. En $[\mathbf{X}(G)]^6$ se observa que hay $11 + 18 + 15 + 8 + 4 = 52$ caminos que empiezan en el vértice 0 pero no terminan en el vértice 6 y que hay $2 + 2 + 2 + 2 + 1 = 9$ caminos que terminan en el vértice 6 pero no empiezan en el vértice 0.

Suponiendo que existe la misma probabilidad de que se forme cualquiera

de los caminos que inician en el vértice 0 y que tienen longitud 6 (consideración razonable puesto que la cantidad de cada una de las moléculas que representaban a cada vértice y a cada arista era la misma y era muy grande) y como, además, en la solución las moléculas se consideran distribuidas uniformemente, los 6×10^{13} *primers* de $h(s_0)$ debieron distribuirse equitativamente entre los 54 caminos que iniciaban en s_0 de manera que, aproximadamente, $\frac{6\times10^{13}}{54} \approx 1.1\times10^{12}$ *primers* correspondieron a cada camino.

Esta cantidad es mayor al promedio de copias de cada uno de los caminos que inician en s_0 puesto que, para cada uno de los 54 caminos que empiezan en el vértice 0 (52 que no terminan en el s_6 y 2 que sí terminan en s_6), las 3×10^{13} copias de la molécula representante del vértice 0 debieron distribuirse uniformemente para formar esos 54 caminos, de manera que para cada uno habrían estado disponibles $\frac{3\times10^{13}}{54} \approx 5.6\times10^{11}$ copias del vértice 0, formándose aproximadamente este último número de copias de cada camino.

Así, al disponer de 1.1×10^{12} *primers* para 5.6×10^{11} copias de cada uno de los dos caminos que inician en en s_0 y terminan en s_6, la probabilidad de que los *primers* se unan a estas copias es muy alta. Como en el siguiente paso habrá aproximadamente el doble de cadenas con estas características, la probabilidad de que los *primers* de s_0 disponibles se unan a ellas y no a las otras que inician en s_0 es aún mayor.

En forma similar, para s_6, y bajo los mismos supuestos, los 6×10^{13} *primers* de s_6, debieron distribuirse de manera uniforme entre los 11 caminos de longitud 6 que terminan en el vértice 6 de manera que, en promedio, $\frac{6\times10^{13}}{11} \approx 5.4\times10^{12}$ *primers* estaban disponibles para cada uno de ellos mientras que para cada uno de los caminos que inician en s_0 y terminan en s6 habrían estado disponibles $\frac{3\times10^{13}}{11} \approx 2.7\times10^{12}$ copias del vértice 6, formándose entonces aproximadamente 2.7×10^{12} copias de cada camino. De nuevo, la probabilidad de que los *primers* se unan a estos caminos es alta.

En conclusión, es muy amplia la probabilidad de que al menos una de las moléculas de cada uno de los 2 caminos que comenzaban en el vértice 0 y terminaban en el vértice 6 fuera amplificada, aunque sigue existiendo la posibilidad de que no fuera así.

3. *Mantener solamente los caminos que tengan exactamente el número de vértices que tenga la gráfica.*

En el paso anterior, se observa que las moléculas amplificadas serán cadenas dobles y tendrán 7 moléculas de 20 nucleótidos cada una ó 10 nucléotidos menos al final de la cadena simple que codifica los caminos; estas cadenas

estarán en la misma proporción. Para implementar este paso, al resultado del paso 2 se le aplicó la técnica del gel de agarosa utilizando un buffer con bromuro de etidio y la banda de 140 pares de bases, (correspondiente a las cadenas de ADN que codifican los caminos con exactamente 7 vértices, es decir, sólo aquellas moléculas que se formaron al utilizar como *primer* $h(s_0)$). El gel se cortó recuperando las moléculas correspondientes y amplificando nuevamente las cadenas que iniciaran con s_0 y terminaran con s_6 para disminuir la posibilidad de que, por error, se tomaran caminos de longitud 6, pero que no tuvieran los extremos deseados.

4. *Mantener sólo los caminos que visiten todos los vértices de la gráfica al menos una vez.*

Para implementar este paso se utilizó a las moléculas resultantes del paso anterior. A éstas se les aplicó un sistema de búsqueda de moléculas. Para ello, primero se desnaturalizaron dichas moléculas para obtenerlas en forma simple, después se utilizó una técnica de búsqueda de moléculas teniendo como *sonda* s_1, de manera que se conservaran solamente los caminos que pasaran por el vértice 1 y este proceso fue repetido con s_2, s_3, s_4 y s_5.

Por tanto, el número de veces que se repita este paso depende de manera lineal del número de vértices, puesto que se debe repetir una vez para cada vértice, excepto para el primero y el último.

Como resultado, todos los caminos conservados codificarían a un camino hamiltoniano de la gráfica de manera que, tomando alguno o varios de ellos y leyendo su secuencia a través del método de Sanger, se conocería el camino buscado.

3.1.3. Resultados

El experimento de Adleman tomó aproximadamente 7 días de trabajo de laboratorio, pero un aumento en el tamaño de la gráfica aumentaría el número de veces que se repita el proceso de buscar moléculas que pasen por cada uno de los vértices de la gráfica (paso 4) y el número de vértices y líneas que deban codificarse (paso 1).

Para analizar estos cambios tanto en el tiempo como en la cantidad de ADN utilizado, se considerará una gráfica simple (aquélla que no contiene bucles ni líneas paralelas), completa (aquélla en la que cada par de vértices es adyacente), dirigida, de n vértices. En una gráfica con estas características,

el número de líneas es $n(n-1)$ porque de cada vértice sale una línea hacia cada uno de los $n-1$ vértices restantes.

Se elige una gráfica simple porque al buscar un camino hamiltoniano los bucles pueden ignorarse, puesto que se requiere que el camino pase solamente una vez por cada vértice. En cuanto a las líneas paralelas, si existen, éstas también pueden ignorarse puesto que lo único que importa es saber el orden en que deben recorrerse los vértices y no la línea por la que deben pasar en caso de que haya varias. Aun si se buscara resolver el problema del agente viajero, donde las líneas paralelas pueden tener asociadas diferentes necesidades de recursos, siempre puede elegirse la de menor tiempo, distancia o costo. Por otro lado, se elige una gráfica completa para tomar el enfoque de análisis de peor caso, siendo ésta la gráfica que requeriría codificar mayor cantidad de líneas.

Así, para los cambios en la cantidad de tiempo en función del tamaño de la gráfica, el tiempo necesario para llevar a cabo el paso 4 será una función de $n-2$, puesto que es éste el número de vértices que deben buscarse en los caminos conservados. Si se elige este paso como la operación básica en este problema por ser ésta la que mayor influencia tiene sobre el crecimiento del tiempo, el tiempo para resolver este problema mediante computadoras de ADN es una función de $n-2$, es decir, el tiempo de solución de este problema es lineal o de complejidad $O(n)$, a diferencia del tiempo requerido en una computadora convencional, que como se ha mencionado es no polinomial.

En cuanto al crecimiento de la cantidad de moléculas de cada tipo utilizadas, para cada camino de longitud i se requieren $2i+1$ moléculas (i para codificar las líneas e $i+1$ para codificar los vértices), por lo que para calcular el número de moléculas requeridas para codificar todos los caminos de tamaño menor o igual a $n-1$[1] se calcula

$$\sum_{i=1}^{n-1}(2i+1)m_i \tag{3.1}$$

donde m_i es el número de caminos de tamaño i.

Ignorando los caminos que incluyan ciclos de cualquier longitud (en la gráfica G se ignoraría, por ejemplo, el camino 1→2→1→3 por contener el ciclo 1→2→1), para $i=1$, $m_1 = n(n-1)$ ya que en cualquier gráfica el

[1]Sólo se consideran los caminos de longitud menor o igual a $n-1$ porque un camino hamiltoniano tiene tamaño $n-1$ y, como ya se ha mencionado, se puede limitar el tiempo del proceso para que no se formen caminos de mayor tamaño.

número de caminos de tamaño 1 es el número de líneas de la gráfica. Para $i = 2$, $m_2 = n(n-1)(n-2)$ puesto que cualquiera de los $n(n-1)$ caminos de longitud 1 podría continuar hacia cualquiera de los $n-2$ vértices que no ha alcanzado. En general, para $i = k$, $m_k = n(n-1)\ldots(n-k)$. Por tanto, el número de moléculas requeridas para codificar todos los caminos de tamaño menor o igual que $n-1$ es:

$$\sum_{i=1}^{n-1}(2i+1)m_i = \sum_{i=1}^{n-1}(2i+1)\frac{n!}{(n-i-1)!} \tag{3.2}$$

$$= 3n(n-1) + 5n(n-1)(n-2) + \ldots + (2(n-2)+1)n! + (2(n-1)+1)n! \tag{3.3}$$

Es decir, el número de moléculas requeridas es mayor que $n!$ y esta función crece más rápido que 2^n puesto que:

$$\lim_{n\to\infty}\frac{2^n}{n!} = \lim_{n\to\infty}\frac{2}{n}\cdot\frac{2}{n-1}\cdot\ldots\cdot\frac{2}{n-(n-2)}\cdot\frac{2}{n-(n-1))} = 0$$

Además, en estos cálculos se ha ignorado el aumento en el número de moléculas debido a la necesidad de utilizar un mayor número de *primers* en el paso 2 si se desea conservar la proporción de estos con relación al total de caminos que comiencen en el vértice inicial o terminen en el vértice final.

Por tanto, el tiempo requerido es de complejidad polinomial, pero el crecimiento en el número de moléculas en función del número de vértices de la gráfica es exponencial.

