Aplicación de Técnicas Evolutivas para el Problema de Plegado de Proteínas

David Alejandro Pelta Pablo Daniel Oña

Departamento de Informática
Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Índice General

Índice de Figuras

Índice de Tablas

Dedicatorias y Agradecimientos

Antes de comenzar con los agradecimientos individuales, queremos mencionar las personas que han contribuído en la realización de este trabajo.

En primer lugar, vaya nuestro reconocimiento a Natalio Krasnogor y al grupo Lifia-BioCom, sin los cuales esta tesis no hubiera sido posible. Sus aportes, sugerencias y recomendaciones han sido invalorables.

A Gabriel Baum por haber confiado en nosotros.

A todos nuestros amigos que tuvieron que escuchar acerca de algoritmos genéticos, *HeadLess Chicken Test*, Protein Folding y otras cosas raras y lo hicieron estoicamente.

A los que sufrieron prestando sus máquinas para correr las innumerables e inofensivas simulaciones de esta tésis.

A todos ellos, infinitas gracias.

Los agradecimientos de David

Esta tesis marca de alguna manera el final de mi carrera de Licenciado en Informática.

Todavía recuerdo las primeras clases del curso de Ingreso donde el Prof. Kaplan daba sus clases de forma magistral enseñandonos las cosas mediante ejemplos memorables como el del "aeropuerto de moscas" y la "fiestita en lo de Luque".

Es ahora el momento de recordar y agradecer a todos aquellos que contribuyeron para que pudiera llegar a esta instancia final.

En primer lugar deseo agradecer a mi familia que hizo un enorme esfuerzo y que siempre me apoyó para que pudiera llegar hasta aquí; existe un viejo dicho que dice que la familia no se elige, te toca. Si esto es cierto, soy muy afortunado: mi viejo, mi mamá y mi hermana son excelentes personas en todos los sentidos, son lo más importante que tengo y mi carrera no hubiera sido posible sin ellos. Esta tesis se las dedico con todo mi afecto.

No tengo palabras para describir lo que siento por la Bobe y el Zeide (mis abuelos maternos). Con estos dos viejos de casi 90 años, he compartido infinidad de cosas y sea donde fuere que me lleve la vida, siempre recordaré con cariño los almuerzos del domingo al mediodía, sus peleas por pavadas, la compota, los knisches y los blintzes (porque nadie los hace como la Bobe), la asombrosa capacidad del Zeide para recordar la superficie de Kamchatka y la población de Burundi y su casi insoportable obsesión para que los impuestos estén pagos al otro día de haber llegado el recibo.

Sin dudas, y sobre todo en estos últimos ocho años, han sido y son una parte muy importante de mi vida.

Quiero recordar y agradecer también a muchas otras personas por su constante apoyo y colaboración. Espero no olvidarme de ninguno y quiero aclarar que el orden de aparición no necesariamente es el orden de importancia.

A mis amigos de "allá", los de Tornquist, y a los de acá (en especial a los miembros del club de la bota), sin los cuales mi vida hubiera sido muy aburrida.

A Pablo Moscato, mi primer director en esto de la investigación y el responsable de haberme presentado el Problema de Protein Folding. Todavía guardo aquella primera versión de mi primer paper, cuidadosamente destruída por sus correcciones.

A Natalio Krasnogor, con quien compartimos éxitos, fracasos, e infinidad de proyectos futuros, los cuales no hubieran sido posibles sin que nos uniera una gran amistad.

A Fernando Lyardet, Natalia Romero y Majo Presso por su valiosa amistad.

A Verónica Argañaraz, mi mejor amiga, por las interminables charlas y sesiones de autoayuda en las que nos sumergíamos en algunos momentos difíciles.

A Soledad Escobar, por tantos momentos compartidos y "locuras" realizadas.

A Yamila Lorenzo, por su increíble aguante durante los primeros años de mi carrera y por todos los momentos vividos.

A los Labarthe: María, Ceci, Fede, Ale, y también Alberto, quien seguramente hubiera conpartido la alegría de este momento.

A mis compañeros de Suma, en especial a Andrés, Roberto, Inés, Lorena y Alejandro, con quienes me une algo más que coincidencias ideológicas. Sin dudas mi participación en la agrupación fue uno de los hechos más relevantes de mi carrera universitaria y siempre los recordaré con respeto y admiración.

A Pablo Oña, porque tuvo que sufrir con LaTex y gcc y lo soportó estoicamente, y porque además es un buen tipo.

A Gustavo Rossi, con quien hemos tenido más de una discusión, pero con quien compartimos objetivos comunes.

A toda la gente del Lifia-Λ y en especial al grupo BioCom. Mi más sincero agradecimiento a Walter, Daniel, Germán, Steve, Natalio, Wanda, Vanesa, Agustina, Hernán, Germán y el Colo por las experiencias compartidas y sus esfuerzos para con el grupo.

Quiero agradecerle a Gaby su infinita paciencia por haber aceptado postergar algunas cosas hasta "cuando termine la tesis". Su amor y comprensión han sido fundamentales para mí. "Yo también te quiero".

Por suerte, ya está!. Ahora podremos empezar los preparativos para nuestro ... y podremos comenzar a delirar. El futuro es nuestro, pituti!!!

Finalmente, esta tesis también está dedicada a todos aquellos que no han tenido las mismas oportunidades que yo, y que día a día sufren, trabajan y pelean por lograr un país mejor y para todos.

Los agradecimientos de Pablo

Es difícil saber que poner en este punto, el agradecer a tantas personas que me apoyaron desde que ingresé en esta facultad alla por el '91.

Recuerdo verme asustado ante tremendo desafío esos primeros días de febrero del '91, comenzaba el curso de ingreso y no tenía ningún conocido en la facultad.

Luego de tres días de comenzado el curso de ingreso me encuentro con Mauro Sayavedra, un compaero de los pagos, juntos comenzamos a estudiar y a conocer mucha gente. A traves de los años de estudio logramos formar un gran grupo humano, y por suerte hoy en día muchos estamos terminando la carrera de Licenciatura en Informática, otros se recibieron de Analistas de Computacin.

Gracias Mauro, Martin (El Lindo), Damian, David, Tincho, Pili, Facundo, Tedy, Matias, Gustavo, Nacho, Natalia, Veronica, Majo, Adriana, Belen, Silvio por todos estos años de estudio y por sobre todo de amistad.

Quiero agradecer muy especialmente a mi abuela Otilia Mellado de Condello, quien me enseño a que todo era posible si uno se lo proponía, gracias abuela se que que hubieras querido estar presente en esta oportunidad.

Le quiero dedicar esta tesis a mi novia Liliana, a quien amo profundamente, y agradecerle la paciencia, comprensión y apoyo durante todos estos años.

También le quiero agradecer a mi compañero de tesis David Pelta, por los gratos momentos y situaciones que vivimos juntos no solo en esta tesis, sino durante toda la carrera. Con vos aprendí el valor del espiritu del investigador, a sacar lo bueno de lo malo, a ser crítico de lo bueno.

Finalmente, quiero agradecerle, a mis padres que gracias a su esfuerzo, me dieron la posibilidad de estudiar en esta facultad. Este trabajo va dedicado a ellos.

Capítulo 1

Introducción

La Biología Molecular se dedica fundamentalmente al estudio de la estructura y funcionalidad de proteínas y ácidos nucleicos.

A partir del descubrimiento de la estructura en doble hélice del ADN en 1953, el área ha tenido notorios avances.

El volumen de información generado a partir de la manipulación de secuencias biomoleculares y la creciente potencia de las computadoras para realizar simulaciones de procesos biológicos complejos, han provocado que los Biológos Moleculares deban interactuar con sus pares de las Ciencias de la Computación y las Matemáticas para poder aprovechar la información generada.

Como consecuencia de esta interacción surge la **Biología Computacional**: área que involucra el desarrollo y uso de técnicas matemáticas y de computación para facilitar el tratamiento de los problemas derivados de la Biología Molecular. Como ejemplo de este trabajo interdisciplinario podemos citar la aplicación de técnicas de bases de datos para almacenar la creciente cantidad de secuencias moleculares descubiertas, y que a través de Internet pueden consultarse, compararse para buscar similitudes y/o diferencias, etc.

Por otro lado, en los últimos 30 años ha existido un creciente interés en el desarrollo de técnicas computacionales para la resolución de problemas complejos, que se basan en la utilización de los principios de evolución y herencia.

Estas técnicas permiten obtener soluciones aproximadas de problemas para los cuales, la obtención de la solución óptima es imposible (ya sea por costo o por tiempos).

Tres elementos son característicos en estas técnicas: el mantenimiento de un conjunto (*población*) de potenciales soluciones (*individuos*), la utilización de algún proceso de selección basado en la calidad (*fitness*) de las mismas, y la incorporación de operadores "genéticos" que permiten modificar o generar individuos.

Como ejemplos de estas técnicas podemos citar: Algoritmos Genéticos, Algoritmos Meméticos, Genetic Programming, Evolution Strategies, etc.

En general la Biología Computacional se asocia con problemas de comparación de secuencias, pero en realidad abarca un espectro mucho más amplio de problemas, como por ejemplo: la construcción de arboles filogenéticos para determinar antepasados comunes entre especies, el rearreglo de genomas, el mapeo físico de cromosomas y los problemas de predicción de estructuras moleculares.

Estos problemas son, en general, "dificiles" de resolver. Por lo tanto la aplicación de técnicas que permitan obtener soluciones aproximadas se vuelve indispensable.

En este trabajo investigaremos la aplicación de técnicas evolutivas a uno de los problemas más dificiles del área: el llamado *Protein Folding Problem*[1], o *Problema de Doblado de Proteínas.*

Esto implicará el abordaje de areas tan disímiles como la biología molecular, la complejidad computacional y el diseño, codificación y testeo de las técnicas evolutivas a utilizar.

1.1 Organización del Trabajo

Este trabajo consta de dos grandes partes divididas en 8 capítulos. En la primera parte se introducen los conceptos necesarios para la comprensión del trabajo y en la segunda, se desarrollan nuestras ideas, se describen nuestras hipótesis y los mecanismos utilizados para intentar verificarlas.

A continuación se encuentra una descripción breve del contenido de cada capítulo.

En el capítulo 2 se introducen algunos conceptos de biología molecular y se define el problema encarado en esta tésis. Se describen los ácidos nucleicos, las proteínas y su proceso de fabricación en la célula, hasta llegar a la definición de *PF*. Se plantean las hipótesis asociadas al problema y se describen brevemente los mecanismos utilizados en la actualidad para intentar resolverlo.

El capítulo 3 comienza con una breve reseña de los conceptos esenciales de la teoría de la complejidad computacional, y continúa con una introducción a las técnicas heurísticas. Se describen las técnicas evolutivas en general y se desarrollan secciones especiales para los algoritmos genéticos y la programación genética.

Luego, en el capítulo 4, se exponen las modelizaciones más utilizadas para representar *PF*. Se describe la complejidad de los problemas asociados y se reseñan algunos resultados respecto a la relación secuencia-estructura implicada por esos modelos.

El capítulo 5 comienza con la descripción de dos trabajos representativos de la aplicación de algoritmos genéticos para *PF*. Se detectan algunas falencias y se describen los experimentos realizados para verificarlas. Posteriormente se sugieren las causas de estas fallas y se proponen mecanismos para corregirlas.

Uno de los mecanismos sugeridos es la utilización de autómatas celulares (desarrollados en el capítulo 6).

En el capítulo 7 se propone su aplicación para resolver *PF*; se plantean los objetivos a corto y largo plazo, las hipótesis de trabajo y se describen los experimentos computacionales realizados para intentar verificar nuestras ideas.

Según nuestro conocimiento, es la primera vez que se intenta combinar autómatas celulares con *PF*.

Finalmente aparecen las conclusiones y una lista de trabajos futuros (Cap. 8.2) que surgen como continuación natural del trabajo aquí presentado.

[1] lo llamaremos PF o PFP de aquí en adelante

Capítulo 2

Conceptos de Biología

2.1 Introducción

En la naturaleza existen seres vivientes y no vivientes. Los seres vivientes pueden moverse, reproducirse, crecer, comer, etc. Se dice que tienen *participación activa* en su ambiente, en contrapartida con los seres no vivientes. Ambos presentan la misma composición y se rigen por las mismas reglas físicas y químicas.

Los seres vivientes actúan debido a continuas reacciones químicas internas. Un organismo viviente intercambia constantemente materia y energía con su entorno. En contraste, todo aquello que esta en equilibrio con su entorno es considerado generalmente muerto (existen algunas excepciones que son las formas vegetativas como las semillas, virus, los cuales pueden estár inactivos durante largos períodos).

La vida comenzó hace aproximadamente 3.5 billones de años y la primer forma de vida era simple; luego de billones de años de evolución los organismos simples se diversificaron formando organismos complejos. Ambos organismos, tanto los simples como los complejos, tienen una química o bioquímica molecular similar.

Los principales actores en la química de la vida son las proteínas y los ácidos nucleicos. Estos dos elementos son el objeto de estudio de la Biología Molecular.

Tanto las proteínas como los ácidos nucleícos son polímeros, es decir, cadenas moleculares formadas por la unión de unidades más simples llamadas monómeros.

En el ADN los monómeros son los nucleótidos, y en la proteínas, los aminoácidos.

En las secciones siguientes, presentaremos las principales características del ADN, describiremos las proteínas y su producción, hasta llegar a la definición del problema que atacaremos en este trabajo: *el Problema de Doblado de Proteínas*

2.2 Proteínas

En primer lugar definiremos qué son y para que sirven las proteínas [1].

Las proteínas se descubrieron en 1838 y hoy se sabe que son los ingredientes principales de las células y que suponen más del 50% del peso seco de los animales.

En nuestro cuerpo encontramos basicamente dos tipos de proteínas:

- las llamadas estructurales, que forman estructuras complejas como el cabello,

[1] El material de este capítulo fue extraído de [MS96, Wat96]

- las enzimas, cuya función es acelerar o retardar las reacciones químicas.

Son moléculas grandes, específicas de cada especie y de cada uno de sus órganos.

Se estima que el ser humano tiene unas 30.000 proteínas distintas, de las que sólo un 2% se han descripto en profundidad.

Además de intervenir en el crecimiento y el mantenimiento de las células, las proteínas son responsables de la contracción muscular. Las enzimas digestivas también son proteínas, al igual que la insulina, los anticuerpos del sistema inmunológico y la hemoglobina, que transporta oxígeno en la sangre. Casi todas las hormonas son proteínas

Todas ellas, desde las humanas hasta las que forman las bacterias unicelulares, son el resultado de las distintas combinaciones entre monómeros llamados aminoácidos.

Existen 20 aminoácidos diferentes (ver Tabla 2.1) y todos comparten una estructura similar: tienen un átomo de carbono central, conocido como carbono alfa (C_α). al cual se ligan un átomo de hidrógeno, un grupo amino (NH_2), un grupo carboxilo ($COOH$), y una cadena lateral R (ver Fig.2.1). Es esta cadena lateral la que distingue un aminoácido del otro. Por ejemplo, en la glicina, el más simple de los ácidos, la cadena R se compone de un único átomo de hidrógeno. En otros aminoácidos, la cadena R es más compleja, conteniendo carbono e hidrógeno, así como oxígeno, nitrógeno y azufre.

Los planos de los enlaces que van desde el C_α al hidrógeno y desde el C_α al grupo carboxilo, pueden estar rotados (ver Fig. 2.2) Los ángulos de rotación asociados se denominan ϕ y ψ y pueden tomar una cantidad restringida de valores a la cual llamaremos z. Se estima que $z \approx 10$

Código de una letra	Código de tres letras	Nombre
A	Ala	Alanina
C	Cys	Cisteína
D	Asp	Ácido Aspártico
E	Glu	Ácido Glutámico
F	Phe	Fenilalanina
G	Gly	Glicina
H	His	Histidina
I	Ile	Isoleucina
K	Lys	Licina
L	Leu	Leucina
M	Met	Metionina
N	Asn	Asparagina
P	Pro	Prolina
Q	Gln	Glutamina
R	Arg	Arginina
S	Ser	Serina
T	Thr	Treotina
V	Val	Valina
W	Trp	Triptofano
Y	Tyr	Tirosina

Tabla 2.1: Los veinte aminoácidos más comunes en la Naturaleza

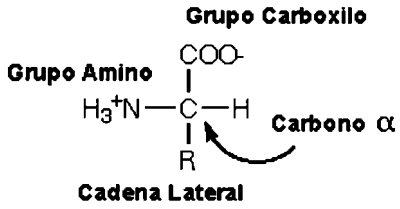

Figura 2.1: Estructura general de los aminoácidos

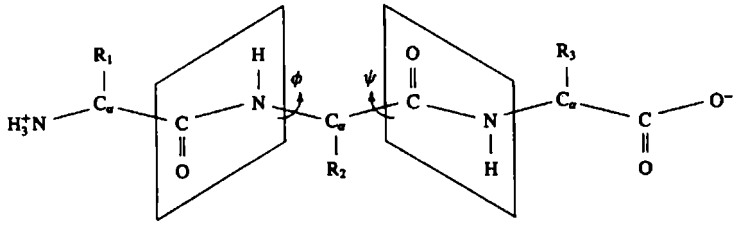

Figura 2.2: Cadena polipeptica. Los átomos dentro de cada cuadrado pertenecen al mismo plano, el cual puede estar rotado según los ángulos ϕ y ψ. Figura tomada de [MS96]

Cuando una célula viva sintetiza proteínas, el grupo carboxilo de un aminoácido reacciona con el grupo amino de otro, formando un enlace peptídico y liberando una molécula de agua. (ver fig 2.3) El grupo carboxilo del segundo aminoácido reacciona de modo similar con el grupo amino del tercero, y así sucesivamente hasta formar una larga cadena (entre 50 y cientos de aminoácidos).

Los enlaces peptídicos, en conjunción con los ángulos ϕ y ψ, permiten determinar el esqueleto o *backbone* de la proteína.

Podemos decir que la cadena "adopta" una estructura tridimensional que es la que determina la funcionalidad biológica de la proteína; es una estructura con cavidades y salientes que permite el acoplamiento de otras proteínas para formar estructuras más complejas como el cabello, o para bloquear el funcionamiento de otras.

Es fácil notar de donde proviene la cantidad de proteínas existentes. Para cierta longitud N, existen 20^N secuencias posibles, siendo 20 el número de aminoácidos. Una estimación gruesa de cuantas estructuras diferentes podrían existir para una determinada secuencia es la siguiente: en cada aminoácido hay que fijar el valor de los ángulos ϕ y ψ; dijimos que existen z valores factibles para ellos, por lo tanto el número de estructuras sería del orden de $(z * z)^N$

Para poder fabricar o *sintetizar* sus proteínas esenciales, cada especie necesita disponer de los veinte aminoácidos en ciertas proporciones. La mayoría de las plantas y microorganismos son capaces de utilizar compuestos inorgánicos para obtener todos los aminoácidos necesarios en su crecimiento, pero los animales necesitan conseguir algunos de los aminoácidos a través de su dieta. A estos aminoácidos se les llama esenciales, y en el ser humano son: lisina, triptófano, valina, histidina, leucina, isoleucina, fenilalanina, treotina, metionina y arginina.