3.2. La propuesta de Lipton

En esta sección se presenta la solución propuesta por Richard J. Lipton en 1995 para resolver otro problema NP-completo, el problema SAT, utilizando computadoras de ADN [22].

3.2.1. El problema SAT

En lógica las proposiciones se clasifican en atómicas y moleculares. Las proposiciones atómicas son afirmaciones simples y se simbolizan mediante variables lógicas que pueden tomar el valor 0 ó 1, mientras que las proposiciones moleculares se forman combinando las fórmulas atómicas mediante los

conectores lógicos para negación, $\sim$, disyunción, $\vee$, y conjunción, $\wedge$, entre otros. En este trabajo se utilizarán indistintamente fórmula lógica o proposición para referirse a proposición molecular y forma atómica para referirse a proposición atómica.

Una fórmula lógica es satisfactible o consistente si resulta en un valor verdadero para al menos una de las combinaciones de valores de verdad para las variables que la forman. Por ejemplo, la fórmula $F_1 = \sim (\sim x_1 \vee x_2) \vee x_3$ es satisfactible porque, como se observa en su tabla de verdad (tabla 3.2) hay al menos una combinación de valores de verdad para x_1, x_2 y x_3 que hace que la fórmula sea verdadera (las combinaciones 1, 3, 4, 5 y 7 hacen que la expresión sea verdadera).

Combinación	x_1	x_2	x_3	$\sim x_1$	$\sim x_1 \vee x2$	$\sim (\sim x_1 \vee x_2)$	F_1
1	1	1	1	0	1	0	1
2	1	1	0	0	1	0	0
3	1	0	1	0	0	1	1
4	1	0	0	0	0	1	1
5	0	1	1	1	1	0	1
6	0	1	0	1	1	0	0
7	0	0	1	1	1	0	1
8	0	0	0	1	1	0	0

Tabla 3.2: Tabla de verdad para una proposición satisfactible

En cambio, la fórmula $F_2 = (x_1 \vee \sim x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\sim x_1 \vee x_3) \wedge \sim x_3$ no es satisfactible porque ninguna de las combinaciones de valores de verdad para sus variables resulta en un valor verdadero, tal como se observa en la tabla 3.3.

x_1	x_2	x_3	$x_1 \vee \sim x_2 \sim x_3$	$x_2 \vee x_3$	$\sim x_1 \vee x_3$	F_2
1	1	1	0	1	1	0
1	1	0	1	1	0	0
1	0	1	1	1	1	0
1	0	0	1	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	1	0
0	0	1	1	1	1	0
0	0	0	1	0	1	0

Tabla 3.3: Tabla de verdad para una proposición no satisfactible

El problema SAT o de la *satisfactibilidad* consiste en determinar si una fórmula lógica es satisfactible y tiene aplicaciones en la demostración automatizada de teoremas [3]. El mejor método conocido para resolver dicho problema consiste en probar las 2^n combinaciones de valores de verdad para una fórmula de n variables, por lo que su tiempo de solución es de complejidad 2^n.

3.2.2. Algoritmo e implementación

La entrada está constituida por una fórmula lógica de n variables en forma normal conjuntiva. Una fórmula, F, está en su forma normal conjuntiva si y sólo si F tiene la forma $F = F_1 \wedge F_2 \wedge \ldots \wedge F_i \wedge \ldots \wedge F_m$, donde cada F_i, con i entre 1 y m, se conoce como cláusula y es una disyunción de l_i proposiciones, es decir, es de la forma $y_1 \vee y_2 \vee \ldots \vee y_{l_i}$, donde cada una de las y_j es una de las n variables consideradas en la fórmula, o su negación.

Para llevar cualquier fórmula lógica a su forma normal conjuntiva se aplican las equivalencias entre diferentes expresiones lógicas para eliminar condicionales, incluir las negaciones dentro de paréntesis y aplicar las reglas distributivas. Se considera que en una misma cláusula sólo aparece x_i ó $\sim x_i$ porque si aparecieran ambas, la cláusula sería verdadera automáticamente y podría ignorarse. De esta forma l_i siempre es menor o igual que n.

La salida es la combinación de valores de verdad que hacen que la fórmula de la entrada sea satisfactible o la indicación de que ninguna combinación de valores de verdad la satisface.

El algoritmo que Lipton propone implementar mediante computadoras de ADN para determinar si una fórmula lógica de este estilo es satisfactible es un algoritmo de búsqueda exhaustiva (probar todas las combinaciones de valores de verdad para ver si alguna satisface la fórmula de entrada) y se puede formular como sigue:

1. Formar el conjunto 0 con las 2^n combinaciones de valores de verdad para las n variables implicadas.
2. Desde $i = 1$ hasta m, repetir los pasos a, b y c:
   - *a*) Formar el conjunto 0^n con los elementos del conjunto $i - 1$.
   - *b*) Desde $j = 1$ hasta l_i repetir los pasos I y II:
     - I) Formar el conjunto j' con los elementos de $(j-1)^n$ en los que la variable que se encuentra en la posición j en la cláusula

i tenga el valor 1 si dicha variable se encuentra en su forma atómica o el valor 0 si se encuentra su negación.

II) Formar el conjunto j" con los elementos de $(j-1)$" que no hayan formado parte del conjunto j'.

c) Formar el conjunto i con

$$\bigcup_{j=1}^{l_m} j'.$$

Con el paso 1 se garantiza que todas las combinaciones de valores de verdad serán consideradas (por ello, el algoritmo se considera de búsqueda exhaustiva), mientras que en el paso 2, se analiza cada una de las m cláusulas que constituyen la fórmula para formar el conjunto i, donde i es el número de cláusula que se está analizando, el cual contendrá las combinaciones de valores de verdad que satisfagan la cláusula en cuestión y todas las anteriores.

A continuación se explica detalladamente la manera en que se desarrolla este segundo paso. En el inciso *a*) se forma el conjunto 0" con las combinaciones del conjunto $i-1$ para buscar las combinaciones de valores de verdad que satisfagan la cláusula i solamente entre las que ya satisfacen la cláusula $i-1$. Esto es con objeto de evitar buscar entre las combinaciones que no satisfacen las cláusulas anteriores a la cláusula i, puesto que para que se satisfaga la fórmula completa es necesario que se satisfagan todas las cláusulas que la componen ya que están unidas a través de conjunciones y una conjunción sólo es verdadera cuando todas las proposiciones que la forman son verdaderas.

En el inciso *b*) se analizan, para la cláusula actual, cada una de las l_i proposiciones que la componen, para conservar en el conjunto i solamente las combinaciones de valores de verdad que satisfagan una o más de las l_i proposiciones, de manera que contenga a todas las combinaciones que satisfacen la cláusula i. En este caso es suficiente que se satisfaga una de las proposiciones para que toda la cláusula sea consistente, puesto que las proposiciones están unidas a través de disyunciones.

En el paso I, se forma el conjunto j', donde j se refiere a la posición de la proposición en la cláusula i, con las combinaciones de valores de verdad que satisfagan la proposición j. Si la proposición j se encuentra en su forma atómica, será verdadera cuando la variable asuma el valor 1 y si se encuentra su negación, será verdadera cuando la variable tome el valor 0, por ello en

el primer caso se conservan las combinaciones con el valor 1 para la variable en cuestión y en el segundo se conservan las combinaciones con el valor 0.

Luego, en el paso II se forma el conjunto j'' con los elementos que no satisfacen la proposición j. Éste será el conjunto sobre el que se busquen las combinaciones que satisfagan la proposición siguiente puesto que, como se ha mencionado, es suficiente que la combinación satisfaga alguna de las proposiciones de la cláusula para que toda la cláusula sea verdadera.

Finalmente, en el inciso c) se forma el conjunto i con la unión de todas las combinaciones que hayan satisfecho una o más proposiciones de la cláusula j. Por tanto, en el conjunto m deberán encontrarse todas las combinaciones que satisfagan la fórmula completa y si, al final, el conjunto m contiene algún elemento, dicho elemento es la solución del problema SAT. Si no, la fórmula no es satisfactible.

Como ejemplo de la aplicación del algoritmo se considera la siguiente fórmula lógica:

$$(\sim x_1 \vee \sim x_2 \vee \sim x_3) \wedge (x_1 \vee x_2 \vee \sim x_4)$$

$n = 4$ variables

$m = 2$ cláusulas

$l_1 = l_2 = 3$ variables en cada cláusula

1. 0= {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}[2]

2. $i = 1$

   a) 0''= {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

   b) $j = 1$

      I) 1'= {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111}

      II) 1''= {1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

      $j = 2$

      I) 2'= {1000, 1001, 1010, 1011}

      II) 2''= {1100, 1101, 1110, 1111}

      $j = 3$

      I) 3'= {1100, 1101}

[2]0111 significa x1=0, x2=1, x3=1 y x4=1.

II) $3^n = \{1110, 1111\}$

c) $1 = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101\}$

$i = 2$

a) $0^n = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101\}$

b) $j = 1$

I) $1' = \{1000, 1001, 1010, 1011, 1100, 1101\}$

II) $1^n = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111\}$

$j = 2$

I) $2' = \{0100, 0101, 0110, 0111\}$

II) $2^n = \{0000, 0001, 0010, 0011\}$

$j = 3$

I) $3' = \{0000, 0010\}$

II) $3^n = \{0001, 0011\}$

c) $2 = \{0000, 0010, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101\}$

Por tanto, la fórmula propuesta es satisfactible para cualquiera de estas combinaciones de valores de verdad.

La implementación propuesta por Lipton para este algoritmo es la siguiente:

1. *Formar el conjunto 0 con las 2^n combinaciones de valores de verdad para las n variables implicadas.*

La implementación de este paso se basa en la correspondencia de las 2^n combinaciones de valores de verdad con los caminos de la gráfica que se muestra en la figura 3.7.

En esta gráfica hay 2^n caminos que inician en v_0 y terminan en v_n puesto que en cada v_i desde $i = 0$ hasta $n - 1$ es necesario elegir entre la línea que conduce a a_{i+1}^0 y la que conduce a a_{i+1}^1; además, las elecciones para cada v_i son independientes entre sí. Así, cada camino tiene $2n+1$ vértices (los $n+1$ vértices $v_0, v_1, \ldots, v_{n-1}, v_n$ y n vértices resultantes de la elección entre a_{i+1}^0 y a_{i+1}^1 desde $i = 0$ hasta $n - 1$).

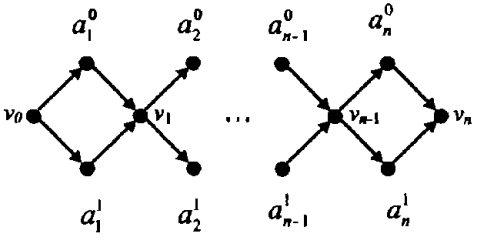

Figura 3.7: Gráfica para la propuesta de Lipton

Para una fórmula lógica con n variables, x_1, x_2, ..., x_n, las diferentes asignaciones de valores de verdad pueden hacerse corresponder con los diferentes caminos de la gráfica, de tal forma que el camino $v_0 \rightarrow a_1^{i_1} \rightarrow v_1 \rightarrow a_2^{i_2} \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n-1} \rightarrow a_n^{i_n} \rightarrow v_n$ corresponde a la asignación donde cada variable x_j toma el valor i_j dentro del conjunto $\{0,1\}$ para $j = 1, 2, \ldots n$.

Dicha gráfica se codifica exactamente igual que en el paso 1 del experimento de Adleman. Nuevamente se debe utilizar la misma cantidad de cadenas para cada vértice y para cada línea, de tal forma que como, además, la gráfica es simétrica[3], todos los caminos deberían formarse en la misma proporción. Además, si se coloca la cantidad de ADN suficiente, es de esperarse que existan cadenas repetidas que codifiquen el mismo camino o la misma combinación de valores de verdad.

A diferencia del experimento de Adleman, donde la gráfica es la entrada del problema y, como consecuencia, las cadenas en el tubo de ensayo inicial cambian también con la entrada, en la propuesta de Lipton para resolver el problema SAT la entrada no se codifica. El tubo de ensayo inicial contiene cadenas que codifican todas las posibles asignaciones de valores de verdad por lo que, si se mantiene fijo el número de variables, se podría resolver el problema para varias fórmulas lógicas al mismo tiempo, tomando varias copias de este tubo de ensayo.

Se considera que todas las cadenas de ADN en el tubo inicial son cadenas simples, lo cual puede lograrse mediante desnaturalización.

Para hacer corresponder la nomenclatura utilizada en el algoritmo, el tubo de ensayo que contenga la codificación de la gráfica será el tubo 0,

[3]Simétrica en la distribución de los vértices, no en el enfoque de Teoría de Gráficas, donde una gráfica dirigida es simétrica cuando para cada línea de un vértice i a un vértice j existe una línea del vértice j al vértice i.

haciendo referencia al conjunto 0.

2. *Desde* $i = 1$ *hasta* m, *repetir los pasos a, b y c.*

   a) *Formar el conjunto 0" con los elementos del conjunto* $i - 1$. Para implementar este paso es suficiente con nombrar 0" al tubo que contenga los caminos que codifiquen los elementos del conjunto $i - 1$.

   b) *Desde* $j = 1$ *hasta* l_i *repetir los pasos I y II.*

      I) *Formar el conjunto j'*. Para implementar este paso se puede utilizar la técnica de búsqueda de moléculas presentada en el capítulo 1, teniendo como molécula objetivo la cadena que codifique a la variable a_k^0 (si la variable está en su forma atómica) ó a_k^1 (si se encuentra su negación), donde k es el índice de la variable que se encuentra en la posición j en la cláusula i.

      II) *Formar el conjunto j"*. La implementación de este paso consiste simplemente en nombrar j" a la solución que contenga las cadenas que no se hayan separado mediante la técnica de búsqueda aplicada en el paso anterior.

   c) *Formar el conjunto i.* La implementación de este paso consiste en colocar en el mismo tubo de ensayo las cadenas encontradas con las l_i aplicaciones de la técnica de búsqueda del paso I del inciso b). Para ello, después de cada aplicación de la técnica de búsqueda, se pueden verter las moléculas encontradas en el tubo de ensayo que contendrá al conjunto i, de manera que cuando se hayan realizado las l_i aplicaciones de técnica de búsqueda, el conjunto i esté completo.

Finalmente, para encontrar la solución, se leerán la o las cadenas que se hayan obtenido al formar el conjunto m y se decodificará la solución. Dados los errores que podrían existir, será conveniente verificar que la combinación encontrada como solución realmente satisfaga la fórmula que se investiga.

En la implementación del inciso *b*) podría ocurrir que, por error, alguna de las cadenas que debía ir al conjunto j' vaya al conjunto j" y viceversa, pero como probablemente dicha cadena estará repetida varias veces, en la mayor parte de los casos debería ir al tubo correcto, de manera que aún podría obtenerse la solución correcta.

Además, para que esta implementación funcione correctamente, el tamaño de las cadenas que codifican a cada uno de los valores de verdad (en este caso, 20) debe garantizar que no se aparezca por accidente la secuencia de ninguna de estas cadenas a lo largo de ninguno de los caminos que codifican una asignación de valores de verdad.

En este caso, como la secuencia de las cadenas es generada de manera aleatoria, en cada una de las 20 posiciones de una cadena podría ir cualquiera de los cuatro nucleótidos con la misma probabilidad, $\frac{1}{4}$, y como la asignación de un nucleótido para alguna de las posiciones es independiente de la asignación del resto de los nucleótidos, la probabilidad de que una cadena se repita es muy baja $(\frac{1}{4})^{20} \approx 9 \times 10^{-13}$.

Por otra parte, como la gráfica en cuestión tiene $2n+1$ vértices, éste es el número de cadenas que deben unirse para formarla y la probabilidad de que al unirse estas cadenas se presente la misma secuencia de una ellas, la cadena i, en una posición que no le corresponde por accidente es la probabilidad de que para cualquiera de las primeras $2n$ cadenas, por ejemplo para la cadena j, sus últimos 1, 2, ..., 18 ó 19 nucleótidos sean los primeros 1, 2, ...,18 ó 19 de la cadena i y los primeros nucleótidos de la cadena $j+1$ sean los últimos nucleótidos de la cadena i.

La probabilidad de que los últimos r nucleótidos de la cadena j sean los primeros de la cadena i es $(\frac{1}{4})^r$ y la probabilidad de que los primeros $20-r$ nucleótidos de la cadena $j+1$ sean los últimos de la cadena i es $(\frac{1}{4})^{20-r}$; como estos dos eventos son independientes, la probabilidad de que ambos ocurran es $(\frac{1}{4})^{20}$ y la probabilidad de que ocurran en cualquiera de las primeras $2n$ cadenas es $2n(\frac{1}{4})^{20}$, por tanto ésta es la probabilidad de que la secuencia de alguna de las cadenas se repita en una posición que no le corresponde y la probabilidad de que esto mismo ocurra para cualquiera de las $2n+1$ cadenas codificadas es:

$$p(n) = \begin{cases} (2n)(2n+1)(\frac{1}{4})^{20} & \text{para } 1 < n < 524,287 \\ 1 & \text{para } n > 524,287 \end{cases}$$

Para $n < 16,579$, la probabilidad es menor a 0.001. Por tanto, para problemas con hasta 16,579 variables, el tamaño 20 para las cadenas es suficiente para mantener muy pequeña la probabilidad de que se repita alguna de las secuencias de las cadenas por error.

En este trabajo se pretende demostrar que la implementación propuesta por Lipton permite resolver el problema SAT en tiempo polinomial. En el

siguiente capítulo se hará este análisis y se mostrará la superioridad de las computadoras de ADN con relación a las computadoras electrónicas en cuanto al tiempo de solución de este problema, mostrando que las computadoras electrónicas requieren un tiempo exponencial para resolver el problema con una computadora en particular, aunque las computadoras de ADN tienen la desventaja de requerir una cantidad de ADN que crece de manera exponencial.

Capítulo 4

Complejidad del problema SAT

¿Cómo hacerte saber que siempre hay tiempo?
Que uno tiene que buscarlo y dárselo...

Mario Benedetti

Para demostrar la superioridad de las computadoras de ADN con relación a las computadoras clásicas se propondrá un modelo para calcular el tiempo de solución de un problema SAT de n variables en computadoras de ADN y en computadoras electrónicas, mostrando que el tiempo de solución en computadoras electrónicas es exponencial, mientras que el tiempo de solución en computadoras de ADN es polinomial, aunque la cantidad de moléculas de ADN que se requieren para resolverlo crece de manera exponencial.

4.1. Solución en computadoras electrónicas

Como el algoritmo para resolver este problema en computadoras de ADN es el de búsqueda exhaustiva, también será éste el que se considere en la solución del problema SAT en computadoras electrónicas.