Todos ellos se encuentran en cantidades adecuadas en los alimentos de origen animal ricos en proteínas, y en ciertas combinaciones de proteínas de plantas. Aparte de los aminoácidos de las proteínas, se han encontrado en la naturaleza más de 150

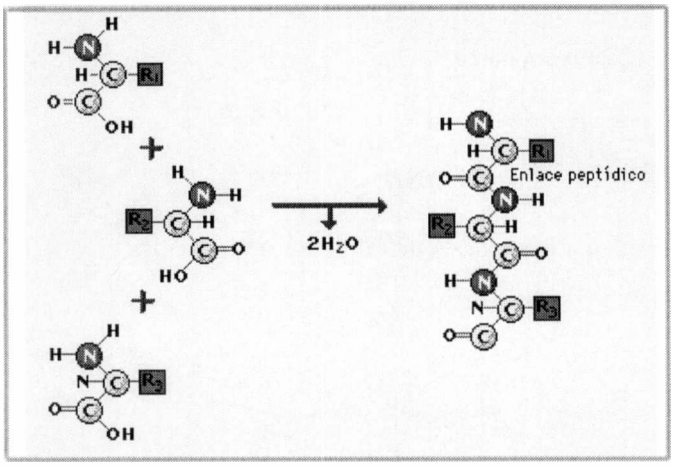

Figura 2.3: Formación de enlaces peptídicos + liberación de agua (H_2O)

tipos diferentes de aminoácidos, incluídos algunos que contienen los grupos amino y carboxilo ligados a átomos de carbono separados. Estos aminoácidos de estructura poco usual se encuentran sobre todo en hongos y plantas superiores.

En la siguiente sección introduciremos los ácidos nucleicos, que son los responsables de la producción de proteínas.

2.3 Los ácidos nucleicos: ADN y ARN

Los ácidos nucleicos son moléculas muy complejas producidas por las células vivas y los virus. Reciben este nombre porque fueron aisladas por primera vez en el núcleo de células vivas. Sin embargo, ciertos ácidos nucleicos no se encuentran en el núcleo de la célula, sino en el citoplasma celular. Los ácidos nucleicos son las sustancias fundamentales de los seres vivos, y se cree que aparecieron hace unos 3.000 millones de años, cuando surgieron en la Tierra las formas de vida más elementales.

Dentro de la célula encontramos dos variedades de ácidos nucleicos: ADN y ARN. Comenzaremos esta sección describiendo el primero de ellos

Cada célula del organismo posee unas pocas moléculas de ADN; cada una de ellas recibe el nombre de *cromosoma* y secciones contiguas del ADN codifican la información necesaria para producir proteínas. Estas secciones son los llamados *genes*. El conjunto de cromosomas dentro de la célula recibe el nombre de *genoma*

Cada molécula de ADN está constituida por dos tiras o bandas de monómeros, llamados *nucleótidos*, dispuestas en forma de doble hélice.

El descubrimiento de esta estructura lo realizaron los biofísicos británicos Francis Crick, Maurice Wilkins y Rosalind Franklin, y el bioquímico estadounidense James Watson. Utilizando una fotografía de una difracción de rayos X de la molécula de ADN obtenida por Wilkins en 1951, Watson y Crick elaboraron un modelo de la molécula de ADN, que fue completado en 1953. Este trabajo, mereció el Premio Nobel de Medicina en 1962.

Cada nucleótido de la cadena está formado por tres unidades: una molécula de azúcar llamada *desoxirribosa*, un *grupo fosfato* y uno de cuatro posibles compuestos

Especie	Cromosomas	Genoma (*bp*)
Bacteriofago λ (virus)	1	$5x10^4$
Escherichia Coli (bacteria)	1	$5x10^6$
Saccharomyces cerevisia (levadura)	32	$1x10^7$
Caenorhabditis elegans (gusano)	12	$1x10^8$
Drosophila melanogaster (mosca de la fruta)	8	$2x10^8$
Homo sapiens (humanos)	46	$3x10^9$

Tabla 2.2: Tamaños de genomas de algunas especies. Notar las diferencias de tamaño.

nitrogenados llamados *bases*: *adenina, guanina, timina y citosina*. De las iniciales de estas bases proviene las letras A, G, T, C que los simbolizan. Ya que la base permite identificar al nucleótido, es válido decir que el ADN es una secuencia de bases.

ADN significa *ácido desoxiribonucleico* y recibe el nombre por la molécula de azúcar que lo compone.

La molécula de desoxirribosa contiene 5 átomos de carbono que se numeran del $1'$ al $5'$. El enlace de los nucleótidos se establece entre el carbono $3'$ del azúcar de una unidad, el grupo fosfato, y el carbono $5'$ del azúcar de la siguiente unidad.

Por convención, se define la orientación del ADN comenzando en el carbono $5'$ del primer nucleótido, terminando en el $3'$ del último. De esta manera, al ver una secuencia de ADN en una base de datos o en un artículo, sabremos que el orden de los nucleótidos es $5' \rightarrow 3'$.

Cómo se mantienen enlazadas las dos "tiras" de ADN en la estructura de doble hélice? Cada base en una de las tiras, se aparea o enlaza con una base en la otra siguiendo una simple receta: la base Adenina siempre se enlaza con la base Timina y la base Citosina siempre con la base Guanina (ver Fig.2.4).

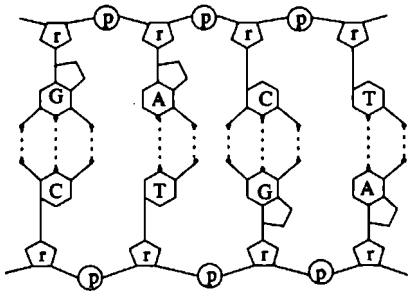

Figura 2.4: Visión esquematica de la estructura del ADN. Figura tomada de [MS96]

Se dice que (A $\Leftrightarrow$ T) y (C $\Leftrightarrow$ G) son bases complementarias. Esto permite definir la longitud de una molécula de ADN en *pares de bases* o *bp*. En la tabla 2.2, tomada de [MS96], se comparan los tamaños de los genomas de varias especies utilizando esta medida.

Es importante aclarar que cada una de las tiras de ADN preserva su orientación y que ambas son opuestas. Es decir, el 5' de una se corresponde con el 3' de la otra. Por esta razón se dice que las dos tiras son *antiparalelas*. Viendo la cadena de ADN

como una lista de caracteres tenemos:

$$5'\ldots TAACGTA\ldots 3'$$
$$3'\ldots ATTGCAT\ldots 5'$$

La consecuencia fundamental de esta estructura es que a partir de una de las tiras, la otra puede inferirse.

Supongamos que tenemos la tira $s = GCCGTAAGT$ y deseamos obtener la otra. El primer paso es hacer el reverso de s, $s' = TGAATGCCG$ y luego reemplazar cada base por su complemento: $\hat{s} = ACTTACGGC$. Este simple mecanismo es el que permite al ADN *autoreplicarse* en la célula.

El otro ácido nucleico que aparece en la célula y que describiremos es el *ARN* o *ácido ribonucleico*.

Las moléculas de ARN son bastante similares a las de ADN salvo las siguientes diferencias: el azúcar es *ribosa* en lugar de *desoxiribosa*, la base Timina es reemplazado por Uracilo ("U" en la simbología), y no forma una estructura de doble hélice como el ADN. A veces, bases complementarias de la misma molécula se acoplan entre sí para formar estructuras tridimensionales muy variadas.

2.4 Síntesis proteica

En esta sección describiremos el proceso de síntesis; el proceso por el cual la célula fabrica proteínas.

Dijimos que el ADN incorpora las instrucciones para producir proteínas y que estas instrucciones son los genes.

En organismos simples, cuyas células no tienen núcleo, el ADN se encuentra "libre" dentro de la célula; mientras que en organismos superiores, el ADN se encuentra en el núcleo de las células. Los primeros, son los llamados *procariotas* mientras que los últimos se denominan *eucariotas*

En general los genes no cubren todo el cromosoma. Existen porciones de la molécula, llamadas *ADN basura* (*junk DNA*), para las cuales no se conoce cual es su función, no codificarían nada. En los procariotas casi todo el cromosoma son genes, pero en los eucariotas se estima que el cromosoma puede tener hasta el 90% de *junk DNA*.

Vimos que una proteína es un polímero formado por monómeros llamados aminoácidos, que determinan su estructura y función. *La secuencia de aminoácidos está a su vez determinada por la secuencia de nucleótidos del ADN.*

Cada secuencia de tres nucleótidos o bases, llamada tripleta, constituye una palabra o codón del código genético, y permite especificar un aminoácido determinado. Así, la tripleta *GCU* (guanina, citosina, uracilo) es el codón correspondiente al aminoácido Alanina, mientras que la *CCA* (citosina, , citosina, adenina) corresponde al aminoácido Prolina (ver Fig. 2.5). Por lo tanto, para codificar una proteína de 100 aminoácidos es necesario un segmento de ADN de 300 nucleótidos.

Si prestamos atención al código genético podemos observar que existen $(4 * 4 * 4 = 64)$ posibles tripletas y solo 20 aminoácidos; esto se debe a que existe más de una forma de codificar un aminoácido y a que algunas tripletas sirven para indicar donde comienza y termina la proteína a fabricar (es decir, permiten detectar el comienzo y finalización de los genes).

PRIMERA LETRA	SEGUNDA LETRA U	SEGUNDA LETRA C	SEGUNDA LETRA A	SEGUNDA LETRA G	TERCERA LETRA
U	Fenilalanina	Serina	Tirosina	Cisteína	U
	Fenilalanina	Serina	Tirosina	Cisteína	C
	Leucina	Serina	Parada	Parada	A
	Leucina	Serina	Parada	Triptófano	G
C	Leucina	Prolina	Histidina	Arginina	U
	Leucina	Prolina	Histidina	Arginina	C
	Leucina	Prolina	Glutamina	Arginina	A
	Leucina	Prolina	Glutamina	Arginina	G
A	Isoleucina	Treonina	Asparagina	Serina	U
	Isoleucina	Treonina	Asparagina	Serina	C
	Isoleucina	Treonina	Lisina	Arginina	A
	(Inicio) Metionina	Treonina	Lisina	Arginina	G
G	Valina	Alanina	Ácido aspártico	Glicina	U
	Valina	Alanina	Ácido aspártico	Glicina	C
	Valina	Alanina	Ácido glutámico	Glicina	A
	Valina	Alanina	Ácido glutámico	Glicina	G

Figura 2.5: El código genético

Otra cosa interesante de ver es que los codones del código están especificados con las bases de ARN (aparece la "U" en lugar de "T"). Más adelante veremos cuál es la razón.

De las dos cadenas de nucleótidos que forman una molécula de ADN, sólo una, la llamada paralela, contiene la información necesaria para la producción de una secuencia de aminoácidos determinada. La otra, llamada antiparalela, ayuda a la replicación.

El proceso de fabricación de proteínas se denomina *síntesis*. En los eucariotas, la síntesis comienza en el núcleo de la célula con la separación de la molécula de ADN en sus dos hebras. En un proceso llamado *transcripción*, una parte de la hebra paralela actúa como plantilla para formar una nueva cadena que se llama ARN mensajero o ARNm. Esta cadena es idéntica a la hebra antiparalela, salvo que la base T es reemplazado por U

Luego el ARNm sale del núcleo celular y se acopla a los ribosomas, unas estructuras celulares especializadas que actúan como centro de control para el proceso de síntesis.

En los ribosomas se inicia el proceso de *traducción*, donde "simplemente" se aplica el código genético.

En la Fig. 2.6 podemos ver este proceso en más detalle e interpretar la formación de la proteína como el resultado del trabajo en una línea de montaje: se "lee" la tripleta corriente en el ARNm, luego el ARN de transferencia (ARNt) "lleva" el aminoácido asociado (según el código genético) al ribosoma y finalmente el aminoácido se "enlaza" con los anteriores para formar la proteína.

Este flujo de información genética en la célula queda reflejado en el llamado *Dogma central de la biología molecular* que se muestra en la Fig. 2.7

2.5 El Problema de Doblado de Proteínas

En las secciones previas, presentamos los conceptos básicos de la biología molecular: ácidos nucleicos, proteínas y proceso de síntesis.

En esta sección veremos con más detalle a las proteínas y definiremos el problema estudiado en esta tesis.

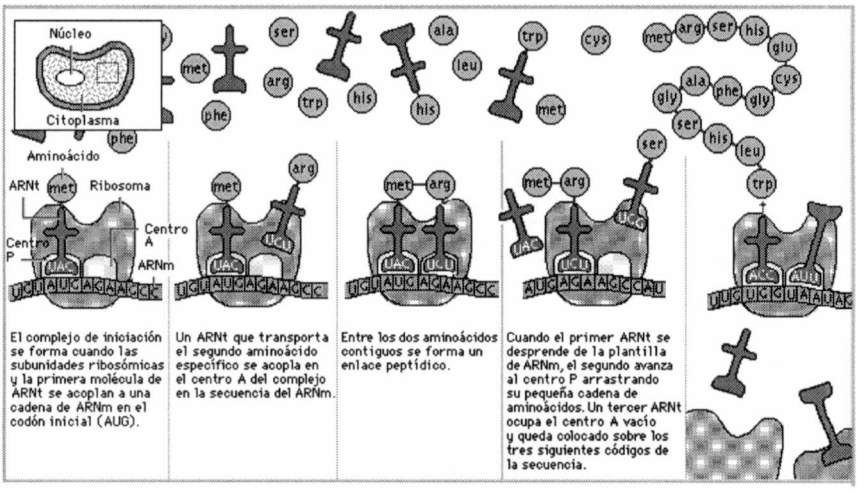

Figura 2.6: Visión esquemática del proceso de síntesis

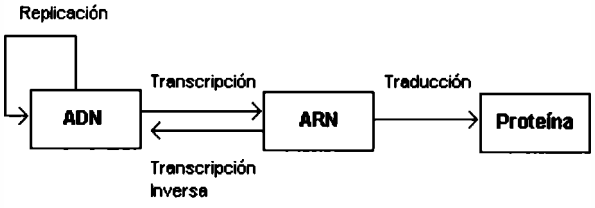

Figura 2.7: Dogma central de la biología molecular

Vimos que el proceso de síntesis sigue la secuencia:

$$Nucleotidos \Rightarrow Tripletas \Rightarrow Aminoacidos \Rightarrow Proteinas$$

De los nucleótidos a los aminoácidos se llega a través del código genético, pero como se llega de los aminoácidos a una proteína funcional?.

Estructuralmente, las proteínas pueden ser analizadas a diferentes escalas.

Se denomina *estructura primaria* de una proteína, a la secuencia de aminoácidos que la componen.

Estos aminoácidos se agrupan en estructuras llamadas $\alpha - helices$, $laminas - \beta$ y *loops*, las cuales reciben el nombre de *estructuras secundarias.*

Estas subestructuras se pliegan o "doblan" en el espacio tridimensional hasta alcanzar una configuración que se conoce como *estructura terciaria* o *estado nativo.* Es esta estructura tridimensional "final" la que determina la funcionalidad biológica de la proteína.

Es aceptado que uno de los elementos que más influye en la determinación de esta estructura, es el efecto hidrofóbico, el cual describiremos brevemente. Los aminoácidos pueden clasificarse en hidrofílicos o hidrofóbicos según sea su comportamiento en un medio acuoso (es decir, si se "sienten a gusto" o no en el agua). En el proceso de plegado los aminoácidos hidrofóbicos tienden a agruparse en el centro de la molécula

formando una especie de coraza interna, mientras que los hidrofílicos tienden a quedar expuestos al solvente. El comportamiento del aminoácido en el medio acuoso responde a las características de su cadena lateral R.

Finalmente, lo que nos interesa saber es:

> **dada la secuencia lineal de aminoácidos,**
> **cuál es la estructura tridimensional correspondiente?**

Esta pregunta es la que define al *Problema de Doblado de Proteínas* o *Protein Folding Problem.*

La respuesta no es un tema menor; a pesar del gran desarrollo de las ciencias involucradas, químicos, biólogos, y físicos no han podido establecer fehacientemente como la Naturaleza realiza este proceso en forma tan veloz y eficaz.

Todos los trabajos relacionados con este tema asumen que la secuencia de aminoácidos es suficiente para determinar, completa y unívocamente, la estructura tridimensional.

Esta hipótesis está relacionada con el siguiente experimento realizado por Anfinsen en 1961 [AHea61]: trabajando *in vitro*, y modificando ciertas condiciones, se logra que una proteína se "desnaturalice" o desdoble (es decir, pierde su funcionalidad). Al restablecer las condiciones, la proteína retoma su forma tridimensional original muy rápidamente [2].

Como resultado se han desarrollado varias hipótesis que tratan de explicar como ocurre este proceso de "plegado" o folding y que describimos a continuación:

- *Hipótesis Termodinámica*: surge a partir de los experimentos de Anfinsen. Según esta hipótesis, el proceso de plegado es guiado por las leyes de la termodinámica. La proteína, como sistema biológico, tiende a estabilizarse utilizando el menor esfuerzo posible. Esta conformación estable o de "mínima energía libre" se denomina estado nativo o estado funcional. Bajo esta hipótesis, el problema se puede plantear en términos de la minimización de alguna función de energía adecuada sujeta a restricciones. Nada se establece acerca de como este mínimo valor de energía es alcanzado.

- *Hipótesis "Dinámica"*: según Levinthal [DTM69], es imposible que una proteína, dentro de los tiempos biológicos del proceso, alcance su estado nativo a partir de una búsqueda exhaustiva en el espacio de conformaciones posibles [3].

  Según esta hipótesis, el proceso de plegado es similar a una reacción química; de alguna manera las proteínas siguen un *camino específico* que las "guía" desde un estado no funcional hacia el estado nativo pasando por estados intermedios semi estables: $\mathcal{U} \Rightarrow \mathcal{I}_1 \Rightarrow \mathcal{I}_2 \ldots \Rightarrow \mathcal{I}_k \Rightarrow \mathcal{N}$

  Una hipótesis más general, denominada de "túneles de plegado" o *folding funnels*, sugiere que en cada paso intermedio la proteína puede "elegir" entre un "conjunto" restringido de conformaciones: $\mathcal{U} \Rightarrow \mathcal{S}_1 \Rightarrow \mathcal{S}_2 \ldots \Rightarrow \mathcal{S}_k \Rightarrow \mathcal{N}$ donde cada $\mathcal{S}_i = \{\mathcal{I}_i, \ldots, \mathcal{I}_t\}$

  Las dos formas proveen una explicación a la velocidad conque se realiza el proceso de plegado, pero implican que el proceso es "camino dependiente".

[2] Se estima que las proteínas se doblan en 10^{-2} segundos

[3] Una búsqueda exhaustiva estaría en el orden de los 10^{42} años

- *Lenguaje Oculto*: establece que este proceso está codificado en la secuencia de aminoácidos en términos de algún lenguaje desconocido. Esto implicaría que el proceso de folding puede modelizarse como un proceso mecánico o algorítmico; aparentemente el procesamiento no podría ser planteado como secuencial o símbolo por símbolo, sino como un proceso paralelo que opera simultaneamente utilizando partes remotas de la secuencia. De ser válida esta hipótesis, su verificación permitiría establecer un "2^{do} código genético" que iría desde los aminoácidos a la estructura.

En [GG98] se muestra que las hipótesis dinámica y la termodinámica no son mutuamente excluyentes partiendo de la idea que el estado nativo coincide con el estado de mínima energía libre como consecuencia del proceso evolutivo y no de cuestiones termodinámicas.

En ocasiones se cree que *PF* es un problema de predicción de estructuras; sin embargo, el *PF* también involucra las cuestiones dinámicas existentes en el proceso de plegado. Afortunadamente el conocimiento obtenido por predicción de estructuras brinda información acerca de la dinámica del folding y viceversa.

En la sección siguiente veremos los métodos que se emplean actualmente para obtener las estructuras terciarias y para indagar en el proceso de plegado.

2.5.1 Problemas en la determinación de la estructura

La determinación de la secuencia de aminoácidos que componen una proteína se realiza a partir del conocimiento de la secuencia de ADN que la codifica. El proceso de obtener la secuencia de nucleótidos que forman una cadena de ADN se denomina *secuenciación*.

Los métodos utilizados para el secuenciamiento permiten trabajar con moléculas de ADN del orden de los 700 *bp*; recordando que el cromosoma humano tiene alrededor de 10^8 *bp* es claro que estamos frente a un problema de escala realmente importante.