Para resolver el problema SAT de n variables en una computadora electrónica se requiere comparar la evaluación de 2^n combinaciones de valores de verdad con el valor 1 en el peor caso, es decir, cuando ninguna de las $2^n - 1$ primeras combinaciones resulten en una evaluación 1, puesto que el

problema SAT se resuelve en cuanto se encuentra la primera combinación que satisfaga la fórmula o cuando se hayan probado todas las combinaciones y ninguna de ellas satisfaga la proposición.

Por tanto, eligiendo como operación básica la comparación de la evaluación de cada combinación de valores de verdad en la fórmula de interés con el valor lógico 1, el número máximo de operaciones básicas que el algoritmo efectúa con una entrada de tamaño n es 2^n, es decir, el tiempo de solución es una función de complejidad exponencial, $O(2^n)$. El tiempo absoluto depende del tiempo que tome probar cada combinación y éste, a su vez, depende de la velocidad de la computadora que se esté utilizando.

Una forma de medir la velocidad de una computadora es en flops, que significa operaciones de punto flotante por segundo. Algunos múltiplos de los flops son los gigaflops (Gflops) y los teraflops (Tflops); un gigaflop son mil millones de flops u operaciones de punto flotante por segundo, mientras que un teraflop es un billón de operaciones de punto flotante por segundo [20].

Para ejemplificar el tiempo absoluto requerido para resolver el problema SAT se tomará el caso de la supercomputadora clasificada hasta junio del 2004 como la de mayor velocidad en el planeta en la lista de las 500 supercomputadoras más rápidas presentada por la Universidad de Mannheim, la Universidad de Tennessee, y el National Energy Research Scientific Computing Center (NERSC) del Laboratorio Nacional Lawrence Berkeley.

Esta lista fue elaborada por primera vez en 1993 y se actualiza cada 6 meses. Está basada en el tiempo de ejecución de un programa de prueba que resuelve un sistema de ecuaciones matemáticas [33].

La supercomputadora mencionada se llama *Earth Simulator Center* y se encuentra en Japón. Fue desarrollada por la Agencia Nacional para el Desarrollo del Espacio de Japón (NASDA: National Space Development Agency of Japan), el Instituto Japonés para la Investigación de la Energía Atómica (JAERI: Japan Atomic Energy Research Institute) y el Centro de Ciencia y Tecnología Marina de Japón (JAMSTEC: Japan Marine Science and Technology Center). Su instalación terminó en febrero del 2002 y se utiliza principalmente para la simulación y predicción de la variaciones en las condiciones de la tierra y para la simulación de procesos industriales y científicos [13].

Está formada por 640 nodos procesadores que, a su vez, están formados por 8 procesadores aritméticos capaces de realizar 8 gigaflops de manera que, en teoría, la computadora *Earth Simulator* es capaz de realizar hasta 40,000

gigaflops (640×8×8= 40960), es decir 40 teraflops, aunque en la práctica realiza aproximadamente 36 teraflops [33].

En los cálculos posteriores se considerará que cada una de las comparaciones necesarias para resolver el problema SAT toma el mismo tiempo que realizar una operación de punto flotante, aunque las operaciones con números enteros se ejecutan en menos tiempo que las operaciones de punto flotante. Esta falta de precisión se justifica porque con estos cálculos solamente se pretende ejemplificar la forma en que crece el tiempo de solución del problema SAT en función de su número de variables y no establecer con precisión éste ya que, como se menciona en el capítulo 2, este enfoque tiene, entre otras, la desventaja de estar en función de la velocidad actual de la computadora analizada, característica que a su vez depende del desarrollo actual de la tecnología, por lo que varía constantemente.

Bajo estos supuestos, el tiempo necesario para resolver el problema SAT de n variables en la supercomputadora *Earth Simulator* es:

$$T_E(n) = \frac{2^n}{36 \times 10^{12}} \text{ segundos} \tag{4.1}$$

El tiempo calculado para algunos valores de n se resume en la tabla 4.1[1], mientras que en la figura 4.1, se observa la gráfica del tiempo de solución del problema SAT en función de su número de variables.

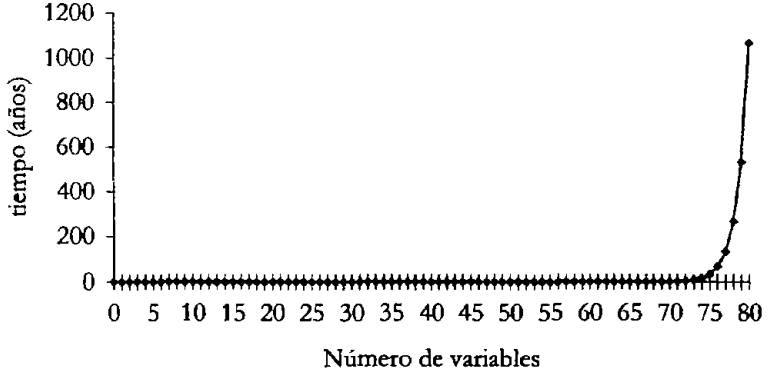

Figura 4.1: Gráfica del tiempo de solución en la *Earth Simulator*

[1]Para estos cálculos se considera 1 año =365 días y 1 mes = 30 días.

Variables (n)	Tiempo de solución
1	5.6×10^{-14} segundos
10	2.8×10^{-11} segundos
20	2.9×10^{-8} segundos
30	3×10^{-5} segundos
40	0.03 segundos
50	31.3 segundos
55	16 minutos
60	8 horas
65	11 días
70	1 año
71	2 años
72	4 años
73	8 años
74	16 años
75	33 años
76	66 años
77	133 años
80	1,064 años
85	34,075.37 años
90	1×10^{6} años
100	1×10^{9} años
200	1.4×10^{39} años

Tabla 4.1: Tiempo de solución del problema SAT en la *Earth Simulator*

Como se puede observar, la tasa de crecimiento del tiempo con relación a la tasa de crecimiento del número de variables es muy alta y la cantidad de tiempo necesaria para resolver el problema SAT hace que el problema sea irresoluble en la práctica para más de aproximadamente 70 variables, utilizando la supercomputadora especificada.

4.2. Solución en computadoras de ADN

En esta sección se presentará un modelo para calcular el tiempo de solución del problema SAT de n variables y m cláusulas en una computadora de ADN siguiendo el algoritmo propuesto por Lipton y se planteará un modelo para calcular el volumen de ADN mínimo que se requiere para resolver dicho problema.

4.2.1. Complejidad temporal

Para este análisis, de los pasos implicados en el Algoritmo de Lipton presentado en la sección 3.2.2 (ver página 58), se elige como operación básica la separación de los elementos que representan a las combinaciones en las que la j-ésima variable de la i-ésima claúsula tiene el valor 1 si dicha variable se encuentra en su forma atómica en la claúsula o el valor 0 si se encuentra su negación (paso I del del algoritmo).

Los otros pasos se descartan porque el paso 1 sólo se implementa una vez, mientras que para implementar los los pasos 2.*a*) y II del inciso 2.*b*) es suficiente darle un nombre a los tubos de ensayo que se indican en la misma sección y la implementación del paso 2.*c*) tiene una correspondencia con el paso I que permite incluirla en este último.

El paso elegido como operación básica implica aplicar una vez la técnica de búsqueda de moléculas para cada una de las l_i variables de cada cláusula i (con i entre 1 y m) pero, como ya se ha establecido, l_i siempre es menor o igual a n, por lo que este paso se repite a lo más mn veces.

Por su parte, cada implementación del paso del inciso 2.*c*). consiste en colocar en el mismo tubo de ensayo las cadenas encontradas con las l_i aplicaciones de la técnica de búsqueda del paso I, vertiendo las moléculas encontradas después de cada aplicación de la técnica de búsqueda en el tubo de ensayo que contendrá al conjunto i. Sin embargo, una vez que las moléculas son encontradas siempre se vierten en algún contenedor, por lo que el paso

2.c) solamente especifica el tubo al que se verterán las moléculas encontradas de manera que se puede considerar parte del paso I.

Por tanto, eligiendo como operación básica la extracción, como se llama a la separación de las cadenas de un tubo de ensayo en aquéllas que contienen una secuencia específica de nucleótidos y aquéllas que no la contienen [5], para resolver un problema SAT de m cláusulas y n variables, el número máximo de operaciones básicas que el algoritmo propuesto por Lipton efectúa es mn, es decir, el tiempo de solución es una función de complejidad cuadrática, $O(mn)$. En otras palabras, la complejidad del tiempo de solución del problema SAT se reduce de exponencial a polinomial utilizando computadoras de ADN en lugar de computadoras electrónicas.

El tiempo absoluto depende del tiempo que tome realizar cada una de las extracciones y éste, a su vez, depende de la habilidad de la persona o personas que las estén realizando, de la velocidad de las reacciones implicadas y de los métodos e instrumentos utilizados.

En este trabajo se considerará que cada extracción toma una hora, puesto que en su repetición del experimento de Adleman en 1995, Peter Kaplan, un científico de la Universidad de Princeton, encontró que cada extracción puede realizarse en aproximadamente 1 hora [5]. Este tiempo también ha sido establecido por Lönneborg, Sharma y Stougaard [32].

Por tanto, el tiempo necesario para resolver el problema SAT de n variables y m cláusulas en una computadora de ADN es:

$$T_A(n) = mn \text{ horas} \tag{4.2}$$

Para simplificar el modelo el tiempo de solución se presentará en función de r, donde r es el máximo entre m y n. Por tanto,

$$TA(r) = r^2 \text{ horas} \tag{4.3}$$

Para ejemplificar, se presentan algunos cálculos en la tabla 4.2[2] y la gráfica correspondiente en la figura 4.2.

Como puede observarse, aun para 95 variables el tiempo necesario para resolverlo hace factible encontrar la solución del problema utilizando aproximadamente la misma cantidad de tiempo que el requerido para resolver el problema SAT de 70 variables en la supercomputadora con mayor velocidad en el mundo.

[2]Para estos cálculos se considera 1 año =365 días y 1 mes = 30 días.

máx{número de cláusulas, número de variables}	Tiempo de solución
1	1 hora
10	4 días
20	17 días
30	1.2 meses
40	2.2 meses
50	3.5 meses
55	4.2 meses
60	5 meses
65	5.9 meses
70	6.8 meses
71	7 meses
72	7.2 meses
73	7.4 meses
74	7.6 meses
75	7.8 meses
76	8 meses
77	8.2 meses
80	8.9 meses
85	10.03 meses
90	11.2 meses
100	1.1 años
200	4.5 años

Tabla 4.2: Tiempo de solución en una computadora de ADN

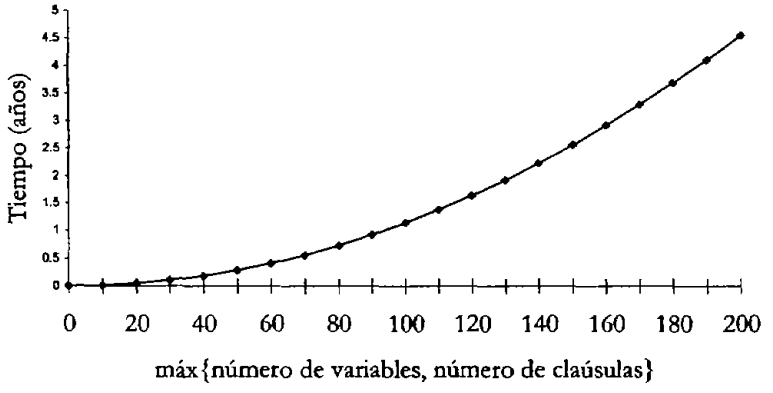

Figura 4.2: Gráfica del tiempo de solución en una computadora de ADN

Sin embargo, en la siguiente sección se observa que la cantidad de ADN requerida para resolver el problema hace que sea poco factible resolverlo después de cierto número de variables.

4.2.2. Complejidad espacial

De acuerdo con el algoritmo propuesto por Lipton, para resolver el problema SAT de n variables en una computadora de ADN se requiere codificar una gráfica cuyos caminos representan las combinaciones de valores de verdad de las variables. Esta gráfica tiene $3n + 1$ vértices y 4n líneas. Por tanto, para codificarla se requiere utilizar $7n + 1$ cadenas simples diferentes de tamaño 20.

Se pretende ejemplificar la cantidad de ADN necesaria para resolver el problema de hasta 200 variables, por lo que se requieren a lo más $7 \times 200 + 1 = 1401$ cadenas diferentes, pero con cadenas de tamaño 20 se pueden lograr $4^{20} = 1.0995 \times 10^{12}$ combinaciones, cantidad mucho mayor a la mínima requerida, por lo que utilizar cadenas de tamaño 20 es suficiente para codificar la gráfica.

Por otra parte, para poder resolver el problema, garantizando que la solución obtenida es la correcta, inicialmente, se requiere que se encuentren codificadas todas las posibles combinaciones de valores de verdad para las n variables (2^n), es decir, se requiere que se formen todos los posibles caminos

de la gráfica mencionada.

Por tanto, como mínimo se requiere una cantidad de ADN que al menos haga posible la formación de cada uno de ellos. Si se colocan en un tubo de ensayo las $7n + 1$ cadenas diferentes de ADN que representan los vértices y las líneas de la gráfica se formará solamente uno de los caminos de la gráfica. Al repetir el experimento 2^n veces existe la posibilidad de que se forme un camino diferente en cada uno de los 2^n ensayos y por tanto, de que se codifiquen todas las posibles combinaciones de valores de verdad para las n variables.

Por tanto, para resolver un problema SAT de n variables, se requieren como mínimo $2^n(7n + 1)$ cadenas de 20 nucleótidos cada una.

Como ya se mencionó, en el experimento de Adleman para resolver el problema del camino hamiltoniano se colocaron aproximadamente 3×10^{13} copias de cada molécula para cada vértice i y para cada artista de i a j que, en solución, ocuparon un volumen total de 100 μl. Es decir, $3\times10^{13}(7 + 13)$ moléculas de 20 nucleótidos cada una (un total de 1.2×10^{16} nucleótidos) ocuparon 100 μl. Por lo que 10^{20} nucleótidos ocuparían 833,333.33 μl = 833.33 ml.

Sin embargo, de acuerdo con los cálculos presentados en 1996 por Adleman, Rothemund, Roweis y Winfree, $2^{56} \times 17,360$ nucléotidos (2^{56} cadenas simples de 11,580 nucleótidos complementadas en 5,780 nucleótidos) ocupan, en solución, 140 ml [2]. Por tanto, 10^{20} nucleótidos ocuparían 11.19 ml. Se tomará este último dato y no el del primer experimento de Adleman por ser más reciente.

Por tanto, $2^n(7n + 1)$ cadenas simples de tamaño 20 de ADN ocuparían un volumen total de:

$$V_{2_n}(n) = \frac{2^n(7n + 1) \times 20 \times 140}{2^{56} \times 17360} \tag{4.4}$$

$$V_{2_n}(n) = \frac{2^n(7n + 1)}{2^{56} \times 6.2} \tag{4.5}$$

Es decir, el volumen que ocupa el ADN requerido para codificar 2^n combinaciones de valores de verdad es $O(n2^n)$; el volumen mínimo de ADN requerido para resolver el problema SAT crece en un factor exponencial.

En la tabla 4.3 se presentan algunos valores para este volumen mínimo y en la figura 4.3 se muestra una gráfica donde se observa el crecimiento de dicho volumen para hasta 80 variables.

Número de variables	Volumen mínimo
1	3.58×10^{-17} ml
10	1.63×10^{-13} ml
20	3.31×10^{-10} ml
30	5.07×10^{-7} ml
40	6.92×10^{-4} ml
50	0.88 ml
55	31.13 ml
60	1.09 l
65	37.66 l
70	1,297.51 l
71	2,632.01 l
72	5,338.01 l
73	10,824.01 l
74	21,943.99 l
75	44,479.92 l
76	90,143.71 l
77	182,655.17 l
80	1,518.07 m^3
85	51,608.88 m^3
90	1.75×10^{6} m^3
100	1.99×10^{9} m^3
200	5.04×10^{39} m^3

Tabla 4.3: Volumen del ADN mínimo para resolver el problema SAT

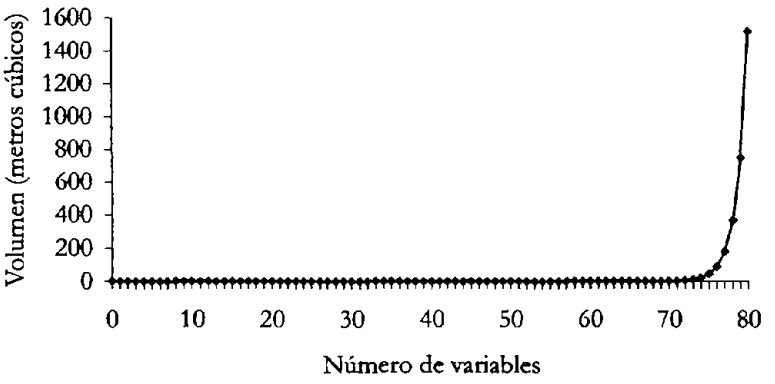

Figura 4.3: Gráfica del volumen de ADN para resolver el problema SAT

Se observa que el crecimiento en el volumen de ADN necesario para resolver el problema SAT hace que a partir de cierto número de variables, el volumen sea tan grande que difícilmente podría manejarse. Este límite de factibilidad depende en gran parte del equipo con el que se cuente.

4.3. Análisis de resultados

Puede observarse que mientras que para resolver el problema SAT de n variables en una computadora electrónica se requiere una cantidad de tiempo exponencial, para resolver el mismo problema en computadoras de ADN se requiere un tiempo polinomial. Sin embargo, en este último caso, el volumen de ADN requerido crece de manera exponencial con el número de variables.

De esta forma, tanto para computadoras electrónicas como para computadoras de ADN se tiene un límite de factibilidad que para las primeras depende de la velocidad de la computadora electrónica utilizada. Así, para la supercomputadora clasificada actualmente como la más rápida del planeta, este límite puede establecerse en 70 variables, puesto que para un problema SAT con este número de variables se requeriría un año para obtener la solución, mientras que para 71 se requeriría el doble de tiempo y a partir de ese valor, el crecimiento en el tiempo comienza a ser cada vez mayor.

En cambio, para computadoras de ADN en un año podría resolverse un problema de hasta 95 variables. Sin embargo, en este caso, el límite de

factibilidad viene de la cantidad de ADN que debe utilizarse.

Además, sólo se ha analizado la cantidad mínima de ADN que se requiere, sin considerar que para tener mayor probabilidad de obtener la respuesta correcta es necesario utilizar cantidades de ADN mayores.