Esta brecha está en el centro de otros problemas de la Biología Computacional como el "mapeo físico de cromosomas" y el "reensamblado de fragmentos".

Con el simple cálculo mostrado en 2.2 quedó claro que la determinación de la estructura tridimensional no puede realizarse a partir de la enumeración exhaustiva de las conformaciones posibles buscando por la mejor (también es un problema decidir qué medir para decir cuál es la mejor) En la actualidad, para determinar la estructura terciaria de una proteína dada se pueden seguir dos caminos: utilizar métodos experimentales o trabajar en base a similaridades con proteínas conocidas.

Dentro de los métodos experimentales, se encuentran la cristalografía de rayos X y la resonancia magnética nuclear(NMR). El primero de ellos, permite obtener abundante información estructural pero la determinación de las condiciones de cristalización es muy complicada. Además el proceso de cristalización puede producir deformaciones en la estructura, por lo cual, la información debe ser analizada muy cuidadosamente. La NMR permite analizar las proteínas en solución pero provee información solamente sobre algunos tipos de átomos. Se pierde detalle estructural y se deben asumir elementos como la geometría del *backbone*.

Como generalmente ocurre con este tipo de métodos, son caros, las condiciones de experimentación deben ser cuidadosamente establecidas y no siempre es posible aplicarlos.

Como consecuencia de esto, hoy en día existe una importante diferencia entre la cantidad de información disponible de secuencias de proteínas y de sus respectivas

configuraciones espaciales.

Toda esta información se almacena en grandes bases de datos mantenidas por fundaciones y grandes consorcios de laboratorios. Estas bases de datos han contribuido en el desarrollo teórico-práctico del área de las VLDB (*very large databases*).

Entre las bases de datos que mantienen secuencias de ADN, podemos citar **GenBank** (`http:\\www.ncbi.nlm.nih.gov\`) y **EMBL** (`http:\\www.embl-heidelberg.de\`). Un registro en GenBank de la bacteria *Alcaligenes eutrophus* se muestra en la Fig. 2.10

Por otro lado también existen bases de datos de proteínas, donde cada registro de las mismas contiene la secuencia y la estructura tridimensional dando, para cada átomo de cada aminoácido, las coordenadas en $3D$ con una precisión de tres decimales.

Entre ellas podemos citar: **Protein DataBank**(`http:\\www.pdb.bnl.gov\`) y **Protein Identification Resource** (`http:\\www.gdb.org\`)

En la fig 2.11 se muestra la descripción de una proteína de una planta. Este archivo pertenece a la base Protein DataBank y se muestra reducido. Existen programas de visualización que permiten mostrar esta información graficamente. En las Fig. 2.8 y 2.9, pueden verse el *backbone*, las estructuras secundarias y un modelo *filling-space* correspondientes a la proteína anterior.

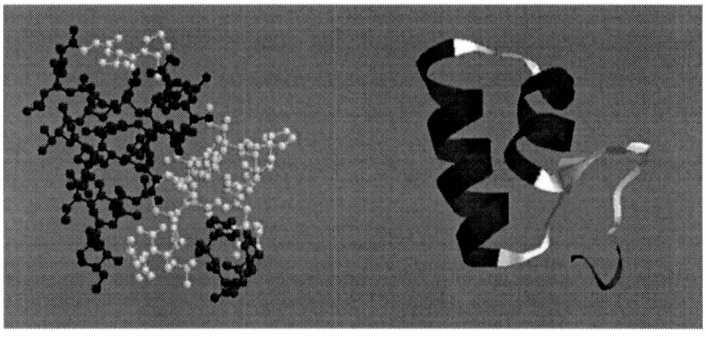

Figura 2.8: Modelo simplificado de la estructura tridimensional y representación esquemática de las estructuras secundarias de la proteína *Crambin*

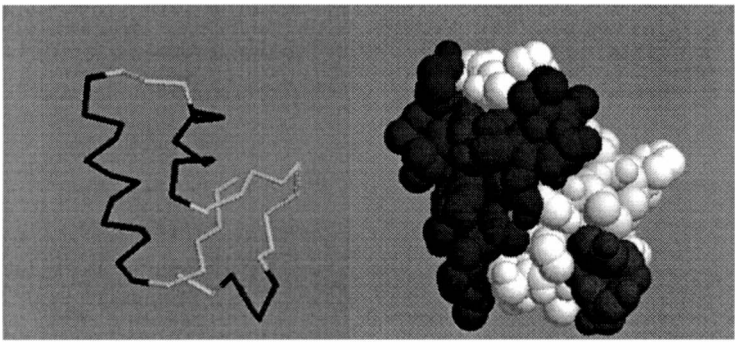

Figura 2.9: Estructura del backbone y un modelo *filling space* de la proteína crambin

La otra forma de determinar la estructura, u obtener algunos indicios de la misma, es utilizar técnicas de modelización comparativas o por homología. Dada una secuen-

cia de aminoácidos, se utilizan secuencias "similares" con estructuras conocidas, para determinar la estructura asociada. Este enfoque es posible ya que un pequeño cambio en la secuencia resulta, usualmente, en un cambio reducido en la estructura tridimensional.

Estos métodos constan basicamente de 3 pasos. En primer término, se realiza una búsqueda en las bases de datos previamente citadas para obtener secuencias "similares" con estructura conocida. Luego se realiza un alineamiento de las secuencias y se determinan las estructuras que serán utilizadas como modelos. Finalmente, se evalúa la secuencia sobre los modelos de acuerdo a varios criterios hasta que alguno de ellos brinde resultados satisfactorios. Si para una proteína no existen secuencias similares, este método no puede ser utilizado para determinar su estructura.

Es en este tipo de modelización y en las consultas a las bases de datos donde aparecen los algoritmos de comparación y alineamiento de secuencias. Estos algoritmos están en constante desarrollo ya que las bases de datos están en continua expansión y es creciente la necesidad de trabajar con secuencias cada vez más largas.

2.6 Conclusiones

En este capítulo hemos presentado los conceptos y elementos fundamentales de la biología molecular. Describimos los ácidos nucleicos, las proteínas y su proceso de fabricación, y definimos el problema que nos incumbe: *El problema de Doblado de Proteínas*.

Vimos que es un problema abierto, planteamos las hipótesis que intentan explicarlo y reseñamos los métodos utilizados actualmente para obtener las estructuras tridimensionales.

Dado lo "ineficientes" que son estos métodos, la cantidad de factores intervinientes en el proceso de plegado y la ausencia de explicaciones contundentes acerca del funcionamiento del mismo, se hace imprescindible la utilización de modelos de proteínas y simulaciones en computadora para poder obtener la información y/o los indicios que permitan, en último término, realizar predicciones *ab initio*; es decir poder determinar la estructura terciaria *sólo* a partir de la secuencia de aminoácidos.

```
LOCUS       AEISAE01   1112 bp    DNA          BCT       30-JUN-1993
DEFINITION  A.eutrophus insertion sequence IS1086.
ACCESSION   X58441 S51074
KEYWORDS    insertion sequence; insertion sequence IS-AE-01.
SOURCE      Alcaligenes eutrophus.
  ORGANISM  Alcaligenes eutrophus
            Eubacteria; Proteobacteria; beta subdivision; Alcaligenaceae;
            Alcaligenes.
REFERENCE   1  (bases 1 to 1112)
  AUTHORS   Dong,Q.H.
  TITLE     Direct Submission
  JOURNAL Submitted (14-MAR-1991) to the EMBL/GenBank/DDBJ databases.
 Q.H. Dong, Lab of Gene and Biotechnology, Centre d'Etude Nucleare,
        200 Boeretang, 2400 Mol, Belgium
COMMENT     NCBI gi: 48875



FEATURES             Location/Qualifiers
 source          1..1112
                 /organism="Alcaligenes eutrophus"
                 /strain="ATCC 43123"
                 /sub_strain="CH34"
 CDS             56..1075
                /gene="ORF 1"
                /note="NCBI gi: 48876"
                /codon_start=1
                /translation="MTRTKYQQLQPEERMRIEIWKAEDVSLRAMARRLGRAPSTLMRE
                LRRNATARGGYGAMSAQACRTQRLKASRPVAKLAPDGVLWGVVRHFLDQKWSPQEISG
                TLKRAFPDQPDLNVSHETIYNAIYAYPRGELRRQLIACLRQARTKRLPRSRGTDRRGQ
                IPDMVSIHVRPPEVNDRLMPGHWEGDLIKGAGNQSAVGVLVERMSRAVLLVKMPDATA
                ASALAGFTGKLQSLVAPLRQTLTYDQGREMARHAELSAATGVRVYFCDPHSPWQRGTC
                ENTNGLLRQYLPKGTDLSVYSQEELDAIADSLNGRPRKTLNWHSPLQVLAQVLANPTD
                RLPVQ"
BASE COUNT      220 a    385 c    326 g    181 t
ORIGIN
1 cctggcggcc tcaaatctga agtgcaacac cttgccattc ggtaaggtgt ggtgaatgac
61 gagaacgaag taccaacaac tacaacccga agaacgcatg cgcatcgaga tttggaaggc
121 agaagatgtc agcctgcggg ccatggcccg caggcttggc cgcgcgcctt cgacactgat
181 gcgtgagctt cgccgcaatg ccaccgcccg tggcggctat ggcgcaatga gcgcacaagc
241 ctgccgtacg caacgcctca aggccagccg tccggtcgct aagctcgctc ctgatggcgt
301 attgtggggc gtggtgcgcc acttcctcga tcagaagtgg tcgccccagg aaatctccgg
361 tacgctcaag cgggcctttc ctgaccaacc cgacctcaac gtgtcccacg agaccatcta
421 caacgccatc tacgcatacc cgcggggtga gctgcgccgc cagctcatcg cctgcttgcg
481 acaggcccga accaagcgtc tgccgcgctc gcgtggaacc gaccggcgcg gccagattcc
541 cgacatggtc agcattcacg tgcgcccgcc cgaggtgaac gacaggctga tgcccgggca
601 ctgggagggc gacctcatca agggggcggg caaccaatcc gccgtaggcg tgctggttga
661 gcgcatgagc cgcgccgtac tgctggtcaa gatgccagac gccaccgctg catcggctct
721 ggccggcttt accgggaagc tgcaatccct ggtggcgccg ctgcgccaga ccctgaccta
781 cgaccagggt cgagagatgg ccaggcacgc cgaactgagc gccgccactg gcgtgcgagt
841 gtacttctgc gacccgcaca gcccctggca gcgcggcacc tgtgagaaca ccaatggtct
901 gctgcgccag tacttaccca agggcaccga cctgtccgtc tactcgcagg aggaacttga
961 cgccatcgcc gacagcctca acggccgccc gcgcaaaacc ctgaactggc actccccgct
1021 acaagtcctc gctcaggttc tggccaatcc cacggaccgg cttcctgttc aataaccaag
1081 gggtgttgcg cttcgcactt gaaaccgccc ct
```

Figura 2.10: Registro de GenBank donde aparece la secuencia de nucleótidos del ADN de una bacteria. En el campo "translation" se muestra la lista de aminoácidos asociados

```
HEADER    PLANT SEED PROTEIN                          30-APR-81
COMPND    CRAMBIN
SOURCE    ABYSSINIAN CABBAGE (CRAMBE ABYSSINICA) SEED
AUTHOR    W.A.HENDRICKSON,M.M.TEETER
REVDAT   5   16-APR-87 D   1       HEADER
......
REMARK   1 REFERENCE 3
REMARK   1  AUTH   M.M.TEETER,W.A.HENDRICKSON
REMARK   1  TITL   HIGHLY ORDERED CRYSTALS OF THE PLANT SEED PROTEIN
REMARK   1  TITL 2 CRAMBIN
REMARK   1  REF    J.MOL.BIOL.                  V.  127  219 1979
REMARK   1  REFN   ASTM JMOBAK  UK ISSN 0022-2836                070
......
SEQRES   1     46  THR THR CYS CYS PRO SER ILE VAL ALA ARG SER ASN PHE
SEQRES   2     46  ASN VAL CYS ARG LEU PRO GLY THR PRO GLU ALA ILE CYS
SEQRES   3     46  ALA THR TYR THR GLY CYS ILE ILE ILE PRO GLY ALA THR
SEQRES   4     46  CYS PRO GLY ASP TYR ALA ASN
HELIX    1  H1 ILE      7  PRO     19  1 3/10 CONFORMATION RES 17,19
HELIX    2  H2 GLU     23  THR     30  1 DISTORTED 3/10 AT RES 30
SHEET    1  S1 2 THR     1  CYS     4  0                                  A
SHEET    2  S1 2 CYS    32  ILE    35 -1
TURN     1  T1 PRO    41  TYR    44
SSBOND   1 CYS       3    CYS     40
SSBOND   2 CYS       4    CYS     32
SSBOND   3 CYS      16    CYS     26
CRYST1   40.960   18.650   22.520  90.00  90.77  90.00 P 21          2
.....
ATOM      1  N   THR     1      17.047  14.099   3.625  1.00 13.79
ATOM      2  CA  THR     1      16.967  12.784   4.338  1.00 10.80
ATOM      3  C   THR     1      15.685  12.755   5.133  1.00  9.19
ATOM      4  O   THR     1      15.268  13.825   5.594  1.00  9.85
ATOM      5  CB  THR     1      18.170  12.703   5.337  1.00 13.02
ATOM      6  OG1 THR     1      19.334  12.829   4.463  1.00 15.06
ATOM      7  CG2 THR     1      18.150  11.546   6.304  1.00 14.23
.....
ATOM    238  N   ILE    34      11.490  15.773   7.038  1.00  5.52
ATOM    239  CA  ILE    34      12.552  15.877   6.036  1.00  6.82
ATOM    240  C   ILE    34      13.590  16.917   6.560  1.00  6.92
ATOM    241  O   ILE    34      13.168  18.006   6.945  1.00  9.22
ATOM    242  CB  ILE    34      11.987  16.360   4.681  1.00  8.11
ATOM    243  CG1 ILE    34      10.914  15.338   4.163  1.00  9.59
ATOM    244  CG2 ILE    34      13.131  16.517   3.629  1.00  9.73
ATOM    245  CD1 ILE    34      10.151  16.024   2.938  1.00 13.41
.....
ATOM    314  N   ALA    45      14.342   8.640  15.422  1.00  4.76
ATOM    315  CA  ALA    45      15.445   7.667  15.246  1.00  5.89
ATOM    316  C   ALA    45      15.171   6.533  14.280  1.00  6.67
ATOM    317  O   ALA    45      16.093   5.705  14.039  1.00  7.56
ATOM    318  CB  ALA    45      15.680   7.099  16.682  1.00  6.82
.....
END
```

Figura 2.11: Proteína de una planta, archivo extraído de Protein DataBank. Pueden verse, además de la secuencia de aminoácidos (campo SEQRES), las posiciones de los átomos que los componen.

Capítulo 3

Introducción a las técnicas evolutivas

Este capítulo consta de tres secciones. En la primera, se repasan algunas definiciones básicas y se desarrollan los conceptos fundamentales de complejidad computacional.

La segunda parte contiene una descripción de las técnicas heurísticas, su clasificación y algunas sugerencias respecto a cuando deben ser aplicadas.

Dentro de las heurísticas se destacan las técnicas evolutivas, para las cuales se desarrolla una sección especial.

Dada la amplitud de los temas, durante el desarrollo del capítulo se ha puesto énfasis en presentar solamente los conceptos fundamentales. Se ha citado abundante bibliografía para que el lector pueda profundizar en los temas de su interés.

3.1 Conceptos de Complejidad Computacional

En esta sección [1] se hará un repaso de algunos conceptos fundamentales de complejidad computacional en particular y optimización combinatoria en general.

Informalmente, los problemas de optimización combinatoria son aquellos para los cuales el espacio de soluciones factibles es finito pero muy grande; donde cada solución factible es compatible con las restricciones específicas de cada problema pero no necesariamente óptimas.

Se repasaran, entre otros, los conceptos de Máquina de Turing [2], problemas de decisión - optimización, NP-Completitud, etc.

3.1.1 Máquinas de Turing

Una MT consiste de:

- Una cinta doblemente infinita dividida en celdas, que pueden considerarse numeradas como $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$.
- Un alfabeto $\Sigma = \{0, 1, \lambda\}$.

[1] Agradecemos a Natalio Krasnogor por haber cedido parte del material de esta sección

[2] MT en adelante

- Una cabeza sobre la cinta capaz de leer un caracter simple de la cinta o bien escribir uno, y además puede moverse una celda en alguna de ambas direcciones.

- Una lista finita de estados, tal que en cada instante la máquina esta en uno y solo uno de ellos. Los estados posibles de MT son, ante todo, los estados regulares $q_1, \ldots, q_s$, y ademas, tres especiales $q_0 = START$, $q_Y = FINAL_{ACCEPTING}$, $q_N = FINAL_{REJECTING}$.

- Un programa que dirije a MT a través de los pasos de una computación en particular, toma un estado y un símbolo y retorna un nuevo estado, un nuevo símbolo y la especificación del movimiento de la cabeza lectora.

Formalmente definimos MT como:
$T = (k, \Sigma, \Gamma, \alpha, \beta, \gamma)$ donde $k \geq 1$ es un número natural (arriba consideramos $k = 1$), que especifica la cantidad de cintas de MT. Σ y Γ conjuntos finitos. Σ es el conjunto de los símbolos disponibles en la cinta, con $\lambda \in \Sigma$. Γ es el conjunto de estados, $START, STOP \in \Gamma$, además
$\alpha \mid \Gamma \times \Sigma^k \mapsto \Gamma$,
$\beta \mid \Gamma \times \Sigma^k \mapsto \Sigma^k$,
$\gamma \mid \Gamma \times \Sigma^k \mapsto \{-1, 0, 1\}^k$
son mapeos arbitrarios. Donde 'α' genera un nuevo estado, 'β' el símbolo a ser impreso en la cinta y 'γ' determina cuanto es el movimiento de la cinta. 'Σ' esta formado por $\{0, 1, \lambda\}$.

En la definición de arriba hemos considerado que los alfabetos de entrada y salida son el mismo, siendo sencillo formalizar el caso en que esto no ocurre así. Si bien hemos definido algunos estados como "especiales" los mismos no son necesarios si MT es *transductora* pero si en el caso de ser *reconocedora*.

Una *Máquina de Turing Universal* U, es una MT especial que recibe como entrada la descripción de una MT T y un conjunto de datos x y simula la ejecución de T sobre x.

Si T para, también lo hace U de acuerdo al estado final de T. En caso contrario, o si la entrada no es una descripción válida de una MT, se asume que U no para.

Las definiciones de arriba son con el solo objeto de introducir los conceptos al lector, y han sido simplificadas para nuestro propósito. Para un tratamiento clásico del tema ver [BS86, LG94]

3.1.2 Problemas de Decisión - Optimización

Un problema de decisión, PD en adelante, es un problema donde la salida es solo *SI* o *NO*. Por ejemplo, dado un grafo $G = (V, E)$ un problema de decisión asociado al mismo es verificar si puede ser coloreado con k colores, o si existe un tour T con $longitud(T) \leq L$ donde toda arista de T pertenece a E. Todo PD puede ser asociado con un problema de optimización, PO en lo que sigue. Para los ejemplos de arriba: Cual es el menor número cromático de $G = (V, E)$? , o cual es el tour T más corto para G?

Si podemos resolver "rápido" un PD, entonces, con un poco más de esfuerzo podremos resolver "rápido" el PO asociado. Por ejemplo, supongamos que contamos con un algoritmo polinomial para verificar si un grafo puede ser coloreado con k colores. Entonces, podremos extenderlo para buscar el menor k de la siguiente manera:

1. Se inicializa k, esto es, $k =\mid V \mid$. Así, hemos asignado a cada vértice de G un color distinto, lo que genera el coloreo trivial.
2. Verificamos en tiempo polinómico si G puede ser $k - coloreado$.
3. En caso afirmativo, guardamos $k(old_k = k)$, asignamos a k el valor $\lfloor k/2 \rfloor$, y volvemos al punto 2.
4. Caso contrario, si $k > old_k$ obtuvimos lo buscado y seguimos en 5, sino hacemos $k = k + \lfloor \frac{old_k - k}{2} \rfloor$, y vamos al punto 2.
5. Devolver old_k, siendo este el menor número cromático de G.
6. Fin.