En este sentido, si se coloca la cantidad mínima de ADN necesaria para resolver el problema SAT, $2^n(7n+1)$ cadenas de 20 nucleótidos cada una, la probabilidad de que las dos primeras moléculas formadas sean diferentes es el complemento de la probabilidad de que sean iguales, es decir, es:

$$1 - \frac{1}{2^n} = \frac{2^n - 1}{2^n}$$

La probabilidad de que la tercera molécula formada sea diferente a las dos primeras es el complemento de la probabilidad de que sea igual a alguna de ellas, es decir, es:

$$1 - \frac{2}{2^n} = \frac{2^n - 2}{2^n}$$

En general, la probabilidad de que la i–ésima molécula formada sea diferente a las $i-1$ primeras moléculas es:

$$\frac{2^n - i}{2^n}$$

Como la probabilidad de que todos los caminos formados sean diferentes es la probabilidad de que este último evento ocurra para todos los valores de i entre 2 y 2^n y además estos eventos son independientes, la probabilidad, P_{2_n}, de que todos los caminos formados al colocar $2^n(7n+1)$ cadenas de 20 nucleótidos cada una, sean diferentes es el producto de las probabilidades de estos eventos, es decir:

$$P_{2_n} = \frac{2^n - 1}{2^n} \cdot \frac{2^n - 2}{2^n} \cdot \ldots \cdot \frac{2}{2^n} \cdot \frac{1}{2^n} = \frac{(2^n - 1)!}{(2^n)^{2^n - 1}} \tag{4.6}$$

Esta probabilidad tiende a 0 conforme n tiende a infinito. En la tabla 4.4 se observa que P_{2_n} es prácticamente 0 para 5 variables.

Por otra parte, si se colocan $m(7n+1)$ cadenas de 20 nucleótidos cada una, la probabilidad de que la i–ésima molécula formada sea diferente a las $i-1$ primeras moléculas es $\frac{m-(i-1)}{m}$. Como la probabilidad de que todos los caminos formados sean diferentes es la probabilidad de que este último

Número de variables	Probabilidad
1	0.5
2	0.094
3	0.0024
4	1.1×10^{-6}
5	1.8×10^{-13}

Tabla 4.4: Probabilidad de que los caminos formados sean diferentes

evento ocurra para todos los valores de i entre 2 y 2^n en alguno de los subconjuntos de 2^n caminos de los m que se van a formar en total y además estos eventos son independientes, la probabilidad, P_m, de que todos los caminos formados al colocar $m(7n+1)$ cadenas de 20 nucleótidos cada una, sean diferentes en cada subconjunto es el producto de las probabilidades de estos eventos, es decir:

$$P_m = \frac{m-1}{m} \cdot \frac{m-2}{m} \cdot \ldots \cdot \frac{2}{m} \cdot \frac{1}{m} = \frac{(m-1)!}{m^{m-1}} \tag{4.7}$$

Como esto debe ocurrir en alguno de los $\binom{m}{2^n}$ subconjuntos, la probabilidad, P, de que entre todos los caminos formados al colocar $m(7n+1)$ cadenas de 20 nucleótidos cada una haya 2^n que sean diferentes es:

$$P = \sum_{i=1}^{\binom{m}{2^n}} \frac{(m-1)!}{m^{m-1}} = \binom{m}{2^n} \frac{(m-1)!}{m^{m-1}} \tag{4.8}$$

Por otra parte, los límites de factibilidad establecidos no son cantidades fijas porque están directamente relacionadas con el desarrollo actual de ambas tecnologías y con el algoritmo utilizado para resolver el problema.

En este sentido, en el siguiente capítulo se mencionarán otros tipos de algoritmos que podrían implementarse en computadoras de ADN, además de presentar las principales ventajas y desventajas de estas computadoras con relación a las computadoras electrónicas y delinear las líneas actuales y futuras de investigación en este campo.

Capítulo 5

Perspectivas

El futuro tiene la mala costumbre de volverse historia demasiado pronto.

Daniel E. Sevo

En este capítulo se describen algunas de las ventajas y desventajas de las computadoras de ADN con relación a las computadoras electrónicas actuales y se presenta una visión sobre el posible futuro de las computadoras de ADN.

5.1. Ventajas

Las características del ADN dan cierta superioridad a las computadoras de ADN sobre las computadoras electrónicas en diversos sentidos.

En cuanto a su tamaño, las cadenas de ADN pueden almacenar una gran cantidad de información ocupando un volumen muy pequeño. Prueba de ello es que toda la información genética de un ser humano se encuentra en el ADN contenido en el núcleo de una célula que mide aproximadamente 3×10^{-6} cm de diámetro y ocupa un volumen cercano a 9×10^{-18} cm^3.

En este sentido, se ha estimado que el genoma humano, es decir, la totalidad del material genético del ser humano, tiene aproximadamente 3×10^9 pares de bases [10].

Si se utiliza el modelo sticker en el que cada secuencia de 20 bases de la cadena simple correspondiente se considera un bit con valor 1 cuando se encuentre unida a su base complementaria y 0 cuando se encuentre en su

forma simple [28] se podrían almacenar $\frac{3\times10^9}{20} = 1.5\times10^8$ bits de información, equivalente a 0.0174 GB[1], en un volumen similar al del núcleo de una célula humana.

Por tanto, para almacenar 80 GB se requeriría un volumen de $\frac{80(9\times10^{-18})}{0.0174} = 4.12\times10^{-14}$ cm^3, mientras que en una computadora electrónica un disco duro de la misma capacidad ocupa un espacio mucho mayor. Así, utilizadas como medio de almacenamiento, las computadoras de ADN permitirían reducir drásticamente el espacio utilizado para ello.

Además, el almacenamiento de esta información tiene la garantía de años de evolución que han hecho de las moléculas de ADN el medio utilizado por la naturaleza para almacenar toda la información de un ser vivo con resultados favorables.

Otra ventaja esencial que ofrece su tamaño es la posibilidad de aplicar las mismas manipulaciones a una cantidad muy grande de moléculas al mismo tiempo, es decir, la posibilidad de operar sobre varias moléculas en paralelo lo que, como se ha mostrado en este trabajo, disminuye la complejidad del tiempo de solución de los problemas cuyos mejores algoritmos conocidos requieren probar una cantidad muy grande de soluciones potenciales.

En este sentido, en los capítulos anteriores se observa que la complejidad del tiempo de solución del problema del agente viajero y de la *satisfactibilidad* disminuye de exponencial a polinomial utilizando computadoras de ADN, lo que constituye un mejoramiento sustancial en la utilización de recursos de los algoritmos utilizados para resolver los problemas mencionados ya que, aunque el tiempo necesario para encontrar la solución de un problema siempre está limitado y para algún tamaño del mismo dicho tiempo es tan grande que no resulta factible resolverlo, este límite se alcanza extremadamente más rápido si la complejidad del tiempo de solución es exponencial que si se está limitado por una función polinomial.

Otra propiedad que dota de gran potencial computacional a las moléculas de ADN es la complementariedad, característica fundamental para implementar los algoritmos de solución de los problemas que han sido analizados en este trabajo. Esta propiedad también se utiliza en el modelo sticker que se ha mencionado arriba y, en general, en muchas de las técnicas de biología molecular utilizadas para manipular el ADN.

Una de las principales ventajas que otorga la complementariedad en

[1]b: bit, B: byte, kB: kilobyte, MB: megabyte, GB: gigabyte; 8 b = 1 B, 1024 B = 1 kB, 1024 kB = 1 MB, 1024 MB = 1 GB [20]

términos computacionales consiste en ser un proceso que la molécula realiza por sí misma, lo que facilita el proceso de automatización de la computación por ADN.

Además, el tamaño de las moléculas de ADN y su complementariedad las vuelve idóneas para utilizarse en nanotecnología[2] en la construcción de estructuras de nanoescala que requieren manipulaciones a nivel molecular. Uno de los métodos para ensamblar nanopartículas a través de moléculas de ADN consiste en pegar cadenas simples de ADN a las nanopartículas de manera que se unan las cadenas simples que sean complementarias provocando que al mismo tiempo se ensamblen las nanopartículas que están atadas a ellas [29].

Esta aplicación potencial es muy importante dada la extensa búsqueda de la nanotecnología por encontrar métodos que permitan hacer manipulaciones a escala molecular de manera masiva y con precisión y dadas las muchas aplicaciones que tiene la construcción de estructuras a escala nanométrica.

Por otra parte, las reacciones en el ADN ocurren de manera casi instantánea. Por ejemplo, la velocidad a la cual la polimerasa sintetiza cadenas de ADN es de 1000 pares de bases por segundo, mientras que se pueden realizar más de 10^{14} operaciones de ligado por segundo [29]. Esta gran velocidad podría permitir que, al menos en teoría, los problemas fueran resueltos con rapidez.

Además, la energía utilizada por las computadoras de ADN es muy baja. Si cada ligado de moléculas de ADN se considera la operación fundamental, se pueden realizar 2×10^{19} operaciones por joule, mientras que las supercomputadoras actuales sólo realizan 10^{9} operaciones por joule [29]. Por otra parte, dado que al unirse las moléculas de ADN liberan cierta cantidad de energía, parece posible aprovechar ésta para disminuir aún más la energía total requerida, aunque también debe considerarse la energía que se requiere, por ejemplo, para activar el campo eléctrico en la electroforesis o para elevar la temperatura de la solución en la aplicación de la técnica de PCR [2].

Por todo lo anterior se observa que las computadoras de ADN superan a las computadoras electrónicas en muchos aspectos. Sin embargo, presentan ciertas desventajas que serán detalladas en la siguiente sección.

[2]La nanotecnología estudia la manipulación de moléculas y átomos para crear materiales y dispositivos de reducidas dimensiones; nano- antepuesto al nombre de una unidad de medida significa la milmillonésima parte de la misma [25].

5.2. Desventajas

Las desventajas que presentan las computadoras de ADN con relación a las computadoras electrónicas provienen de la forma en que se manipulan y de la naturaleza de los algoritmos utilizados.