El algoritmo de arriba trabaja en tiempo $O(log(\mid V \mid) * q(\mid G \mid))$, donde $q(\mid G \mid)$ es un polinomio en el tamaño del grafo. De esta manera fuimos capaces de resolver un PO con un poco más de tiempo de cómputo, $O(log(\mid V \mid))$ veces $q(\mid G \mid)$, a partir de la solución al PD asociado [3].

Todo PD tiene una respuesta *SI* o *NO*, de tal forma que se puede considerar que un PD consiste en decidir si una dada palabra x sobre Σ pertenece a L. L es el conjunto de todas las palabras para las cuales la respuesta es *SI*. En el contexto de este capítulo, la palabra x es solo una "buena" codificación de alguna instancia I de un problema Π. Para el PD asociado con Π, la respuesta para I será *SI* cuando su codificación x pertenezca al lenguaje L.

3.1.3 Nociones Sobre las Clases P y NP

Un PD Π pertenece a P(polinomial) si existe un algoritmo A y una constante c tales que: $\forall I \in \Pi$, A es $O(B^c)$ donde B es el número de bits necesarios para representar la instancia I, $B =\mid x \mid$. En resumen, los PDs que pertenecen a P son "fáciles" de resolver.

Por otro lado, un PD pertenece a la clase NP si existe un algoritmo A que verifica:

1. Asociado con cada palabra del lenguaje Q existe un certificado $C(I)$, y cuando $(I, C(I))$ es dado como entrada para A este reconoce que $I \in Q$.
2. Si I no esta en Q entonces no existe ningún $C(I)$, de forma que A no reconoce I.
3. A es un algoritmo de tiempo polinomial.

Podemos ver que P $\subset$ NP ya que $\forall \Pi \in P, \forall I \in \Pi, C(I) \neq \emptyset$. Una definición mas rigurosa puede encontrarse en [BS86].

Dado un problema Π_1, podemos decir que pertenece a la clase de los problemas NP si puede ser resuelto por un algoritmo **no determinístico** que realiza su trabajo en a lo sumo tiempo polinomial en el tamaño del problema. Es obvio que la clase P de los algoritmos polinomiales determinísticos esta incluida en NP, P $\subseteq$ NP. Seguramente, si la inclusión es propia es una de las preguntas más importantes de la teoría de la complejidad ($P = NP?$). Diremos que Π_1 es transformable polinomialmente a otro problema Π_2, $\Pi_1 \prec_p \Pi_2$, si existe una función f que mapea las instancias de Π_1 en instancias de Π_2, y se verifica lo siguiente:

[3] Desafortunadamente tal solución al coloreo de grafos no existe

- f es polinomial y determinística.
- Una solución a la instancia $f(I)$ de Π_2 puede ser convertida en una solución para la instancia I de $\Pi_1 \forall I$.

En otras palabras, esto nos dice que si contamos con el algoritmo mas rápido para resolver Π_2, entonces el tiempo necesario para alcanzar una solución para Π_1 es como mucho el tiempo necesario para solucionar Π_2 mas un término polinómico. Tenemos entonces que **complejidad**(Π_1) $\leq$ **complejidad**(Π_2) + **Polinomio**. En lo que sigue se entenderá que las transformaciones serán polinomiales y se usará $\prec$ en vez de $\prec_p$.

Para concluir, enunciamos el siguiente teorema:

Teorema: Si $\Pi \in NP$, entonces existe un polinomio p tal que Π puede ser resuelto por un algoritmo determinístico en tiempo $O(2^{p(n)})$.

Es decir, todo lo que podemos resolver con un algoritmo nodeterminístico también puede ser resuelto con uno determinístico, pero con "más trabajo".

El lema que aparece a continuación es solo a fin de completar algunos conceptos mencionados con anterioridad.

Lema: Si $L_1 \prec_p L_2$, entonces $L_2 \in P \Rightarrow L_1 \in P$ (de la misma manera, $L_1 \notin P \Rightarrow L_2 \notin P$).

3.1.4 Completitud en NP

Un problema Π es NP-Hard si las siguientes condiciones se satisfacen:

- $\Pi\prime \prec \Pi, \forall \Pi\prime \in NP$.
- $\Pi \in P \Rightarrow P = NP$.

Si existe un algoritmo determinístico polinomial para Π entonces existe uno para cualquier otro problema en NP. Ahora podemos definir que es un problema NP-Completo:

Un problema Π es NP-Completo si pertenece a la clase de los NP y además es NP-Hard. Existen algunas relaciones importantes entre P y NP [GJ79b]. Como ya se dijo antes, un PD Π es NP-Completo si pertenece a NP y todo problema en NP es polinomialmente reducible a él. En otras palabras, un problema que es "mayor" que cualquier otro problema NP, esto es un problema al cual cualquier otro en NP puede ser reducido, se llama NP-Hard. Si un problema NP-Hard es además NP se dice NP-Completo. Entonces, todos los problemas NP-Completos son igualmente difíciles de resolver pues son reducibles entre sí.

Para ver una lista extensa de problemas NP-Completos con sus clases de aproximación el lector puede referirse a [CK].

3.2 Heurísticas

Una gran cantidad de problemas que surgen en la industria, economía, logística, etc., son NP-Completos. Como ya se dijo antes, es imposible resolver este tipo de problemas eficientemente [4] a menos que $P = NP$, lo que difícilmente sea cierto.

[4] Consideramos eficientes a los algoritmos polinómicos en el tamaño de su entrada

Sin embargo, aún así debemos encontrar soluciones a estos problemas. La teoría de la complejidad nos dice que no debemos esperar resolver este tipo de problemas con un algoritmo polinomial en el tamaño de la entrada en forma óptima o exacta. Entonces, mientras sea necesario resolverlos, deberemos utilizar técnicas especiales para encontrar la solución.

Dentro de estas técnicas especiales están las técnicas heurísticas, a las que se define como "procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas díficiles, de un modo fácil y rápido" [ZE81].

Otra definición posible establece que una heurística es un conjunto bien definido de pasos para identificar rapidamente una "buena" solución para un problema determinado.

En general, se asume que las soluciones son *factibles*, es decir, satisfacen todas las restricciones del problema.

3.2.1 Factores que justifican su utilización

Varios factores justifican la utilización de las heurísticas. Su aplicación es factible cuando:

- no existe un método exacto de resolución o éste requiere mucho tiempo de cálculo o memoria.
- no se necesita la solución óptima; a veces solo es necesario proveer una solución mejor que la disponible.
- los datos son poco fiables entonces no tiene sentido buscar la solución "exacta".
- existan limitaciones de tiempo, espacio (para almacenamiento de datos), etc., que obliguen al empleo de métodos de rápida respuesta, aun a costa de la precisión.
- permitan obtener buenas soluciones que sirvan como punto de partida para la ejecución de algoritmos exactos.

3.2.2 Tipos de Heurísticas

Las técnicas heurísticas permiten explorar el espacio de soluciones de un problema. Según el modo en que se realice esta exploración y se construyan las soluciones, las heurísticas pueden dividirse, según [SVdW80], en:

- *Métodos constructivos:* se añaden componentes individuales a la solución hasta que se obtiene una solución factible. El más popular de estos métodos lo constituyen los algoritmos "golosos" (*greedy*), los cuales, en cada paso de iteración, eligen el mejor movimiento posible.
- *Métodos de descomposición (Divide and Conquer):* se intenta dividir el problema en subproblemas más pequeños, siendo la salida de uno, la entrada del siguiente, de forma que al resolverlos todos se obtiene una solución para el problema global.
- *Métodos de reducción:* intentan identificar alguna característica que presumiblemente deba poseer la solución óptima y de ese modo simplificar el problema. Por ejemplo, se puede detectar que una variable debe valer siempre 1, entonces se la puede fijar como constante y eliminar la variable.

- *Manipulación del modelo:* estas heurísticas modifican la estructura del modelo con el fin de hacerlo más sencillo de resolver, deduciendo, a partir de su solución, la solución del problema original. Por ejemplo, se podrían linealizar funciones, agregar o eliminar restricciones, etc.

- *Métodos de búsqueda por entornos:* dada una solución s se define su "entorno" $N(s)$ como el conjunto de soluciones que pueden alcanzarse aplicando a s alguna modificación u operación elemental. La idea básica del método es, a partir de una solución s, pasar a una solución $s' \in N(s)$ que sea mejor en algún sentido. Luego, se repite el procedimiento partiendo desde s' y así sucesivamente hasta que no sea posible encontrar una mejor solución o algún criterio de parada se haya cumplido. La solución final será la mejor de todas las soluciones visitadas. *Simulated Annealing, Método de Monte Carlo, Tabu Search*, etc. son métodos que pertenecen a esta categoría.

- *Técnicas Evolutivas:* la idea básica de estas técnicas, es construir soluciones a partir de soluciones. Para ello mantienen un conjunto de potenciales soluciones y mediante la aplicación de operadores especializados, las mejores son combinadas y modificadas para obtener nuevas soluciones.

En general se utilizan combinaciones de estas técnicas y es común encontrar en la literatura palabras como *meta-heurística* para referirse a heurísticas iterativas que incorporan algún procedimiento de este tipo en su ejecución.

En la sección siguiente, abordaremos con más profundidad las técnicas evolutivas y describiremos dos de ellas: los algoritmos genéticos y la programación genética. Sugerimos al lector interesado la consulta de [DGG+96] y las referencias que allí aparecen para una introducción a los métodos restantes.

3.3 Técnicas Evolutivas y Algoritmos Genéticos

En los últimos 30 años ha existido un creciente interés en el desarrollo de técnicas basadas en los principios de evolución y herencia, para resolver problemas complejos.

Como ya se dijo, estas técnicas mantienen un conjunto de potenciales soluciones, tienen algún proceso de selección basado en la calidad de las mismas, e incorporan operadores que permiten modificar o generar nuevas soluciones.

Ejemplos de estas técnicas son: Algoritmos Genéticos, Algoritmos Meméticos, Genetic Programming, Evolution Strategies, etc.

En esta sección describiremos los Algoritmos Genéticos [5] ya que son la base de este trabajo y mencionaremos brevemente a la Programación Genética [6].

3.3.1 Conceptos Generales

Los algoritmos genéticos son una familia de modelos computacionales inspirados en los mecanismos de herencia y evolución natural (*supervivencia del más apto*).

En los organismos biológicos, la información hereditaria es pasada a través de los cromosomas. Varios organismos se agrupan formando una población, y aquellos que mejor se adaptan son los que más probabilidades tienen de sobrevivir y reproducirse. Algunos de los supervivientes son seleccionados por la naturaleza para ser cruzados y

[5] AG de ahora en más

[6] Una descripción de los métodos restantes puede encontrarse en [Mic96]

así producir una nueva generación de organismos. Esporádicamente, los genes de un cromosoma pueden sufrir ligeros cambios (mutaciones).

Mediante una imitación de este mecanismo, los algoritmos genéticos exploran el espacio de soluciones asociado a un determinado problema.

Las areas de aplicación donde los AG resultan útiles son realmente muy variadas. Entre ellas podemos citar la investigación operativa, el diseño de circuitos electrónicos, la biología molecular y la fisicoquímica, búsquedas en bases de datos y hasta composición musical [DGG+96, Esh95].

Dada la diversidad de enfoques, variantes y elementos que existen en el tema de algoritmos genéticos, en la sección siguiente sólo describiremos los conceptos básicos y los ejemplificaremos con problemas prácticos.

En la Fig. 3.1 se presenta el esquema básico de un *AG*. En cada iteración t, se mantiene una *población* de *individuos* $P(\mathrm{t}) = \{x_1^t, \ldots, x_n^t\}$ donde cada individuo representa una potencial solución del problema a resolver. Cada solución x_i^t se evalúa para obtener cierta medida de su calidad o *fitness*. Luego, se genera una nueva población $(P(t+1))$ tomando como base a los mejores individuos; algunos de los cuales sufren transformaciones a partir de la aplicación de operadores "genéticos". Típicamente, estos operadores son *cruzamiento* o *crossover* (nuevos individuos son generados a partir de la combinación de dos o más individuos), y *mutación* (nuevos individuos son generados a partir de pequeños cambios en un individuo).

```
Begin
  t = 0
  inicializar P(t)
  evaluar P(t)
  mientras no se cumpla criterio de finalizacion
       t = t + 1
       Seleccionar P(t) desde P(t-1)
       Hacer-Cruzamientos(P(t))
       Aplicar-Mutaciones(P(t))
  Reportar mejor solucion encontrada
End
```

Figura 3.1: La estructura básica de un Algoritmo Genético

Varios elementos son necesarios a la hora de aplicar un *AG* sobre un problema particular:

1.- una representación para las potenciales soluciones (individuos)
2.- un mecanismo para crear la población inicial
3.- una función de evaluación para medir la calidad de las soluciones
4.- operadores genéticos para alterar la composición de los hijos
5.- valores para los parámetros del AG (tamaño de la población, probabilidades de aplicación de los operadores, etc.)

Planteamos que un individuo representa una solución al problema a resolver. Respetando cierto vocabulario biológico, a esta representación se la puede asociar con un *cromosoma* y a las partes que la componen, con los *genes*. La cantidad de genes es fija e igual para todos los individuos de la población. Un gen se identifica por la

posición que ocupa en el cromosoma (*loci*) y representa un atributo de individuo, el cual puede tomar diferentes valores (*alelos*). También se utiliza el término *genotipo* para referenciar al cromosoma y distinguirlo del *fenotipo* o individuo que se construye a partir de la información codificada.

La información o "buenas características" que los individuos poseen, o representan, se denominan *building blocks* o *bloques constructivos*.

Cuál es el significado de los operadores genéticos?

La idea intuitiva del crossover es permitir el intercambio de información (o *building blocks*) entre individuos de la población.

La mutación, en cambio, permite incorporar nueva información que no se puede generar a partir de la clausura transitiva del crossover sobre la población.

Estos operadores tienen asociada una probabilidad de aplicación que llamaremos P_m para la mutación y P_x para el crossover. Antes de aplicarlos, se genera un número al azar $r \in [0 \ldots 1]$ y luego se lo compara con la probabilidad correspondiente. Si r es menor, el operador se aplica.

Finalmente, introduciremos un concepto relacionado con los *building blocks* y el crossover, el concepto de *epistasis*. En el ámbito de los AG, este término significa que existe una fuerte interacción entre genes de un cromosoma. Es decir, una parte de la solución es buena solo si aparece simultaneamente con alguna otra. En el capítulo 5 se verá claramente cuales son las consecuencias de la *epistasis*.

En la sección siguiente, y mediante dos ejemplos, mostraremos como los elementos descriptos anteriormente pueden ser definidos. Se define el problema y se plantean: una representación para los individuos, una función de evaluación y los operadores de crossover y mutación asociados.

3.3.2 Ejemplos de aplicación

Dada la funcion $f(x) = x * sin(10\pi * x) + 1.0$, deseamos encontrar el $x \in [-1 \ldots 2]$ que la maximice.

Utilizaremos un vector de dígitos binarios para representar los valores reales posibles para la variable x. Es claro que la longitud de este vector dependerá de la precisión que querramos.

Supongamos que utilizamos 6 decimales, entonces el rango $[-1 \ldots 2]$ debe ser dividido en $3 * 1.000.000$ partes. Este valor puede ser representado con 22 bits. Los vectores (0000000000000000000000) y (1111111111111111111111) representan los extremos del rango considerado; los vectores intermedios se transforman a números reales haciendo la transformación de base 2 a base 10 y aplicando el desplazamiento adecuado.

Cada vector binario representa un individuo. Por lo tanto, una forma simple de crear la población inicial es generar aleatoriamente vectores binarios de longitud 22.

La evaluación de un individuo es equivalente a la aplicación de la función $f(rep(v))$, donde $rep(v)$ es el valor real representado por el individuo v.

El operador genético de mutación, simplemente cambia el valor de uno o más dígitos binarios elegidos al azar. Si el valor era 0 lo cambia por 1 y viceversa.

Para ejemplificar el operador de cruzamiento o *crossover*, mostraremos el llamado $1Point - Crossover$. Dado un punto elegido al azar p_i (marcado con una línea vertical), con $1 \leq p_i \leq 22$, y dos individuos v_1 y v_2 se generan dos hijos de la siguiente manera:

supongamos que $p_i = 10$, entonces si

$v_1 = (1111100000|111110000011)$
$v_2 = (0000000000|111111111111)$

los hijos resultantes son:

$h_{12} = (11111000001111111111)$
$h_{21} = (00000000001111110000011)$

Esta forma de crossover puede generalizarse para n puntos de corte.

No siempre es factible representar las soluciones mediante vectores binarios. A veces es necesario utilizar estructuras de datos complejas, para las cuales el diseño de los operadores de mutación y crossover no es un tema trivial; estos elementos determinan la performance del *AG*.

Utilizaremos el "Problema del Viajante de Comercio"[7] para mostrar un caso donde la representación de los individuos no es un vector binario.

El TSP es un problema bien conocido:

> "dadas N ciudades $(c_1, c_2, \ldots, c_n)$ y una matriz de distancias $D : N * N$ (donde cada $D_{i,j}$ representa la distancia entre la ciudades c_i, c_j) el objetivo es encontrar un recorrido o "tour", es decir una permutación de la lista de ciudades, tal que la longitud del mismo sea mínima. La longitud de un tour se define como:
>
> $$L(Tour) = \sum_{k=o}^{k=N-1} D_{Tour(k),Tour(k+1)}$$
>
> donde $Tour(N)$ se identifica con $Tour(0)$ para cerrar el ciclo."

Una representación factible para los individuos, es utilizar un vector de enteros donde cada valor representa una ciudad. Cada individuo será entonces, una permutación de la lista de ciudades. La evaluación será simplemente medir la longitud del tour asociada a la permutación correspondiente.

La población inicial puede crearse en forma random o utilizando un mecanismo "goloso", como por ejemplo, la técnica del *vecino más cercano*:

Supongamos que tenemos N ciudades, y comenzamos el tour en la ciudad 4 (c_4), entonces:

$Tour = \{c_4\}$
$c_{prev} = c_4$
Mientras no estén todas las ciudades en el tour
  c_{curr} = **Más-Cercana**(c_{prev})
  $Tour = Tour + \{c_{curr}\}$
  $c_{prev} = c_{curr}$

Retornar $Tour$

donde c_{prev} es la ciudad previamente insertada, c_{curr} es la corriente y el proced. **Más-Cercana**(c_i) retorna la ciudad más cercana a c_i que no esté en el $Tour$

[7] TSP de ahora en más

Es decir dada una ciudad inicial, la heurística pone como siguiente ciudad en el tour aquella que esté más cercana a la previamente insertada. Es claro que iniciando desde diferentes ciudades se obtienen diferentes permutaciones.

Una posible mutación consiste en intercambiar la posición en la secuencia de un par de ciudades (c_i, c_j): a partir de $(c_1, c_2, \ldots, c_i, c_{i+1}, \ldots, c_j, c_{j+1}, \ldots, c_n)$, se obtiene $(c_1, c_2, \ldots, c_j, c_{i+1}, \ldots, c_i, c_{j+1}, \ldots, c_n)$

Veamos que pasa con el operador de crossover de 1 punto. Supongamos que nuestro problema tiene $N = 6$ ciudades. Sean I_1, I_2 los individuos padres y $p_c = 3$ el punto de corte, entonces:

$I_1 = (1, 2, 3, 4, 5, 6)$
$I_2 = (6, 5, 4, 3, 2, 1)$

El hijo que se obtiene es: $H_{12} = (1, 2, 3, 3, 2, 1)$ que no representa una solución factible al problema planteado. Claramente la utilización de más puntos de corte no resuelve este problema.

Dos alternativas son posibles en estos casos: diseñar y ejecutar un *algoritmo de reparación* que nos permita construir una solución factible a partir de una inválida; o utilizar operadores de crossover diseñados *ad hoc* , como por ejemplo *PMX, OX, CX* (por *mapped, order* y *cycle* crossovers).

Para una descripción de los aspectos concernientes a este problema y de los métodos aplicados para su resolución, sugerimos consultar [Mic96, Mos].

3.3.3 Mecanismos de Selección

Hasta el momento, hemos hablado de la representación de los individuos y de los operadores de crossover y mutación. En esta sección describiremos los mecanismos de selección, los cuales son responsables de asignar a los mejores individuos, mas chances de sobrevivir.

Para simplificar la explicación, veremos cada iteración del *AG* como un proceso en dos etapas. En la primera, se aplica algún mecanismo de selección sobre la "población actual" para crear una "población intermedia"; en la segunda etapa, a partir de esta población, se aplica crossover y mutación para generar una "nueva población".

A veces, la población intermedia recibe el nombre de *conjunto de entrecruzamiento* (*mating pool*).

Antes de continuar debemos aclarar el significado de dos términos: *evaluación* de un individuo y *fitness* de un individuo. Si bien ambos términos hacen referencia a la "bondad" de un individuo, el *fitness* permite establecer cuán bueno es ese individuo en el contexto de la población.

El mecanismo de selección es el responsable de decidir cuales individuos van a formar parte del conjunto de entrecruzamiento; es deseable que los individuos con mejores valores de fitness tengan mayor probabilidad de ser elegidos (*supervivencia del más apto*).

Definiremos f_p^i como el "fitness plano" del individuo i; es decir como el valor retornado por la función de evaluación, y definiremos f_n^i como el "fitness normalizado" del individuo i; este valor representará la calidad del individuo en el contexto de la población.

Existen diversos mecanismos para realizar la selección. Quizás el mas conocido sea el *muestreo estocástico con reemplazo* (*Stochastic sampling with replacement*)

En este método se define $f_n^i = f_p^i / F_p$ donde $F_p = \sum_{i=1}^{i=N} f_p^i$ y N es el tamaño de la población. De esta forma $0 \leq f_n^i \leq 1$ y $\sum_{i=1}^{i=N} f_n^i = 1$.

Luego se construye una "ruleta" o "rueda de la fortuna" con N sectores (1 por individuo), donde el tamaño del sector i es proporcional al fitness normalizado del individuo i.

En la Fig. 3.2 puede verse una ruleta con 5 individuos. El porcentaje indica cuanta porción de la ruleta es ocupada por cada individuo.

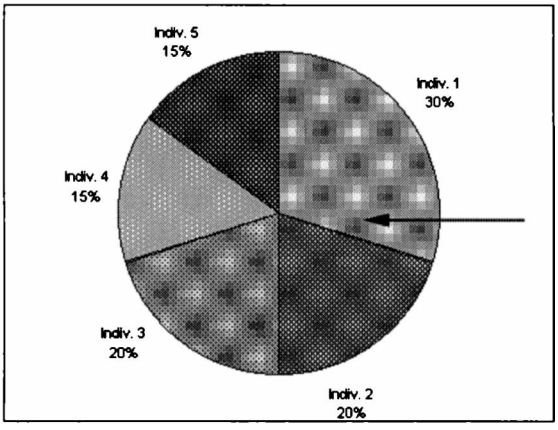

Figura 3.2: Ruleta para selección mediante muestreo estocástico con reemplazo

El proceso de selección consiste, entonces, en realizar "tiradas" de la ruleta y almacenar en el conjunto de entrecruzamiento, el individuo seleccionado. Como a mayor fitness normalizado, le corresponde mayor área de ruleta, los mejores individuos tendrán más posibilidades de ser seleccionados.

El principal inconveniente de este mecanismo se da cuando existe un individuo muy bueno en la población. Este individuo, al tener un f_n alto recibirá una porción muy grande de la ruleta y será seleccionado muchas veces. En consecuencia, en el conjunto de entrecruzamiento habrá varias copias del mismo, lo que puede provocar que en pocas generaciones, todos los individuos sean "parecidos" a él. Este fenómeno recibe el nombre de *convergencia prematura*.

Existe un mecanismo alternativo para hacer selección denominado *Muestreo estocástico universal* (*Stochastic universal sampling*) donde el fitness normalizado se define como: $f_n^i = f_p^i / F_{avg}$ siendo $F_{avg} = (\sum_{i=1}^{i=N} (f_p^i))/N$ y N, como antes, el tamaño de la población. Ahora $f_n^i \geq 0$. Es decir, el fitness normalizado representa cuán por arriba o por abajo está un individuo respecto al fitness promedio.

El proceso de selección es como sigue: se construye una ruleta como en el caso anterior, pero además se utiliza una ruleta externa de N punteros. Ahora, en una sola tirada de la ruleta, N individuos quedan seleccionados.

Este mecanismo implementa la siguiente idea: supongamos que el individuo i posee un $f_n^i = 2.35$, entonces se colocan 2 copias (la parte entera del valor) del individuo en el conjunto de entrecruzamiento y además se le da una chance de 0.35 de ubicar una tercera copia.

En la Fig. 3.3 puede verse la misma ruleta que en la Fig. 3.2 con los 5 punteros

dividiendo la ruleta en 5 porciones equivalentes.

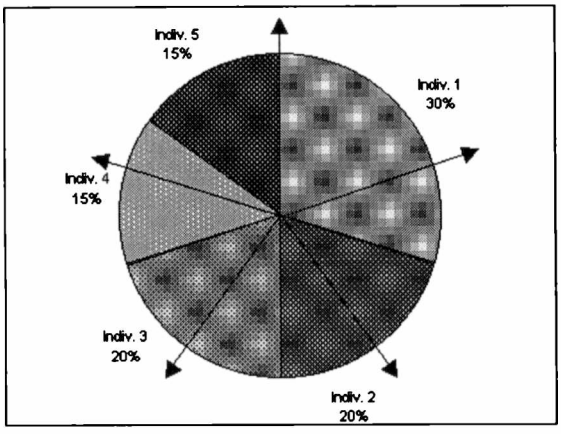

Figura 3.3: Ruleta para selección mediante muestreo estocástico universal

Sea cual fuere el método elegido, es válido plantear que el resultado del proceso de selección es un conjunto de entrecruzamiento formado por los mejores individuos de la población. En base a ellos, la etapa de crossover elegirá los "padres" y generará los "hijos"; sobre la salida del crossover, se ejecutará la etapa de mutación.

3.3.4 Por qué funcionan los AG?

En esta parte presentaremos dos conceptos importantes asociados a la teoría de AG: el Teorema del Esquema y la Hipótesis de Bloques Constructivos.

El **Teorema del Esquema** de Holland fue la primera explicación rigurosa de cómo los AG trabajan.

Se define un *esquema* como un patrón de valores de genes que pueden representarse (en código binario) por una cadena de alelos en el alfabeto {0,1,*}. Siendo el signo * un alelo especial introducido, que indica indiferencia. Un cromosoma particular contiene un esquema particular cuando los alelos de todos sus genes coinciden (coincidiendo * con cualquier alelo). De esta forma un esquema incluye a todos los individuos que encajan con su representación, por lo que se conoce también como patrón de similitud.

Por ejemplo el esquema 1*0* representa a los cromosomas 1000, 1100, 1101 y 1001. El esquema **** representa a todos los cromosomas de longitud cuatro; también puede expresarse que dicho esquema está contenido en todos los cromosomas de longitud cuatro.

El Teorema establece que:

> "En las sucesivas generaciones de un AG, la presencia de un esquema evoluciona estadísticamente de modo exponencial."

Holland demostró que la manera óptima de explorar el espacio de búsqueda, es asociando la probabilidad de reproducción con el fitness de los individuos. De esta manera los esquemas "buenos" se van presentando en una proporción, dentro de la población, de crecimiento exponencial durante las sucesivas generaciones. Además demostró que la cantidad de esquemas que que son efectivamente procesados en cada generación es de orden cúbico con respecto al tamaño de la población (propiedad de

paralelismo implícito).

La **Hipótesis de los Bloques Constructivos** plantea que:

> "Los AG exploran el espacio de búsqueda por medio de la yuxtaposición de esquemas ventajosos, cortos y de bajo orden."

Esta hipótesis de Goldberg aún no ha sido probada. Según ella el poder de un AG depende de su capacidad de encontrar buenos bloques constructivos. Al ser estos de longitud menor que un cromosoma, la idea es que se vayan incorporando gradualmente dentro de los individuos.

Está fuertemente asociada a las cuestiones de representación. Se dice que una representación (o codificación) es perfecta cuando posee las cinco propiedades siguientes:

1.- Completitud: Tiene la capacidad de representar todos los objetos de la clase indicada.
2.- Coherencia: No puede representar a un objeto que no sea de esa clase.
3.- Uniformidad: Todos los objetos deben estar representados por la misma cantidad de codificaciones.
4.- Sencillez: Debe ser fácil de codificar y decodificar.
5.- Localidad: Pequeños cambios en el objeto codificado debe corresponder con pequeños cambios en el objeto no codificado.

Aunque difícilmente se pueda encontrar una representación que verifique todos estos puntos, se dice una codificación es exitosa si favorece la formación de bloques constructivos, verifica que genes relacionados se encuentran cercanos dentro de un cromosoma y favorece una reducida interacción (epistasis) entre los genes.

3.3.5 Consideraciones generales

Varias preguntas quedan por responder: que tamaño debe tener el conjunto de entrecruzamiento, quienes forman la nueva población (hijos + padres, o solo hijos), si existen otros esquemas de selección, si el tamaño de la población puede variar, etc. Potencialmente cada respuesta posible a este tipo de preguntas, genera un algoritmo genético nuevo. Es por eso que al momento de escribir esta sección, nos planteamos como objetivo describir los conceptos mínimos para que el lector comprenda cuál es la idea general de los *AG*.

Nos permitimos recomendar al lector interesado en profundizar en estos temas, la lectura de [Mic96, Esh95, DGG+96] y la consulta a las referencias allí citadas.

3.3.6 Programación Genética

En esta sección introduciremos los conceptos básicos de Programación Genética[8] y citaremos algunos ejemplos de aplicación.

El paradigma de Programación Genética, propuesto por John Koza [Koz90] durante la década del '80, es una extensión de los algoritmos genéticos que difiere de éstos en la forma en que se representan los individuos de la población, la PG trata de generar soluciones a problemas a partir de la inducción de "programas" que lo resuelvan . No

[8] PG, en adelante

se especifica el tamaño, forma y complejidad estructural de los programas-solución sino que los programas evolucionan hasta generar soluciones satisfactorias.

En el vocabulario de PG, podemos tomar como equivalentes las nociones de programa de computadora, fórmula, plan, estrategia de control, procedimiento computacional, etc. De la misma forma, las entradas del programa de computadora pueden ser llamadas variables independientes, variables de estado, valores de sensores, argumentos de una función, etc. A su vez las salidas del programa de computadora pueden denominarse variables dependientes, un movimiento, un actuador, el valor regresado por una función, etc.

Como toda técnica evolutiva, en PG existen un conjunto de soluciones, mecanismos de selección, y operadores de cruzamiento y mutación.

Una forma simple de interpretar estos elementos, es considerar a cada individuo de la población como un árbol de derivación de una gramática "generadora" de programas.

Debe notarse que los arboles siempre serán *sintacticamente* correctos, aunque *semánticamente* pueden no tener sentido.

La evaluación de una solución implica ejecutar el programa y analizar la salida.

Los operadores de mutación y crossover operan, en su forma más básica, a nivel sintáctico sobre los arboles de derivación.

Por lo tanto, los puntos de corte o de mutación estarán asociados a símbolos terminales y no terminales del árbol correspondiente.

El operador de crossover funciona de la siguiente manera: se eligen dos padres (P_1, P_2) y de P_1 se elige al azar un símbolo no terminal k (esto representa un subarbol $t1_i^k$). Posteriormente se identifican en P_2 todos los subarboles con raíz k, esto determina un conjunto $\mathcal{T} = (t2_1^k, \ldots, t2_r^k)$. Finalmente, se elige algún $t2_j^k \in \mathcal{T}$ y se generan 2 hijos de la siguiente forma:

$$H1 = t1_i^k \odot_{P_1} t2_j^k$$
$$H2 = t2_j^k \odot_{P_2} t1_i^k$$

donde la operación $S_i^k \odot_T S_j^k$ reemplaza en el árbol T, el subarbol S_i por el subarbol S_j (ambos con raíz k). Es decir se generan dos nuevos arboles a partir del intercambio en los padres de los subarboles elegidos

La mutación toma como base al operador de crossover. Se elige el individuo a mutar P_1, se genera un individuo al azar P_{rand} y se aplica crossover entre ambos. De los dos hijos generados, uno se mantiene y el otro se descarta.

El diseño de operadores *ad hoc* para cada problema, permite incorporar conocimiento específico que permite hacer más efectiva la búsqueda de soluciones.

El área de aplicación de la PG se encuentra en constante expansión. Trabajos en el área relacionados con la evolución de estrategias de cooperación, procesamiento de datos temporales, generación de estructuras de datos, etc. pueden encontrarse en [Esh95]

3.4 Conclusiones

En este capítulo repasamos los conceptos básicos de complejidad computacional: problemas de decisión, de optimización, clases **P** y **NP** y problemas NP-Completos.

Dado que, de todas formas, los problemas "dificiles" deben ser resueltos, la utilización de métodos alternativos se vuelve imprescindible. Entre estos métodos sobresalen las técnicas heurísticas y dentro de ellas, destacamos las técnicas evolutivas.

Describimos sus motivaciones y desarrollamos una seccion especial para un miembro destacado de este conjunto de técnicas: los algoritmos genéticos. Introducimos los elementos básicos de los AG y ejemplificamos su uso en dos problemas diferentes.

A lo largo del capítulo se explicaron solamente aquellos elementos necesarios para la comprensión de este trabajo. Como complemento se ha citado abundante bibliografía para orientar al lector interesado en profundizar en estos temas.

Capítulo 4

Modelos Simples para Protein Folding

Los métodos experimentales (resonancia magnética nuclear, cristalografía de rayos X) utilizados para obtener la estructura tridimensional de una proteína, o para indagar en la dinámica del proceso de folding, son muy costosos y además, poco útiles para secuencias de tamaño importante.

Es por eso que el desarrollo de modelos adecuados para realizar simulaciones por computadora, es de gran importancia. Actualmente, con la tecnología computacional disponible, es imposible realizar simulaciones que involucren todas las interacciones proteína-solvente a nivel atómico.

En *"Modeling protein folding: the beauty and power of simplicity"* [E.I96] E. Shakhnovich plantea que las proteínas representan sistemas muy complejos para permitir una modelización exacta y sugiere la necesidad de realizar simplificaciones en la formulación de los modelos de trabajo.

Es decir, el desarrollo de técnicas computacionales que permitan indagar en las propiedades cuali-cuantitaivas de las proteínas debe ser realizado sobre modelos reducidos.

Dentro de estos modelos reducidos se destacan los modelos basados en reticulados. En ellos cada vértice de la grilla es ocupado por un aminoácido de la cadena y aminoácidos consecutivos se ubican en posiciones adyacentes del reticulado.

Entre otras características que los hacen atractivos, vale la pena destacar que permiten discretizar el espacio de conformaciones, facilitan el diseño y prueba de variantes (por ej. respecto a la dimensionalidad, y la cantidad de elementos involucrados en la modelización), y pueden utilizarse como soporte para recolectar información estadística.

En general, en cualquier modelo (reducido o no) deben estar claramente definidos cuatro elementos:

1.- Cuáles son los aminoacidos a considerar,
2.- Cómo se forman las secuencias válidas,
3.-Cómo se representa un "plegado",
4.- Cómo se mide la "bondad" de una estructura terciaria particular ("función de energía").

El punto 1 se refiere a que no todos los modelos utilizan los 20 aminoácidos que ocurren en la Naturaleza; por ejemplo, en el modelo de Dill que veremos más adelante, solo se utilizan 2.

El segundo punto está relacionado con el hecho que las secuencias válidas sean más restringidas que cualquier concatenación de aminoácidos. Por ejemplo, se pueden establecer restricciones respecto a la cantidad de aminoácidos del mismo tipo que pueden aparecer consecutivos.

Para representar las estructuras o "plegados", se pueden utilizar [PM98]:

- *Coordenadas Cartesianas*: cada aminoácido se representa con 2 o 3 coordenadas, dependiendo si la estructura pertenece al plano o al espacio tridimensional.
- *Coordenadas Internas*: la posición de cada aminoácido se define en términos de sus vecinos, especificando distancias, ángulos, etc.
- *Distancia Geométrica*: describe la estructura en términos de una matriz que contiene las distancias para cada par de aminoácidos

Los modelos en reticulados utilizan las coordenadas internas para modelizar las estructuras terciarias de las proteínas. Esto es, fijada la posición del aminoácido i, existen δ valores para representar la posición del $i+1$ dependiendo del reticulado utilizado. Por ejemplo, en un reticulado cuadrado $\delta = 4$, ya que un aminoácido puede estar *Arriba* o *Abajo* o a la *Izquierda* o a la *Derecha* de su predecesor en la cadena. Las estructuras quedan, entonces, representadas por strings $s \in \{Arriba, Abajo, Izquierda, Derecha\}^+$.

Si δ es un valor pequeño, todas las combinaciones posibles pueden ser testeadas. Potencialmente, existen δ^N posibles estructuras asociadas a una secuencia de longitud N y PF en estos modelos implica encontrar la única, o unicas, "correctas".

La representación en coordenadas internas es equivalente a la utilizada para describir los movimientos de la tortuga en el lenguaje *LOGO*.

A continuación se presentan en más detalle los modelos en reticulados, sus variaciones, las formas de medir cuán buena es una estructura y algunos aspectos interesantes de la relación *secuencia - estructura* que en ellos surge.

4.1 Modelos en Reticulados

Existe una gran cantidad de variantes dentro de los modelos en reticulados, y estas variaciones no se refieren solamente a la dimensionalidad (en 2D o 3D) sino también a que tipo de grilla es utilizada (triangular, cuadrada, etc), a las formas de medir las estructuras, etc. Todos estos elementos determinan la complejidad computacional del problema de optimización asociado.

El modelo más simple de PF en reticulados, es el llamado **modelo de Dill**.

Solo considera 2 tipos de aminoácidos, donde cada tipo representa si es hidrofóbico (representado con una H) o hidrofílico (representado con una P) .

Una proteína entonces, será una secuencia $w \in \{H, P\}^+$. Graficamente consideramos a los aminoácidos H de color negro y a los P de color blanco.

Generalmente se considera que la proteína se encuentra en un reticulado cuadrado o cúbico (según sean 2 o 3 dimensiones), donde cada posición es ocupada por a lo sumo un aminoácido.

La correspondencia entre aminoácidos y posiciones se llama *embedding* , y cuando es inyectiva se denomina *self avoiding*, es decir, los cruces no son permitidos.

	H	P
H	-1	0
P	0	0

	H	P
H	-3	-1
P	-1	0

Tabla 4.1: Matrices de interacción $\varepsilon_{i,j}$

La función de energía utilizada, solo tiene en cuenta las interacciones entre aminoácidos que sean adyacentes en el reticulado, pero no consecutivos en la secuencia (*vecinos topológicos*). Cada interacción de este tipo se denomina *bond*.