En cuanto a su manipulación, el manejo de preparaciones con ADN, que es extremadamente cambiante con las condiciones de temperatura y pH, requiere de condiciones muy rigurosas, las cuales se deben acentuar aún más por el riesgo de contaminacin de los sistemas biológicos por microorganismos. En este sentido, aunque las reacciones en el ADN son muy rápidas, el tiempo requerido actualmente para preparar los experimentos y llevar a cabo las manipulaciones que implica la construcción de una computadora de ADN son muy grandes, volviéndolas lentas en las resolución de problemas cuya solución no requiere un gran paralelismo.

Además, la forma actual de manipular el ADN permite que se cometan diversos errores. Por ejemplo, al verter las moléculas de ADN para formar el conjunto solución en el algoritmo de Lipton, podrían quedar moléculas pegadas a los tubos de ensayo y no obtenerse la solución correcta [28].

Incluso cuando estas manipulaciones se realizaran sin cometer ningún error, los procesos bioquímicos tienen errores intrínsecos. Por ejemplo, la polimerasa Taq que se usa en la técnica de PCR tiene una tasa de mutación de 1 por cada mil, es decir, uno de cada mil nucleótidos que agrega no corresponde al que debería haber agregado.

Estos errores hacen que el usar computadoras de ADN no garantice la obtención de la solución correcta y aunque las computadoras electrónicas también están sujetas a errores, actualmente la probabilidad de que existan errores de hardware es muy baja, por lo que son mucho más precisas que las computadoras de ADN [3].

También puede haber errores por cadenas complementarias en la mayor parte de su longitud que se unan parcialmente, pero los algoritmos están pensados para que las cadenas que se unan sean complementarias en toda su extensión.

Por otra parte, para programar las computadoras de ADN es necesario conocer las reacciones bioquímicas implicadas, tener acceso a un laboratorio, manejar las técnicas utilizadas y contar con ADN sintético, enzimas y reactivos para manipularlo, además de todo el equipo necesario para preparar el experimento con los costos que esto implica. Para ejemplificar este punto, basta mencionar que en el Instituto de Fisiología Celular de la UNAM

sintetizar una secuencia sencilla de ADN de 20 nucleótidos cuesta alrededor de 28 dólares[3].

Otra desventaja fundamental es que las computadoras de ADN no son reutilizables, aunque, dependiendo del problema a resolver, el algoritmo podría adaptarse para que al menos una parte de los pasos no tengan que repetirse. Por ejemplo, en el algoritmo de Lipton la gráfica que codifica todas las combinaciones de valores de verdad puede generarse solamente una vez para cada tamaño de la gráfica y mantener almacenada una o más copias de la misma.

Además, si los algoritmos utilizados no aprovechan el paralelismo proporcionado por las computadoras de ADN, bajo las condiciones actuales resultarían mucho más tardadas que en computadoras convencionales.

En cuanto a la resolución de problemas NP con computadoras de ADN, los algoritmos utilizados actualmente para su resolución en computadoras de ADN no son deterministas, lo que hace que no se tenga un control total de las mismas y constituye otra fuente de errores.

En este sentido, tanto en la computadora de Adleman como en la computadora de Lipton es necesario que se formen todos los caminos posibles en la gráfica utilizada para representar el problema, pero la formación de estos caminos no está garantizada aunque se coloque una cantidad muy grande de ADN, por lo que podría ocurrir que el camino que es la solución del problema no se formara.

Además, al tratar de eliminar las soluciones potenciales que no sean soluciones al problema en cuestión, se asume que al amplificar con PCR las cadenas que cumplan con ciertas condiciones se conservan solamente las cadenas con estas características porque al tomar cadenas al azar en el siguiente paso del algoritmo que se esté implementando lo más probable es que se tome una de las cadenas que fueron amplificadas. Sin embargo, aún existe la posibilidad de que se tome una de las cadenas que no cumplan con estas características porque aún están presentes en la solución que contiene a todas las cadenas.

La disminución de estos errores implica un aumento en la cantidad de ADN utilizado, cantidad que además tiende a crecer exponencialmente con el tamaño del problema. Como se ha establecido, en el caso del problema del camino hamiltoniano el crecimiento es $O(n!)$, mientras que en el caso del

[3]Laura Ongay. Unidad de Biología Molecular. Instituto de Fisiología Celular, UNAM: Comunicación personal

problema SAT es $O(n2^n)$.

Sin embargo, aunque los errores mencionados pueden llevarnos a declarar que una respuesta incorrecta resuelve el problema NP en cuestión, la validez de esta respuesta podrá ser verificada con rapidez, ya que, como se ha establecido antes, los problemas NP tienen la característica de que una vez que se cuenta con una solución potencial, el tiempo que se requiere para verificar si ésta es solución del problema, es de orden polinomial. Pero si la respuesta del problema NP es que no tiene solución no se puede garantizar que sea así; sólo se puede afirmar que no se encontró ninguna solución con el algoritmo utilizado.

5.3. El futuro de las computadoras de ADN

Del análisis anterior, se deduce que la principal ventaja de las computadoras electrónicas sobre las computadoras de ADN es su nivel de desarrollo actual. Mientras la primera computadora electrónica fue construida en 1936 por el alemán Konrad Zuse, la primera computadora de ADN se construyó en 1994.

Sin embargo, las ventajas de las computadoras de ADN las vuelven potencialmente más rápidas y eficientes que las computadoras electrónicas. De esta forma, dado el rápido avance de la tecnología en general y las posibilidades teóricas de las computadoras de ADN y del cómputo molecular en general, es muy probable que mejoren los tiempos de preparación de experimentos y manipulaciones biológicas, que disminuyan los errores de las manipulaciones, que se desarrollen algoritmos cuya implementación implique la disminución de la tasa de error provocada por el carácter aleatorio de los mismos y que se independice la programación de las computadoras de ADN de las técnicas biológicas usadas en su funcionamiento, aunque esto último quizá en un plazo no tan corto.

Además, modificando algunos detalles de los algoritmos propuestos hasta ahora pueden hacerse mejoras para aumentar la velocidad de algunas reacciones. Por ejemplo, la rapidez con la cual se hibridizan las cadenas complementarias depende del nivel de repetición de nucleótidos en dichas cadenas. Es decir, una cadena formada solamente por adeninas se une más rápidamente a su complemento que una cadena en la que se repita algún patrón de nucleótidos y ésta, a su vez, se une más rápido a su complemento que una cadena conteniendo los cuatro tipos de nucleótidos distribuidos al azar en forma uniforme.

Por tanto, asignando una secuencia con un patrón repetitivo a los vértices y las líneas de las gráficas utilizadas para resolver los problemas analizados, el tiempo que tomaría la formación de los caminos se reduciría[4].

Por otra parte, ya se ha considerado la posibilidad que las computadoras de ADN se combinen con las computadoras electrónicas o con algunos robots para que éstos realicen las manipulaciones biológicas de manera automática[2]. Con estos avances potenciales, podrían construirse computadoras híbridas que tengan tanto componentes electrónicos como moleculares y que, de esta forma, aprovechen las ventajas que ofrecen ambas tecnologías.

En cuanto a la desventaja del crecimiento exponencial en la cantidad de ADN que se requiere para resolver ciertos problemas, ya se está atacando al buscar nuevos algoritmos que generan sólo algunas soluciones potenciales y si éstas no funcionan, generan otras [3]. De esta manera se incrementa el tiempo, pero la cantidad de moléculas disminuye. Lo importante en esta clase de algoritmos es encontrar la combinación óptima entre tiempo y cantidad de ADN que haga factible resolver problemas de mayores dimensiones.

Es importante aclarar que las computadoras de ADN se consideran un paradigma diferente de computación en el sentido de que implican una nueva forma de análisis, diseño e implementación de sistemas computacionales, además de cambiar la complejidad de los recursos usados en la solución de los problemas, pero no alteran el concepto de computabilidad porque cualquier operación que pueda realizarse en una computadora de ADN puede ser simulada por una Máquina de Turing [27].

Lo que resulta sorprendente es que a través de diferentes modelos, entre ellos el modelo sticker que en general tienen como operaciones comunes las presentadas en la sección 1.3, se ha demostrado teóricamente que las computadoras de ADN son un modelo universal de cómputo [7], es decir, que en ellas se puede ejecutar cualquier algoritmo computacional o, dicho de otra forma, que las computadoras de ADN pueden simular una Máquina Universal de Turing [23].

Sin embargo, aunque las computadoras de ADN son equivalentes a una máquina de Turing, probablemente no reemplacen a las computadoras convencionales, sino las complementen y sean utilizadas especialmente para resolver problemas en los que sea conveniente aprovechar el paralelismo ofrecido por las mismas para disminuir el tiempo de solución hasta una escala aceptable de tiempo [23].

[4]Alfonso Vilchis Peluyera. Laboratorio de Biología Molecular. Facultad de Ciencias, UNAM: Comunicación personal.

Por otra parte, dado que hay muchos modelos de cómputo molecular basado en ADN es de esperar que los trabajos posteriores, tanto teóricos como prácticos, indiquen cuál es el más adecuado para ser implementado con niveles de error aceptable y una ejecución eficiente [23], Aunque probablemente algunos modelos sean más adecuados para resolver cierto tipo de problemas y otros modelos se usen en la resolución de otros problemas.

En este sentido, aunque los algoritmos propuestos para resolver el problema del camino hamiltoniano y el problema SAT utilizan el enfoque de generar primero todas las soluciones potenciales y luego eliminar las que no sean soluciones al problema, se han planteado algoritmos para resolver problemas prácticos, como el ataque al Estándar de Encriptación de Datos de los Estados Unidos (DES: Data Encryption Standard) [5] usando el modelo sticker. En [2] se sugiere que este ataque podría llevarse a cabo en 2 horas en una mesa usando aproximadamente un gramo de ADN, algunos brazos mecánicos, bombas y controladores de energía manipulados a través de una computadora electrónica.