Entonces, dada una secuencia con n aminoácidos , $S = (s_1, s_2, \ldots, s_n)$, con $s_i \in \mathcal{A} = \{H, P\}$, un folding para S, $fold(S) = X = (x_1, x_2, \ldots, x_n)$ embebido en un reticulado $\mathcal{L}$, y una matriz de interacción $\varepsilon(s_i, s_j)$, una función de energía posible es:

$$E(S, X) = \sum_{i}^{n} \sum_{j>i+1}^{n} \varepsilon_{i,j} * \Delta(x_i, x_j) \tag{4.1}$$

donde $\Delta(x_i, x_j) = 1$ si x_i y x_j son adyacentes en el reticulado y no consecutivos en la cadena; y 0 en caso contrario; y $\varepsilon_{i,j} = \varepsilon(s_i, s_j)$ es el valor de la fila i, columna j en la matriz de interacción ε.
En la Tabla 4.1 se muestran dos matrices de interacción posibles.

Resolver PF en este modelo es equivalente a minimizar esta función de energía, o lo que es lo mismo a maximizar el número de bonds. Esta afirmación implica de cierta forma, aceptar que el estado nativo coincide con el estado de mínima energía libre.

Las estructuras se representan utilizando el sistema de coordenadas internas. Es decir, fijando la posicion en el reticulado del primer aminoácido, la posición de los restantes se fija utilizando movimientos relativos.

En el caso del reticulado cuadrado, ya vimos que estos movimientos son {*Arriba, Abajo, Izquierda, Derecha*} o, a partir de ahora, {*U, D, L, R*}.

Para aclarar los conceptos expuestos utilizaremos un ejemplo. En la Fig. 4.1 se muestra una instancia de este modelo en 2D. La secuencia es $P = HPPHPHHPHHPH$ y la estructura, representada en coordenadas internas, es $S = DRURRULULDL$. Como matriz de interacción para medir la energía de esa conformación utilizamos la primera matriz de las mostradas en la Tabla 4.1. Con líneas punteadas están unidos los aminoácidos que forman bonds. En este caso son 4. Por lo tanto, decimos que la energía de ese "plegado" para esa proteína es 4.

A pesar de la sencillez del modelo, es posible capturar uno de los elementos fundamentales del proceso de plegado: las interacciones hidrofóbicas. Es interesante ver Fig. 4.1 como aparecen las características de hidrofobicidad de los aminoácidos comentadas en el cap 2. Notese que, de alguna manera, los aminoácidos hidrofóbicos están "aislados del exterior" por una barrera de aminoácidos hidrofílicos que los rodean.

La utilización de reticulados cuadrados o cúbicos como soporte para discretizar el espacio de conformaciones, trae aparejado un serio problema: la *restricción de paridad* (o *parity constraint*).

Esta restricción esta asociada al hecho que en esos reticulados, los aminoácidos que se encuentren a distancia impar, nunca pueden ser vecinos topológicos.

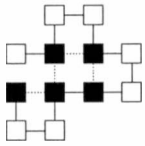

Figura 4.1: Instancia del modelo HP en reticulado cuadrado. Bonds = 4

Como resultado, secuencias del tipo $(HP)^+$ no pueden formar ningún bond, aunque en un "espacio real" sí podrían hacerlo.

Este problema se soluciona si se utilizan, por ejemplo, reticulados triangulares (ver Figs. 4.3 y 4.2). En ellos, para cualquier par de elementos (x, y) no consecutivos de la secuencia, es posible encontrar un *embedding* en el reticulado tal que x sea vecino topológico de y. (ver Fig. 4.4)

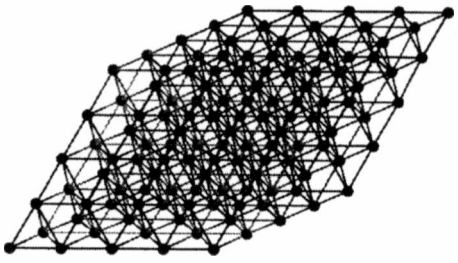

Figura 4.2: Reticulado triangular en 3 dimensiones. Figura tomada de [ABD+97]

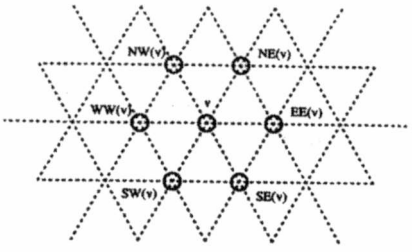

Figura 4.3: Coordenadas internas asociadas al reticulado triangular en 2D. Figura tomada de [ABD+97].

4.2 Análisis de la relación Secuencia - Estructura

El proceso de folding puede ser planteado como una función cuyo dominio son los elementos del espacio de secuencias y el codominio es el espacio de formas o estructuras.

Se sabe que la cardinalidad del conjunto de secuencias es mucho mayor que la del conjunto de estructuras [Bau96]; por lo tanto, caracterizar las relaciones existentes entre estos dos conjuntos es de suma importancia.

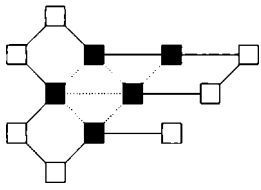

Figura 4.4: Instancia del modelo HP en reticulado triangular. Bonds = 6

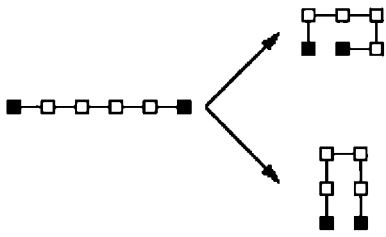

Figura 4.5: Ejemplo de secuencia degenerada y sus dos estructuras óptimas asociadas

Los estudios exhaustivos en reticulados son una herramienta fundamental para indagar en estas relaciones ya que es posible enumerar completamente todas las estructuras que son alcanzables por alguna secuencia, permitiendo obtener información termodinámica, estadística y estructural.

En primer lugar se define que una secuencia es no degenerada, cuando su estado nativo o valor óptimo está representado por una sola estructura (salvo rotaciones). Si más de una estructura representa el valor óptimo de la secuencia, diremos que es una secuencia degenerada.

Por ejemplo, la secuencia $w = HPPPPH$ es degenerada. Dos estructuras que representan el valor óptimo para la misma se muestran en la Fig. 4.5.

Además se dice que una estructura es diseñable, si representa el óptimo de alguna secuencia. De esta manera, se define la diseñabilidad D_S de una estructura S como el número de secuencias que tienen a S como su estructura óptima. Volviendo atrás, una estructura S es diseñable si $D_S \geq 1$.

A continuación, se describen algunos resultados interesantes que surgen a partir de estos estudios sobre reticulados cuadrados y triangulares.

En [BB97], se explora todo el espacio conformacional (en 2D) para secuencias de longitud 18. De las 2^{18} posibles secuencias, se verifica que solo 6439 son no degeneradas y solamente existen 1475 estructuras óptimas diferentes.

En [LHTW96] los autores realizan una enumeración completa de los estados nativos para cada secuencia de longitud 27 en el modelo HP. Analizando solo las estructuras "ubicables" en un reticulado cúbico de $(3 * 3 * 3)$, encuentran que solo el 4.75% de las secuencias son no degeneradas.

Además encuentran que algunas estructuras son altamente diseñables y que, asociadas con ellas, aparecen características de estructura secundaria ("similares" a las hélices-α y a las láminas-β en proteínas reales) que están ausentes en estructuras generadas al azar.

Otro dato interesante es que verifican que el comportamiento en reticulados de 2 o

3 dimensiones es similar y también plantean que las estructuras altamente diseñables son termodinámicamente más estables.

Los autores concluyen que:

> " ...real protein structures must be highly designable and mutable. Since highly designable structures are also more stable, such a selection principle solves the thermodynamic stability problem simultaneously. From an evolutionary point of view, highly designable structures are more likely to be picked through random selection of sequences in the primordial age, and they are stable against mutations.... If in fact nature only selects highly designable structures, then structure prediction algorithms should limit the search of the conformational space to these special structures, which could be only a tiny fraction of the total number of possible structures".

Nunes et.al. [NCH96], trabajando sobre una secuencia de longitud 14 en un reticulado triangular, analizan cuales son los tipos de movimientos en los que puede estar involucrado un aminoácido. Para ello, consideran efectos inerciales y vectores de dirección. De esta forma, le permiten a la proteína, "elegir" los movimientos más probables, en lugar de trabajar a partir de un conjunto predefinido de movimientos.

En [KPT98] se describe un proyecto en marcha para analizar estos resultados sobre secuencias más largas.

En [IS97] los autores realizan estudios exhaustivos sobre reticulados cuadrados y triangulares en 2D, considerando interacciones locales dependientes y no dependientes de la secuencia considerada.

La longitud de las secuencias analizadas varía entre 3 y 18 aminoácidos.

Los autores detectaron diferencias cualitativas entre ambos reticulados y las atribuyeron a la *restricción de paridad* de los reticulados cuadrados.

Utilizando la función de energía estandar del modelo HP, los autores reportan que en el reticulado cuadrado solamente el 2.5% de las secuencias son no degeneradas y que las estructuras más diseñables presentan un valor promedio de D_S que varía entre 2.9 y 4.3 secuencias.

Para el reticulado triangular, los valores son muchísimo menores. Además, no existen secuencias no degeneradas de longitud 13.

A partir de estos datos Sandelin e Irback extienden la función de energía para considerar interacciones locales independientes del tipo de aminoácidos involucrados. La función es la siguiente:

$$E = k * E_L + E_G \tag{4.2}$$

con

$$E_L = 2 * \sum_{i=2}^{i=N-1} (1 - cos(\theta_i)) \tag{4.3}$$

y

$$E_G = \sum_{1 \leq i < j \leq N} \varepsilon i, j * \Delta(x_i, x_j) \tag{4.4}$$

En la Eq. 4.3, θ_i es el ángulo definido por los aminoácidos x_{i-1}, x_i, x_{i+1}. Claramente, este ángulo es independiente del tipo de aminoácido. El factor k es el peso asignado a las interacciones independientes de la secuencia. Si $k = 0$, la Eq. 4.2 es igual a la Eq. 4.1.

Los autores encontraron que el número de estructuras diseñables estaba fuertemente asociado con el valor de k, y que este valor difería para reticulados cuadrados y triangulares. La mayor diferencia fue encontrada para $k = 0$ (modelo HP)

Sin embargo, el número máximo de estructuras diseñables en ambos reticulados, se dió para un valor de $k = -0.5$.

Sin duda estos trabajos contribuyen a la descripción del espacio de búsqueda a explorar y proveen elementos para el diseño de algoritmos. Un elemento sobre el que no hemos encontrado referencias, es acerca de características cualitativas de las secuencias no degeneradas, por ejemplo si existe alguna relación entre la cantidad de H y P de la secuencia, si es factible realizar algún agrupamiento de las mismas en función de la cantidad de bonds de las estructuras óptimas asociadas, etc. Creemos que las respuestas a estas preguntas podrían ser de ayuda, al menos, en la elección de las instancias de prueba de nuevos algoritmos.

En la sección siguiente, se describen algunos resultados de la complejidad computacional sobre los modelos en reticulados de PF.

4.3 Modelizaciones y Complejidad asociada

Varios intentos se han realizado para tratar de establecer la complejidad computacional del *PF* sobre estos modelos.

Sin embargo, cada demostración involucra una definición un poco "diferente" de lo que significa *PF* (en el caso más simple, PF *es* maximizar el número de bonds), por lo que al momento de establecer comparaciones entre modelos se debe tener en cuenta si el problema definido es el mismo.

En 1992, Ngo and Marks [NM92] muestran que un modelo tridimensional de predicción de estructura era *NP-Hard* reduciendo desde el problema de partición (*partition problem*).

En 1993, Fraenkel [Fra93] muestra que un modelo general en dos dimensiones (*MEP*) es *NP-Hard* a partir de una reducción desde el problema de *three dimensional matching (3DM)*.

En el mismo año, Unger y Moult [UM93a] utilizan un modelo similar al de Dill en tres dimensiones y prueban su *NP-Completitud* reduciendo desde el problema de *optimal linear arrangement* [GJ79a].

En 1997, aparece el trabajo de Paterson y Przytycka [PP97] que sin dudas fue un elemento importante para indagar en la complejidad de los modelos como el de Dill. Ellos definieron el *String Folding Problem* como sigue:

dada una cadena finita S, un entero k y una grilla Gr, existe un plegado de S en Gr con un puntaje no menor a k?

donde un plegado de S en Gr se define como un mapeo inyectivo F: $[1 \ldots n] \mapsto Gr$, donde $n = \mid S \mid$, y si $1 \leq i, j \leq n$, i=j-1 entonces $F(i)$ es adyacente a $F(j)$ en Gr;

El puntaje de F se computa contando el número de pares de símbolos idénticos que se mapean a nodos adyacentes de Gr. Esos pares se denominan *bonds*. En la figura 4.6 se muestra un ejemplo de una instancia de *String Folding* con puntaje 4.

En su trabajo Paterson et.al. muestran que *String Folding* es *NP-Completo* en Z^2y Z^3.

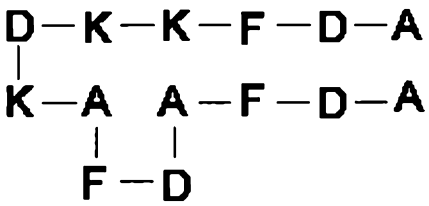

Figura 4.6: Ejemplo de una instancia de *String Folding* con puntaje 4

Dos elementos importantes de este trabajo se refieren a que el tamaño del alfabeto de aminoácidos no está acotado , y a que se utilizaron las restricciones de paridad del reticulado cuadrado en las construcciones y reducciones de las pruebas de complejidad.

Por la misma época aparecen algoritmos de aproximación. Hart e Istrail [IH97] presentan un algoritmo para reticulados cuadrados en dos y tres dimensiones con factores de aprox. de $\frac{3}{8}$ para tres dimensiones y de $\frac{1}{4}$ para dos.

Simultaneamente, Farach et. al. [ABD+97] presentan conjuntos de reglas para realizar "foldings" en reticulados triangulares (modelo de Dill) y logran obtener factores de aproximación para cada conjunto.

Finalmente, a principios de 1998, Crescenzi et. al. [CGP+98] probaron que *PF* en el modelo HP en dos dimensiones es *NP-Completo* y sugieren una extensión para probarlo en tres dimensiones. En el mismo trabajo se menciona un resultado independiente que prueba la *NP-Completitud* del modelo HP en tres dimensiones. Esta prueba fue realizada por Bonnie Berger and F.T. Leighton y figura como enviada para publicación al "Journal of Molecular Biology".

Debemos aclarar que hemos realizado cierto abuso del lenguaje al escribir esta sección del trabajo ya que es incorrecto plantear que el *PF* es *NP-Completo*; lo que es *NP-Completo* es el problema de maximizar el número de bonds en el modelo HP bajo un reticulado específico y una función de energía particular. De ninguna manera este resultado permite indagar en la complejidad del proceso biológico del folding; sí es razonable suponer que modelos más complejos de este proceso serán tan "difíciles" como el HP.

4.4 Conclusiones

En este capítulo presentamos los modelos en reticulados y justificamos por qué deben ser utilizados. Resumimos algunos trabajos que investigan la relación existente entre el conjunto de secuencias y el conjunto de estructuras, y reseñamos los principales resultados que sirvieron de base para la demostración de NP-Completitud de *PF* en el modelo de Dill.

Este resultado brinda bases teóricas al pesimismo existente respecto a la posibilidad de encontrar un algoritmo de orden polinomial que permita resolver *PF* en este modelo y deja plenamente justificada la utilización de métodos aproximados para solucionarlo.

Finalmente, para una discusión interesante sobre el rol de los modelos simples para *PF* sugerimos ver [E.I96] y `http:\\BioMedNet.com\cbiology\fad.htm`

Capítulo 5

Algoritmos Genéticos y Protein Folding

En este capítulo se reseñan y analizan algunos trabajos significativos en la aplicación de algoritmos genéticos a *PF* en el modelo de Dill. Se comparan tres implementaciones diferentes que utilizan el esquema de coordenadas internas para representar las estructuras y se plantean los problemas existentes asociados a esta representación.

5.1 Introducción

Varias técnicas se han utilizado para encarar la resolución del *PF*, entre ellas: Simulated Annealing [SSK94], métodos de programación lineal y entera, algoritmos genéticos, método de Monte Carlo [UM93c], etc. En [PKM95] se presenta un algoritmo polinomial determinístico que sigue las líneas de [UM93c].

Dos buenos ejemplos de aplicación de *AG* en *PF* son los trabajos de Unger y Moult [UM93b] y Patton et.al [PPG95]

En esta sección se intenta mostrar como los aparentemente "buenos" resultados alcanzados con los algoritmos genéticos desarrollados hasta el momento no son tan buenos y se sugiere la manera de corregirlos [1].

5.1.1 Algoritmo de Unger y Moult

En su trabajo [UM93b], Unger y Moult afirman que [2]:

> "... Nuestra aplicación de AGs al problema de plegado de proteínas puede verse como una extensión del método de Monte Carlo que incluye intercambio de información entre un conjunto de simulaciones paralelas"

Se describen a continuación las características más importantes de esta implementación y más adelante se refuta el comentario previo.

Unger y Moult adoptan en su trabajo el modelo de Dill en un reticulado bidimensional e implementan un AG para resolver el problema. Dicho algoritmo genético

[1] El material de este capítulo fue presentado en [KPL+98], [KPLdlC97b] y [KPL+97]

[2] La traducción es propia

aplica el método de *Simulated Annealing* para realizar filtrados en los períodos de mutación y crossover.

Simulated Annealing puede sintetizarse de la siguiente manera:
se parte de una conformación inicial aleatoria S_1 con energía E_1. Luego se realiza un pequeño cambio aleatorio en S_1 para generar S_2 y se calcula su energía, E_2. Si $E_2 \leq E_1$ entonces se acepta S_2. En caso contrario se elegirá, en forma no determinística, si el cambio es aceptado o no de acuerdo a alguna función de probabilidad. Generalmente el criterio es:

$$Rnd \leq \exp^{\frac{E_1 - E_2}{c_k}}$$

donde $0 \leq Rnd \leq 1$ y c_k es el factor de enfriamiento (*annealing*) que va cambiando en función de k, siendo k el número de pasos de iteración. Si no se acepta, se elimina S_2 y se continúa iterando a partir de S_1 nuevamente. En caso de aceptarse, el proceso continúa desde S_2. El proceso se repite hasta alcanzar algún criterio de finalización.

El esquema de Unger y Moult es un esquema de mutación, crossover y selección, donde los pasos de mutación se corresponden con un paso del método arriba descripto. Por esta razón, se plantea que este algoritmo es un híbrido de AG con Monte Carlo.

Los individuos se representan utilizando el esquema de coordenadas internas: cada individuo es una palabra $w \in \{U, D, R, L\}^+$

Para cruzar dos individuos, se utiliza 1−point crossover. Se elige un punto al azar, se corta cada padre por ese punto y se unen las subsecuencias complementarias para generar dos individuos nuevos. Existen tres maneras de unir los substrings, a 0, 90 o -90 grados. Si la energía de estos individuos es menor que el promedio de la energía de los padres entonces son aceptados, en caso contrario se decide no deterministicamente (entre descartarlo y generar otro, o aceptarlo), utilizando el mismo procedimiento que el descripto en Simulated Annealing.

5.1.2 Algoritmo de Patton

En este trabajo [PPG95] se critica el algoritmo de Unger y Moult, no solo por su carácter híbrido, sino tambien por:

- La inicialización no random de la población.
- El alto grado de mutaciones.
- El filtrado de los resultados del crossover y las mutaciones mediante iteraciones de tipo *simulated annealing*.
- El efecto en cascada y acoplado de, por un lado, rechazar las configuraciones inválidas, y por otro lado, los filtrados mediante *SA*, desvían la capacidad de exploración y explotación del AG.
- El reemplazo de toda la población, excepto del individuo mejor.
- La utilización, sin restricciones, de coordenadas internas para representar los individuos.

Para corregir estos "errores" los autores presentan un *AG* con alta probabilidad de crossover (de 1 y 2 puntos), baja probabilidad de mutación y población de tamaño

variable (1000 indiv. +/- 20%). Además admiten configuraciones con cruces en las cuales se penalizan las posiciones con colisión. Su codificación en base a movimientos relativos entre el aminoácido i y el $i-1$ evita algunas colisiones obvias (por ejemplo UD o LR).

Con este algoritmo, los autores consiguen igualar e incluso mejorar los valores de energía obtenidos utilizando un número mucho menor de evaluaciones. En base a estos resultados, consideran que sus ideas han sido verificadas. Ver Tabla 1 en [PPG95] para una descripción completa de los resultados.

Sin embargo nuestras hipótesis de trabajo suponen que los problemas marcados por Patton en la solución de [UM93c], si bien, importantes, no son condición necesaria y suficiente para el mal desempeño del AG en cuestión.

Para demostrar esto se implementó un AG con las supuestas "dudosas" características del trabajo cuestionado y con esas características perniciosas se obtuvieron mejores conformaciones que en [UM93b], con un *speed up* comparable al de [PPG95].

En la secciones siguientes, se describe el AG implementado y se explican los resultados obtenidos.

5.1.3 Características del AG

Por restricciones computacionales se redujo el modelo a 2 dimensiones; sin embargo los principios a estudiar no se pierden al bajar la dimensionalidad del problema [LHTW96, Kar96, HS96]. Cada individuo, como en el trabajo de Unger, es un $w \in \{U, D, R, L\}^+$. Las características principales de la implementación se describen a continuación:

- inicialización no random de la población (del orden de los 50 individuos).
- alto grado de mutaciones.
- Probabilidad de macromutaciones diferente de cero.
- 1-point y 2-point crossover.
- Reemplazo de toda la población, excepto el *elite set*.
- Se admiten configuraciones no válidas en las cuales se penalizan las colisiones.

Durante el proceso de cruzamiento dos padres generan solo un hijo. Esto obliga a usar un conjunto de entrecruzamiento de $2 * (N - Z)$ genomas, siendo Z el tamaño del conjunto de elite.
En la etapa de crossover se seleccionan dos padres y se cruzan con una probabilidad de P_x; si el entrecruzamiento no acontece, entonces el padre con mejor fitness se copia a la proxima generación.

Las macromutaciones son mutaciones que afectan una porción de la estructura y no una sola posición. Existen cuatro opciones de macromutación. Todas ellas, a partir de la selección de dos puntos, modifican la subsecuencia intermedia. Las modificaciones posibles son:

- Rotación: se modifica la subsecuencia para representar un giro de 0, 90 o -90 grados.

- Reflexión: se modifica la subsecuencia para reflejarla, vertical u horizontalmente.
- Estiramiento: la subsecuencia se "estira" en alguna dirección
- Random: cada posición de la subsecuencia es modificada al azar.

La inicialización heurística y no aleatoria de la población lejos de ser perjudicial para el AG, como supone Patton, lo ayuda a salir de las regiones de configuraciones infactibles del espacio de búsqueda. Por otro lado, el crossover en este trabajo es mucho más sencillo que el de Patton que no es escalable debido al alto costo de decodificación, $O(||i|| * n^2)$ donde n es la cantidad de posiciones relativas posibles e $||i||$, el tamaño de la instancia.

5.1.4 Resultados Obtenidos

Las pruebas se hicieron sobre instancias de proteínas extraídas de [UM93c].

Pese a la elección de las características más desfavorables, los valores de energía alcanzados con este AG son mejores que los mostrados en [UM93c] y presentan un speed up comparable a los de [PPG95]. Para una descripción completa de estos resultados, ver [KPLdlC97a, KPL+98]

En el trabajo se realizaron simulaciones exhaustivas para detectar los mejores valores para las probabilidades de crossover, mutación y macromutación (P_x, P_m, P_{mm}). Los resultados obtenidos permitieron verificar que en todos los casos en los que el número de bonds era máximo, la probabilidad de crossover, P_X, era menor o igual que las otras dos probabilidades.

Estos hechos parecerían estar en conflicto con la intuición generalizada respecto del comportamiento de los AG. Es ampliamente aceptado que tanto P_{mm} como P_m deben ser mucho menores que P_x.

Sin embargo en estos experimentos se ha encontrado que los mejores conjuntos de parámetros son aquellos en los cuales las probabilidades de macromutación y mutación son mayores que la de crossover.

Elementos como este permiten afirmar que las críticas de Patton et.al. al trabajo de Unger son solo superficiales.

5.1.5 Interpretación de los resultados

Tradicionalmente en la literatura de algoritmos genéticos se encuentra que la presentación de resultados responde a algunos cánones.

Cuando el algoritmo genético ha sido utilizado como herramienta de optimización combinatoria en algun problema en particular, los autores se conforman con obtener "buenas" soluciones y realizar comparaciones con otros trabajos.

Lo que muy pocas veces se encuentra es un análisis de "cual" fue el camino que recorrió el algoritmo genético para alcanzar esos "buenos" resultados, es decir, un análisis de la dinámica del AG.

En el caso de los AG para PF, este análisis no se ha realizado , o al menos, no ha sido presentado. Probablemente por este motivo, no se ha notado que se está desaprovechando la potencia de los AG para resolver problemas complejos como el PF.

Recordemos que existen dos mecanismos fundamentales en los que un AG se basa para resolver un problema: la mutación y el crossover.

El crossover sirve basicamente a dos propósitos: la transferencia de información entre individuos de la población y, el mantenimiento y reorganización de genes con el objeto de descubrir los building blocks.

La mutación, en cambio, es utilizada para incorporar material genético nuevo a la población, lo que permite obtener individuos que estarían fuera de la clausura transitiva del crossover, y por ende, inalcanzables mediante cruzamiento.

En los trabajos descriptos, el operador de crossover se ha utilizado incorrectamente. La falta de relación entre éste y la representación de coordenadas internas, impide la transferencia de información entre individuos de la población. Este fenómeno se evidencia en todos los trabajos con AG aplicados al PF que utilizan un esquema de representación basado en coordenadas internas.

Veamos, mediante el ejemplo de la Fig. 5.1 que es lo que ocurre cuando se utiliza 1−point crossover.

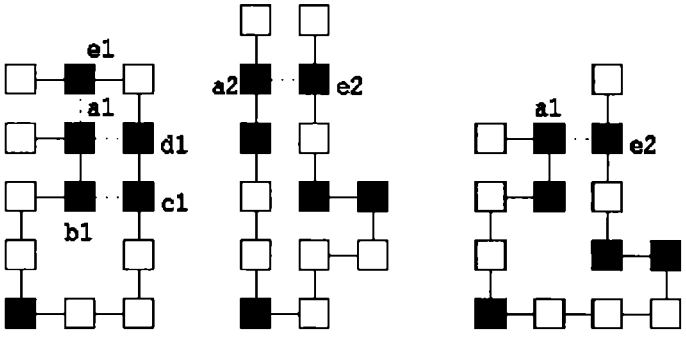

Figura 5.1: Ejemplo de 1-Point Crossover mostrando el carácter epistático de la representación de coordenadas internas

En la figura citada se muestran 2 padres y un hijo de izquierda a derecha. Los bonds en cada proteína se marcan con una línea punteada. Así, el primer padre tiene 3 bonds y el segundo solo 1, al igual que el hijo.
Notese que el bond hallado en el hijo no es transmitido de los padres; esto es, la nueva conformación no hereda nada de sus predecesores. Ese bond esta construído con los aminoácidos (a_1, e_2). Podemos ver que no se produjo intercambio útil de información.

El operador no solo no transmitió información entre individuos de la población, sino que tambien destruyó los bonds previamente encontrados.

El carácter altamente epistático de la representación, determina que los bonds en los padres esten formados por aminoácidos muy lejanos en la representación.

En el ejemplo, $\{(a_1, e_1), (a_1, d_1), (b_1, c_1)\}$ para el primer padre, y $\{(a_2, e_2)\}$ para el segundo.

Es decir, *el mal desempeño se debe a la pobre interacción entre la codificación utilizada y el crossover elegido.* Cuando ocurra una etapa de apareamiento bajo esta codificación con un crossover de tipo N−point, es muy poco probable que los genes que codifican estos bonds sean transferidos juntos. Más aún, suponiendo que se transfieran juntos es poco probable que queden en posiciones relativas que les permitan volver a constituir los bonds en cuestión.

En forma independiente y en trabajos posteriores, el mismo problema y similares conclusiones fueron reportadas en [KC97, PM98]. Coveney sugiere en [KC97] que este problema puede ser solucionado mediante la utilización de un crossover de N puntos.

En su trabajo [PM98] Piccolboni, en cambio, plantea la utilización de un esquema diferente de representación basado en técnicas de *distancia geométrica* donde cada individuo representa una matriz de distancias entre todos los pares de aminoácidos de la secuencia. Aparentemente, esta representación cumple la condición de ausencia de epistasis aunque la evaluación y construcción de las estructuras no es trivial.

5.2 El Mecanismo y la Idea de Crossover

Existe una interpretación simple que permite comprender las cuestiones planteadas en la sección previa. En [Jon95] se distingue entre el concepto o *Idea* del crossover y la*Mecánica* del mismo.

La *idea* subyacente a la utilización de crossover es que los esquemas prometedores que algún individuo fue descubriendo a lo largo de su evolución sean comunicados a la población.
Se espera intercambio de información; de esta forma una población que inicialmente estaba compuesta por individuos aleatorios, debido a la selección convergerá hacia una población con individuos cuya calidad sea mejor que la de una población random.

La *Mecánica* del crossover es una implementación en particular de la idea arriba descripta. Si bien todas las implementaciones se remiten a esta idea, en general pueden ser muy diferentes entre sí.

Un aspecto muy importante a tener en cuenta cuando se analiza el comportamiento de un algoritmo genético en algun entorno de explotación es verificar cual es el beneficio verdadero que reporta el uso del crossover. Esto es, verificar la utilidad no solo de la mecánica del mismo sino tambien, constatar que se esté utilizando la idea del entrecruzamiento.

Si la *Mecánica* del crossover no implementa la *Idea*, el uso de una población no es necesario, y potencialmente, se podrían obtener los mismos resultados realizando hillclimbing en base a macromutaciones en un solo individuo.

Terry Jones, en su tesis de doctorado [Jon95], sugiere un experimento para verificar si el crossover funciona correctamente en un determinado AG. El experimento, denominado *HeadLess Chicken Test*, consiste en comparar un AG estándar con uno idéntico pero que utilice "random crossover". El random crossover funciona de la siguiente manera:

se eligen dos padres $P1$ y $P2$ y se generan 2 individuos aleatorios $R1$ y $R2$. Luego, se cruzan $P1$ con $R1$ y $P2$ con $R2$ para generar dos hijos H_1 y H_2 de los cuales uno se elige (al azar) y el otro se descarta.

Es claro que el crossover aleatorio mantiene la mecánica del crossover pero no refleja la idea de transmitir información.

Si luego de comparar la performance del AG estándar con el AG con random crossover, se observa que el random iguala o supera al crossover "bueno", entonces podemos descartar la población.

En el trabajo descripto, este experimento estuvo incluído, ya que se utilizaron operadores de macromutación (equivalentes a un crossover random de 2 puntos) con los mismos rangos de probabilidades que la mutación y el crossover.

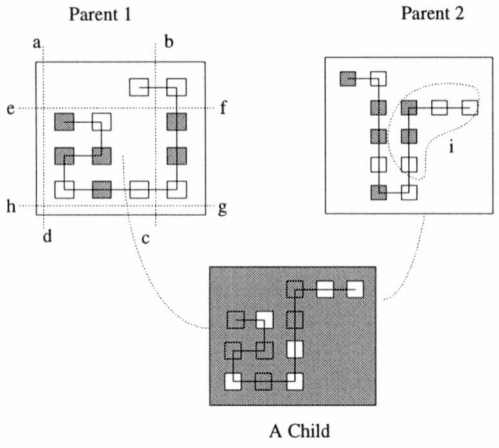

Figura 5.2: El crossover 2DMC: se muestran 2 padres (aParent1 y aParent2) y un hijo (aChild). La región limitada por las líneas verticales $\overline{ad}, \overline{bc}$ y las horizontales $\overline{ef}, \overline{hg}$ determinan la región del genotipo del padre 1 que serán heredadas por el hijo.

5.3 El Crossover 2DMC: una posible solución?

Basicamente existen dos soluciones posibles al problema planteado en la sección anterior: la primera involucra la definición de operadores de crossover adecuados donde se verifique que la *Idea* del mismo queda reflejada; y la segunda consiste en cambiar la representación y construir un nuevo AG con sus respectivos operadores. También en este caso los experimentos para evaluar el beneficio real del crossover deben ser realizados.

En esta sección se describe un nuevo crossover denominado: *2DMC* siglas de *2 Dimensional Memetic Crossover* o *Crossover Memético Bi-dimensional*

El 2DMC presenta características peculiares. Como se mantiene la representación de coordenadas internas, es necesario separar claramante el genotipo (la secuencia) del fenotipo (la estructura). Este crossover elige, al azar, partiendo del fenotipo, que parte del genotipo debe ser transferida. Este mecanismo es exactamente a la inversa del comportamiento usual del crossover.

El funcionamiento básico del 2DMC es el siguiente: dada una secuencia de aminoácidos S y dos padres P_1 y P_2 representando sendas conformaciones, se calculan dos rectángulos o *bounding boxes* que permitan embeber las estructuras. Luego, se selecciona un subrectángulo o bloque en alguno de los padres que permite determinar subsecuencias de S.

Este subrectángulo forma la base para la generación de un hijo. Las partes faltantes de la secuencia son heredadas del padre restante considerando el complemento del bloque seleccionado.

Para aclarar las ideas expuestas, consideremos el ejemplo de la Fig. 5.2. Allí se muestran 2 padres (aParent1 y aParent2) y un hijo (aChild). La región limitada por las líneas verticales $\overline{ad}, \overline{bc}$ y las horizontales $\overline{ef}, \overline{hg}$ determinan la región del genotipo del padre 1 que serán heredadas por el hijo. Los elementos que restan para completar el hijo provienen de la región i del padre 2.

El problema básico que surge con este crossover es como "reconectar" los segmentos en el hijo. La forma más simple se muestra en la Fig. 5.3

```
Crossover_2DMC(Padre1, Padre2, Hijo)
[0] BEGIN
[1]  copiar Padre1 en Hijo
[2]  seleccionar un rectangulo al azar en el Hijo
[3]  Si el Hijo comienza afuera del rectangulo seleccionado, entonces
[4]    copiar desde el Padre2 al Hijo toda la estructura que vaya
       desde el primer aminoacido hasta el ultimo afuera del rectangulo
[5]  para cada par de aminoacidos (i,j) donde i es un aminoacido que sale
     del rectangulo y j es el siguiente en entrar, hacer:
       [6] d  <--- es el i-esimo movimiento relativo en el Padre2
       [7] d1 <--- es el i-esimo movimiento relativo en el Hijo
       [8] while ((exista un mov. relativo d en el Hijo entre las pos. i y j)
                  y (i<j)) do:
              [9]    copiar el i-esimo mov. relativo desde el Padre2 al Hijo
              [10]   en el Hijo, cambiar el mov. relativo d por el d1
              [11]   i  <--- i + 1
              [12]   d  <--- es el i-esimo mov. relativo en el Padre2
              [13]   d1 <--- es el i-esimo mov. relativo en el Hijo
      [14]   end-while
[15] end-for
[16] si la estructura termina fuera del rectangulo, entonces
[17] copiar desde el Padre2 al Hijo la subestructura que comienza en el ultimo
     aminoacido que sale del rectangulo hasta el final de la cadena
[18]END.
```

Figura 5.3: Pseudo Código del crossover 2DMC, versión original

Esta forma de "reconectar" los segmentos produce un gran número de colisiones.

Una versión más sofisticada, le permite al crossover *mirar hacia adelante* cual es la mejor forma de reconectar y generar el hijo a partir de ella. La posibilidad de mirar hacia adelante o *look ahead* permite optimizar localmente la estructura generada. A esta optimización o búsqueda local se debe el término *memético* utilizado para caracterizar al crossover. Este término se utilizó por primera vez en [Mos89] en el contexto de "Algoritmos meméticos"; basicamente son AGs donde aparecen etapas de búsqueda local que permiten a los individuos "mejorarse a sí mismos".

El 2DMC con *look ahead* prueba todas las formas posibles de reconectar dos segmentos en función de un parámetro K que establece la profundidad de la recursión ejecutada.

El pseudocódigo de esta versión se muestra en la Fig. 5.4. Los cambios respecto a la versión simple aparecen en los puntos [9] y [10]. Se define que un movimiento es mejor que otro si el número de inconsistencias que genera es menor.

La probabilidad de intercambio de información, asociada directamente a la probabilidad de elegir subrectángulos de cierta área, puede ser cuantificada.

Por ejemplo, si se desea conocer cual es la probabilidad de transferir desde el primer padre al hijo, un rectángulo de r unidades cuadradas de una secuencia de longitud L, se debe realizar el siguiente cálculo:

$$\sum_{a,b \in Div(r) | a*b=r} P(a,b,L) \tag{5.1}$$

```
Crossover_LA-2DMC(Padre1, Padre2, Hijo, K)
[0] BEGIN
[1] copiar Padre1 en Hijo
[2] seleccionar un rectangulo al azar en el Hijo
[3] Si el Hijo comienza afuera del rectangulo seleccionado, entonces
[4]   copiar desde el Padre2 al Hijo toda la estructura que vaya
      desde el primer aminoacido hasta el ultimo afuera del rectangulo
[5]  para cada par de aminoacidos (i,j) donde i es un aminoacido que sale
     del rectangulo y j es el siguiente en entrar, hacer:
       [6] d  <--- es el i-esimo movimiento relativo en el Padre2
       [7] d1 <--- es el i-esimo movimiento relativo en el Hijo
       [8] while ((exista un mov. relativo d en el Hijo entre las pos. i y j)
                  y (i<j)) do:
             [9]  copiar el i-esimo mov. relativo desde el Padre2 al Hijo
             [10] en el Hijo, cambiar el mov. relativo d por el d1
             [11] si el cambio realizado en [10] genera inconsistencias, entonces
             [12] hacer una busqueda exhaustiva con profundidad K (k look-ahead)
                  buscando por el mejor mov. relativo posible en funcion del
                  numero de inconsistencias generadas.
             [13] en el Hijo, cambiar el movimiento relativo d
                  por el mejor hallado en el paso [12]
             [14] i  <--- i + 1
             [15] d  <--- es el i-esimo movimiento relativo en el Padre2
             [16] d1 <--- es el i-esimo movimiento relativo en el Hijo
      [17] end-while
[18] end-for
[19] si la estructura termina fuera del rectangulo, entonces
[20] copiar desde el Padre2 al Hijo la subestructura que comienza en el ultimo
     aminoacido que sale del rectangulo hasta el final de la cadena
[21]END.
```

Figura 5.4: Pseudo Código del crossover 2DMC, con *Look-Ahead*

donde, en general $P(a, b, L)$ es

$$P(a, b, L) = p1(a, L) * p1(b, L) \tag{5.2}$$

y

$$p1(k, L) = \begin{cases} 1/L & if \quad k=0 \\ \frac{2*(L-k)}{L^2} & if \quad 0<k\leq L-1 \end{cases} \tag{5.3}$$

Desafortunadamente, los primeros resultados obtenidos a partir del 2DMC no han sido satisfactorios desde el punto de vista de los valores de bonds obtenidos.

Si bien es probable que el 2DMC funcione "mejor" que un 2DMC random, parece ser complejo encontrar buenos operadores de crossover que funcionen correctamente bajo la representación de coordenadas internas para los individuos. En el marco de un trabajo paralelo a esta tesis, se están realizando experimentos sistemáticos para poder obtener conclusiones precisas respecto a la bondad del crossover propuesto y a su relación con el análisis probabilístico presentado anteriormente.