Otras aplicaciones potenciales incluye la utilización de la tecnología para el manejo de bases de datos. Por ejemplo, Masami Hagiya y sus colegas del departamento de ciencias de la información en la Universidad de Tokio han propuesto y probado experimentalmente métodos para implementar la operación *join* de una base de datos relacional por amplificación con PCR, siendo las *tuplas*[5] moléculas de ADN [29].

Aún hay muchos problemas teóricos y prácticos relacionados con la implementación de algoritmos en computadoras de ADN que requieren ser analizados y resueltos, pero también hay una gran cantidad de científicos trabajando en ellos, por lo que aunque el campo está empezando tiene grandes posibilidades de desarrollarse hasta convertirse en una tecnología importante que resuelva o al menos ayude a resolver problemas importantes tanto teóricos como prácticos.

[5]Las tuplas son los renglones de una tabla en una base de datos relacional [12].

Conclusiones

Con el desarrollo de este trabajo se ha observado la aplicabilidad de diversos conceptos matemáticos al análisis de las ventajas ofrecidas por las computadoras de ADN y se ha mostrado que la licenciatura en Matemáticas Aplicadas y Computación brinda las bases necesarias para implicarse en diversas disciplinas de la ciencia.

En cuanto al objetivo del trabajo, se ha mostrado que mientras en las computadoras electrónicas existe un límite de factibilidad para resolver el problema SAT porque su tiempo de solución crece de manera exponencial, utilizando computadoras de ADN la complejidad del tiempo de solución disminuye puesto que éste crece de manera polinomial con relación al número de variables del problema.

Por otra parte, aunque en computadoras de ADN también existe un límite de factibilidad determinado por el crecimiento exponencial del volumen de ADN requerido para resolver el problema, la disminución en la complejidad del tiempo de solución de este problema es muy importante porque se trata de un problema NP completo de manera que se cuenta con un algoritmo de tiempo polinomial para resolver todos los problemas NP.

Además, está abierta la posibilidad de desarrollar algoritmos que disminuyan la complejidad del volumen de ADN requerido aun cuando aumenten la complejidad del tiempo de solución porque la disminución de éste último es radical utilizando computadoras de ADN y si se trabaja en el desarrollo de estos algoritmos podrían equilibrarse los recursos de tiempo y ADN de tal manera que se puedan resolver problemas NP de tamaño mucho mayor a los que actualmente se resolven.

De manera general, dado el desarrollo actual de las computadoras de ADN y de las computadoras electrónicas, mientras las técnicas usadas para manipular computadoras de ADN y los algoritmos que se pretenden implementar en ellas no mejoren lo suficiente como para permitir que sus errores

disminuyan hasta un nivel aceptable y que su programación se independice de las manipulaciones biológicas, las computadoras de ADN no sustituirán a las computadoras electrónicas y su uso seguirá a nivel teórico y no práctico.

Sin embargo, las computadoras de ADN son una tecnología relativamente nueva y actualmente se están desarrollando numerosas investigaciones encaminadas al logro de los objetivos anteriores. Estas investigaciones son impulsadas por las ventajas presentadas por dichas computadoras sobre las computadoras convencionales, particularmente por su gran capacidad de almacenamiento y por su enorme paralelismo, característica que permite atacar problemas NP en un tiempo polinomial con algoritmos de búsqueda exhaustiva a diferencia de las computadoras electrónicas que requieren un tiempo exponencial con el mismo tipo de algoritmos.

Finalmente, dadas todas las ventajas y amplias posibilidades de desarrollo de las computadoras de ADN, los avances en este campo prometen una tecnología con muchas e importantes aplicaciones potenciales que contribuyan a facilitar el desarrollo de otras áreas al proporcionar soluciones a problemas que hasta ahora no han podido resolverse por requerir de mayor precisión y poder de procesamiento que los ofrecidos por las computadoras convencionales.

Bibliografía

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problem. *Science*, 266, (noviembre de 1994). 1021–1024.

[2] L. M. Adleman *et al.* On applying molecular computation to the data encryption standard. En: *2nd annual workshop on DNA Computing, Princeton University* L. Landweber y E. Baum (eds.) *DIMACS: series in Discrete Mathematics and Theoretical Computer Science*, 27, American Mathematical Society (1996). 31–44.

[3] S. Baase *et al. Algoritmos computacionales* (Introducción al análisis y diseo). México, Addison Wesley, tercera edition, 2002.

[4] E. Bernstein y U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5), (1997). 1411-1473.

[5] D. Boneh *et al.* Breaking des using a molecular computer. En: *1st DIMACS workshop on DNA Computing,* R. Lipton y E. Baum, (eds.) *DIMACS: series in Discrete Mathematics and Theoretical Computer Science*, 27, American Mathematical Society (1995). 37–66.

[6] J. G. Brookshear. *Teoría de la computación* (Lengjuajes Formales, autómatas y complejidad). México, Addison-Wesley, 1990.

[7] C. S. Calude y G. Păun. *Computing with cells and atoms* (An introduction to quantum, DNA and membrane computing). Nueva York, Taylor & Francis, 2001.

[8] A. Carpi. "Enlaces químicos". National Science Foundation, (2003). <*http://www.visionlearning.com/index.php*> [Consulta: 07 de marzo del 2005].

[9] M. Chitty. "Biopharmaceutical glossary". Cambridge Healthtech Institute, (2004). <*http://www.genomicglossaries.com/default.asp*> [Consulta: 12 de febrero del 2005].

[10] J. S. Choinski. *Experimental cell and molecular biology.* Iowa, Brown Communications, 1992.

[11] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 28, (1936). 345–363.

[12] C. J. Date. *Introducción a los sistemas de bases de datos.* México, Prentice Hall, 2001.

[13] s/a. "Earth Simulator Center". Japan Agency for Marine-Earth Science and Technology, (2004). <*http://www.es.jamstec.go.jp/*> [Consulta: 15 de octubre de 2004].

[14] s/a. *Enciclopædia Britannica* (in 30 volumes). Chicago, The University of Chicago, 1981.

[15] R. P. Feynman. "There's plenty of room at the bottom", (2002). <*http://www.zyvex.com/nanotech/feynman.html*> También disponible en <*http://elearning.stut.edu.tw/m_facture/Nanotech/Web/ch11.htm*> [Consulta: 19 de junio del 2004].

[16] J. Gross. *Graph Theory* (and its applications). Nueva York, CRC Press, 1998.

[17] F. Harary. *Graph Theory.* Nueva York, Addison Wesley, 1969.

[18] D. Harel. *The science of computing* Exploring the nature and power of algorithms. Nueva York, Addison-Wesley, 1989.

[19] J. Hickman *et al.* "Biological computation: How does biology do information technology?". National Science Foundation, (1999). <*http://www.csd.uwo.ca/~lila/biocomp.html*> [Consulta: 26 de marzo del 2004].

[20] P. A. Laplante. *Dictionary of computer science, engineering and technology.* Nueva York, CRC Press, 2001.

[21] A. M. Lesk. *Introduction to Bioinformatics.* Nueva York, Oxford University, 2002.

[22] R. J. Lipton. Using DNA to solve NP–complete problems. *Science*, 268, (abril de 1995). 542–545.

[23] C. Maley. DNA computation: Theory, practice and prospects. *Evolutionary Computation*, 6, Massachusetts Institute of Technology (1998). 201–229.

[24] K. Mullis *et al.* Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. *Cold Spring Harbor Symposia on Quantitative Biology*, LI, (1986). 263–273.

[25] H. S. Nalwa. *Encyclopedia of nanoscience and nanotechnology.* California, American Scientific, 2004.

[26] M. A. Nielsen e I. L. Chuang. *Quantum computation and quantum information.* Cambridge, GB, Cambridge University Press, 2000.

[27] G. Păun *et al. DNA Computing* (New Computing Paradigms). Texts in theoretical computer science. Nueva York, Springer, 1998.

[28] S. Roweis *et al.* A sticker based architecture for DNA computation. En: *2nd annual workshop on DNA Computing, Princeton University,* L. Landweber y E. Baum, (eds.) *DIMACS: series in Discrete Mathematics and Theoretical Computer Science* American Mathematical Society, (1996). 1–29.

[29] A. J. Ruben y L. F. Landweber. The past, present and future of molecular computing. *Nature reviews*, 1, (octubre del 2000). 69–72.

[30] F. Sanger *et al.* Determination of a nucleotide sequence in bacteriophage f1 DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 90(2), (diciembre de 1974). 315-333.

[31] B. Schneier. *Applied cryptography* (Protocols, algorithms, and source code in C). Nueva York, John Wiley & Sons, 1996.

[32] P. Sharma *et al.* Construction of subtractive cDNA library using magnetic beads and pcr. En: *PCR primer, a laboratory manual Cold Spring Harbor Laboratory Press, Plainview* C. W. Dieffenbach y G. S. Dveksler (eds.), Nueva York (1995). 439–452.

[33] s/a. "Top 500 supercomputer sites". Universidad de Mannheim, Universidad de Tennessee, NERSC/LBNL, (Junio del 2004). <*http://www.top500.org*> [Consulta: 15 de octubre de 2004].

[34] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, Series 2, 42, (1936–37). 230–265.

[35] J. D. Watson y F. H. C. Crick. A structure for deoxyribose nucleic acid. *Nature*, 171, (abril de 1953). 737.

Índice alfabético