5.4 Conclusiones

En este capítulo se han descripto tres implementaciones diferentes de AG para PF y se ha demostrado como los, supuestamente buenos, resultados obtenidos en estos casos no son tales.

A partir de los experimentos realizados se puede afirmar que las críticas de Patton al trabajo de Unger y Moult, si bien justificadas en ciertos aspectos, no son condición necesaria y suficiente para que un AG falle al resolver el *PF*.

Para demostrar esto se ha implementado una version de AG que posee dichas características "malas" y aún así se han obtenido resultados mejores que en [UM93c] y comparables con [PPG95].

Para los algoritmos descriptos, los mismos resultados se pueden obtener prescindiendo de la población y optimizando con un solo individuo que haga hillclimbing, como sugiere Jones en [Jon95].

Esta conclusión se fundamenta en el hecho de no existir una buena interrelación entre la codificación utilizada y el crossover elegido. De tal forma que lo único que se esta explotando del crossover es su mecánica y no la idea del mismo.

Por último, la afirmación de Unger y Moult citada en 5.1.1, e implicitamente aceptada por el equipo de Patton ha sido refutada.

Con estos resultados en mano, se propuso uno de los caminos posibles para intentar solucionar el problema detectado: se diseñó un nuevo crossover denominado *2DMC* cuya principal característica es que opera primero a nivel de estructuras y luego genera los cambios correspondientes en la secuencia.

Los resultados obtenidos a partir de su utilización no han sido satisfactorios; en un proyecto paralelo a esta tesis, se están realizando experimentos para verificar en forma sistemática su funcionamiento.

Es importante notar que el tipo de análisis propuesto y realizado en este capítulo es de relevancia no solo para *AG* en *PF*; el mismo análisis debería aplicarse a cualquier *AG* en cualquier aplicación particular para verificar si la herramienta se está utilizando en todo su potencial.

En los capítulos siguientes se propone una nueva alternativa para solucionar los problemas planteados.

Capítulo 6

Autómatas Celulares

6.1 Introducción

Los Autómatas Celulares[1] son sistemas descentralizados, formados por un gran número de componentes simples e idénticos conectados en forma local.

Los *CA* se consideran objetos matemáticos sobre los cuales se puede dar una demostración formal a sus propiedades.

Son capaces de ejecutar computaciones sofisticadas y permiten modelizar el comportamiento de algunos sistemas complejos que surgen en la naturaleza. Por ejemplo, han sido usados para modelar fenómenos físicos y biológicos tales como la dinámica de fluídos, la formación de galaxias, terremotos, y la generación de formas biológicas.

También se utilizan como dispositivos de computación paralela, en aplicaciones de Procesamiento de imágenes, reconocimiento de caracteres, Aritmética paralela, etc.

Desde hace unos años, los *CA* se utilizan como modelos abstractos para estudiar como a partir de interacciones entre componentes simples, ciertos sistemas complejos logran exhibir un comportamiento global o *emergente* sin la intervención de un controlador central.

Ejemplos de este comportamiento son las bandadas de pájaros volando en formación sin un coordinador global, las células cerebrales (neuronas) cuya interacción produce, por ejemplo, la capacidad de razonamiento, etc.

En las secciones siguientes se presentan las principales características de los *CA*, algunos resultados que dan cuenta de su complejidad computacional y finalmente se mencionan algunos elementos que permiten relacionarlos con los sistemas complejos.

6.2 Características de los *CA*

La notación utilizada para describir a los *CA* es muy variada. En este trabajo nos guiaremos por la sugerida en [Mit96]

Un *CA* está formado por dos componentes: un *espacio celular* y una *regla o función de transición*.

El *espacio celular* es un reticulado de N máquinas de estados finitos idénticas llamadas *celdas*, conectadas entre sí mediante algún patron. Potencialmente, el reti-

[1] *CA* de ahora en más

000	001	010	011	100	101	110	111
↓	↓	↓	↓	↓	↓	↓	↓
b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7

Tabla 6.1: Tabla de reglas general para los *ECA*'s

culado puede ser infinito. A los fines prácticos, se debe establecer cierta longitud y agregar las condiciones de borde necesarias para trabajar en los extremos.

Cada celda c, puede tomar valores o "estados" de un conjunto Σ. La cantidad de estados posibles es $k = ||\Sigma||$.

Se denota como S_i^t, con $1 \leq i \leq N$, al estado de la celda c_i en instante de tiempo t. $(S_i^t \in \Sigma)$.

Se define la *vecindad* η_i^t de la celda c_i como el conjunto de estados formado por S_i^t y los estados de las celdas con las cuales la celda c_i está conectada.

La *regla o función de transición* $\Phi(\eta_i^t)$ da el estado de actualización (*update*) S_i^{t+1} para cada celda c_i como una función de η_i^t.

Los *CA* reciben una señal de *update* desde un reloj global imaginario; a cada paso del tiempo todas las celdas actualizan sus estado sincrónicamente de acuerdo a $\Phi(\eta_i^t)$.

Uno de los *CA* más simples es el de una dimensión donde cada celda toma un estado $S_i \in \Sigma = 0, 1$. Cada celda c_i está conectada con cierto número r de celdas a su derecha y a su izquierda.

Este valor r se denomina *radio* y si, por ejemplo, $r = 1$ entonces la celda c_i estará conectada con las celdas c_{i-1} y c_{i+1}.

Si el reticulado es circular o periódico entonces existirá conexión entre las celdas c_N y c_1.

El tamaño de la vecindad η_i^t es $2r + 1$ y la función de transición en estos casos es $\Phi : \Sigma^{2r+1} \rightarrow \Sigma$.

En los *CA* con estados binarios, donde el número de conexiones posibles no es demasiado grande, la reglas del *CA* o función de transición se pueden mostrar como una *tabla de lookup*, o tabla de reglas.

Esta tabla contiene, para cada vecindad posible, cual es el valor de actualización para el estado de la celda considerada.

Supongamos que tenemos un *CA* binario con $r = 1$. El tamaño de la vecindad es 3 y existen 2^3 posibles vecindades cada una asociada con un estado de *update* Estos autómatas se denominan *CA elementales* o *ECA*'s.

En la Tabla 6.1 se muestran las 8 vecindades ordenadas por orden lexicográfico y los estados de update.

Cada $b_i \in \Sigma = \{0, 1\}$ puede considerarse como un bit y el vector $(b_0, b_1, \ldots, b_7)$ como un número binario de 8 bits.

Este número permite describir completamente la tabla de reglas de un *ECA*. Dado que con 8 bits se pueden representar 256 valores, este es el número de *ECA*'s posibles. De esta forma cuando se hable del *ECA* 255, sabremos que en la tabla anterior se debe reemplazar $(b_0, b_1, \ldots, b_7)$ por $(1, 1, 1, 1, 1, 1, 1, 1)$ Este esquema de numeración fue propuesto por Wolfram en 1983.

En la Fig. 6.2 se muestra el *ECA* 110 y una aplicación de la función de transición.

000	001	010	011	100	101	110	111
0	1	1	0	1	1	1	0

t = 0	0	1	1	0	1	0	0	1	0	1
t = 1	1	0	1	1	1	1	1	1	1	0

Tabla 6.2: ECA 110: se muestran, en primer lugar la tabla de reglas correspondiente y luego el cambio de estados desde el instante $t = 0$ al $t = 1$.

Cada regla de la Tabla 6.1 puede ser considerada como un par *(detector, efector)*, correspondiendo el *detector* a la vecindad y el *efector* al estado de actualización.

La sucesiva aplicación de la función de transición permite al *CA* realizar cierto "trabajo" o computación. Una tarea posible sería utilizar un *CA* binario tal que dada una configuración inicial C_{inic}, luego de cierto número de iteraciones, llegue a una configuración final C_{final} con todas sus celdas en 1 si en C_{inic} había mayoría de 1's. Si en C_{inic} había mayoría de 0's, C_{final} tendrá todas sus celdas en 0.

Este es el llamado "Problema de densidad" y no se conocen reglas que permitan, desde cualquier configuración inicial, decidir si había mayoría de 1's o de 0's.

Si bien este problema es de trivial solución si se utiliza un coordinador global que "cuente" la cantidad de celdas en cada estado y luego decida; resolverlo en base a información solamente local es muy complejo.

Una interesante aplicación de los *ECA*'s es su utilización como generadores de números al azar. Supongamos que el espacio celular tiene K celdas y asociamos cada configuración con un número representado con K dígitos binarios. La configuracón inicial del CA representará la semilla del algoritmo. Luego, en base al número del *ECA* utilizado se generaran diferentes "números aleatorios" en cada configuración.

Si bien es un mecanismo bien simple, la cantidad de series diferentes de números posibles de generar es el producto de las configuraciones iniciales por la cantidad de *ECA*'s posibles; es decir $2^K * 256$. Si $K = 200$ entonces existen del orden de 2^{200} series diferentes y cada una generará tantos números como iteraciones se hagan del *ECA* correspondiente.

La evolución de los *CA*s se ilustra usando diagramas de espacio-tiempo. En general, estos diagramas se utilizan para "seguir" el comportamiento de autómatas unidimensionales, ya que hacerlo en más dimensiones no resulta tan simple.

En las Fig. 6.2 y 6.1 se muestran dos diagramas del autómata *OR*. Este es un autómata unidimensional binario con $r = 1$, donde la regla de update para cada celda consiste en reemplazar el valor actual por el resultado de calcular el *XOR* ("OR exclusivo") entre su vecino derecho y el izquierdo. En los diagramas, el color negro representa las celdas con estado 1 y el blanco las celdas con 0.

En la Fig. 6.2 la configuración inicial del autómata tiene una sola celda en 1 y las restantes en 0, lo que da lugar a al tipo de gráfico allí mostrado.

En la Fig. 6.1 se aplica la misma regla sobre una configuración inicial generada al azar. A pesar de lo "caótico" del dibujo puede detectarse la presencia de algunos patrones bien definidos.

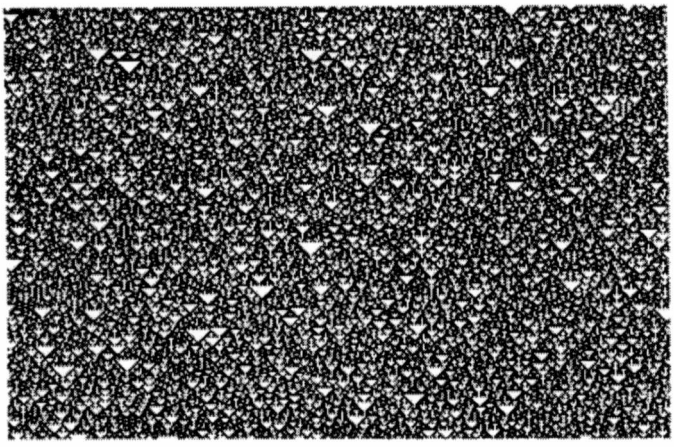

Figura 6.1: Diagrama de espacio-tiempo asociado al autómata OR. El color blanco corresponde a celdas con valor 0 y el negro, a aquellas con valor 1. La configuración inicial es generada al azar.

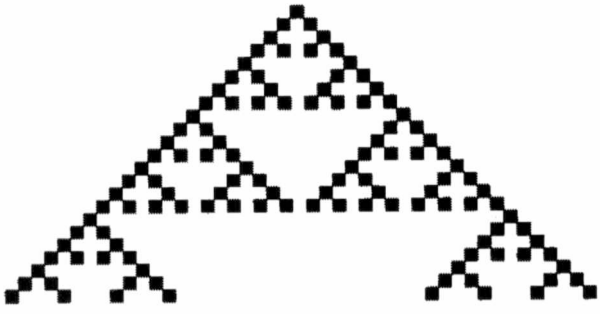

Figura 6.2: Diagrama de espacio-tiempo asociado al autómata OR. El color blanco corresponde a celdas con valor 0 y el negro, a celdas con valor 1. La configuración inicial tiene una sola celda con valor 1.

6.3 Autómatas celulares y su poder computacional

La idea de autómata celular está asociada con la figura del matemático John von Neumman. Él estaba interesado en la posible conexión entre la biología y la teoría de autómatas. Intrigado además por el fenómeno biológico de la auto-reproducción de algunos organismos, formuló la siguiente pregunta: *Qué clase de organización lógica es suficiente para que un autómata sea capaz de autoreproducirse?*.

La idea de utilizar autómatas celulares le fué sugerida por Stanislaw Ulam. Von Neumman creía que una teoría general de la computación basada en "redes complejas de autómatas", como los *CA*, sería esencial para la comprensión de los sistemas complejos naturales y para la simulación de sistemas complejos artificiales.

Finalmente, Von Neumman pudo construir un autómata celular capaz de autoreproducirse. Este autómata consistía de un espacio celular bidimensional, con una regla de transición particular y requería una configuración inicial especial. La cantidad de estados posibles era de 29.

Este autómata, era en realidad un constructor universal ya que permitía generar

copias de cualquier otro autómata cuya descripción estuviera almacenada en su "cinta". Algunos elementos de este desarrollo sirvieron de base para el desarrollo de la máquina *EDVAC*, la primera computadora electrónica con un programa almacenado.

Von Neumman fue más allá y demostró que su autómata podía simular una máquina de Turing Universal. A partir de este resultado, se dice que los autómatas celulares son capaces de realizar computación universal.

La historia de este proyecto puede encontrarse en *Theory of Self-Reproducing autómata*, libro comenzado por Von Neumman y finalizado por A. W. Burks, luego de la muerte en 1957 del primero [Neu66].

Posteriormente se desarrollaron construcciones alternativas a la de Von Neumman.

Uno de los primeros ejemplos fue el de Smith en 1970, que construyó una serie de autómatas celulares simples capaces de lograr computación universal. El primero de la serie era un autómata celular de dos dimensiones que simulaba una máquina de Turing en tiempo real [Smi71].

Un ejemplo muy conocido de autómata celular es *Life* o "El juego de la vida" inventado por John Conway en los sesenta.

Life es un autómata celular binario de dos dimensiones donde la vecindad de cada celda i es la llamada *vecindad de Moore* (el estado de update de cada celda i depende de sus ocho vecinos topológicos).

La regla de transición $\Phi(\eta_i^t)$ es muy simple, si $S_i^t = 1$ entonces $S_i^{t+1} = \Phi(\eta_i^t) = 1$ si y solo si dos o tres vecinos estan en estado 1, sino $S_i^{t+1} = \Phi(\eta_i^t) = 0$.

Si $S_i^t = 0$, entonces $S_i^{t+1} = \Phi(\eta_i^t) = 1$ si y solo si exactamente dos o tres vecinos estan en estado 1, sino $S_i^{t+1} = \Phi(\eta_i^t) = 0$.

Se asume que el espacio celular es infinito y comienza con una configuración que contiene un numero finito de 1s, y todas las otras celdas en 0.

La ejecución del autómata produce el surgimiento en el reticulado de patrones complicados e interesantes que pueden ser asociados con las funciones logicas *NOT, AND, OR.*

A partir de estas funciones y algunos mecanismos adicionales se muestra que *Life* es capaz de computar cualquier función recursiva. En base a estos elementos y considerando que los recursos espacio - tiempo son ilimitados, se demuestra que *Life* es capaz de realizar computación universal.

El trabajo completo respecto a este tema se puede encontrar en [BCG82]

6.4 Dinámica y computación en *CA*

Existe un creciente interés en las relaciones entre el comportamiento dinámico y general de los autómatas celulares y sus habilidades computacionales. Esta cuestión se da en un marco más general donde se analizan y plantean las relaciones existentes entre la teoría de sistemas dinámicos y la teoría de la computación.

Viendo a los autómatas celulares como discretos, Stephen Wolfram, en [Wol84] propuso una clasificación cualitativa del comportamiento de los *CA*, fuertemente análoga a la existente en la teoría sobre sistemas dinámicos.

Las cuatro clases de comportamiento son:

Clase 1: casi todas las configuraciones iniciales tienden a una configuración fija (por ej: todos unos), luego de cierto período de tiempo.

Clase 2: transcurrido cierto tiempo, casi todas las configuraciones iniciales tienden a puntos fijos o a un ciclo temporalmente periódico de configuraciones, las cuales dependen de la configuración inicial.

Cabe destacar que en reticulados finitos, existe un número finito de posibles configuraciones (2^N para un *CA* binario con N celdas), es por eso que todas las reglas finalmente tienen un comportamiento periódico. Esta clase no se refiere a este tipo de comportamiento periódico sino a ciclos con períodos mucho más cortos que 2^N.

Clase 3: casi todas las configuraciones iniciales transcurrido cierto tiempo tienden a un comportamiento caótico (el término caótico se refiere a un comportamiento espacio-temporal aparentemente impredecible).

Clase 4: Algunas configuraciones iniciales resultan en complejas estructuras que muchas veces permanecen fijas por largo tiempo. Se plantea que el *ECA* 110 tiene este comportamiento.

Wolfram (1984) especulaba que todos los *CA*s de *Clase 4* (excepto los *ECA*s) son capaces de realizar computación universal. Sin embargo no existe un método general para probar que una regla dada sea capaz o no de realizar computación universal, por lo tanto esta hipótesis es imposible demostrar.

Otro enfoque para comprender la relación entre la teoría de sistemas dinámicos y la computación en *CA* consiste en clasificar diferentes patrones que surgen en los diagramas de espacio-tiempo del *CA* analizado.

Estos patrones pueden ser definidos en términos de lenguajes formales y sus correspondientes autómatas reconocedores llevando en última instancia a una descripción de la "computación" realizada por el autómata.

Todos estos elementos forman parte del *Computational Mechanical FrameWork* desarrollado por Crutchfield en [Cru94] y cuyo estudio está fuera del alcance de esta tesis.

6.5 Conclusiones

En este capítulo se presentaron los autómatas celulares, se detallaron sus principales características y se comentaron algunos resultados indicando que los *CA* son tan poderosos, computacionalmente hablando, como una Máquina de Turing Universal.

Sugerimos al lector interesado en profundizar sobre la teoría de autómatas celulares, la visita a la página en Internet del Instituto Santa Fe, New Mexico (USA), que se encuentra en la dirección `http://www.santafe.edu`y la consulta de los artículos y libros citados en el presente capítulo.

Capítulo 7

Autómatas Celulares para Protein Folding

7.1 Introducción y Objetivos

En el capítulo 5 hemos descripto como la potencialidad de los algoritmos géneticos puede ser desaprovechada cuando no se tiene en cuenta el alto grado de relación existente entre el operador de crossover utilizado y la codificación de los individuos.

Para el caso de los *AG* para *PF*, esta situación puede solucionarse mediante la utilización de un crossover diseñado *ad hoc* o planteando un cambio en la representación de los individuos.

En este capítulo tomaremos esta segunda opción, lo que implicará explorar un espacio distinto del espacio de conformaciones. En nuestro caso exploraremos el espacio de programas que "producen" conformaciones.

En el capítulo 6 hemos descripto las principales características de los autómatas celulares, su capacidad computacional y su habilidad para mostrar comportamiento emergente a partir de interacciones locales.

En esta sección mostraremos por primera vez en la literatura, según nuestro conocimiento, la aplicación de autómatas celulares para *PF* en el modelo de Dill.

Utilizaremos un autómata celular circular unidimensional, donde cada configuración representará una estructura (probablemente no self-avoiding) en un reticulado bi-dimensional.

La aplicación de las reglas, que permitirán pasar de una configuración a otra, simulará el proceso de plegado. El estado final del autómata representará la estructura final de la proteína.

El objetivo principal es verificar si existe y es posible deducir una serie de reglas para el autómata celular que permitan obtener la estructura terciaria de una proteína partiendo de alguna conformación arbitraria.

En última instancia pretendemos obtener un conjunto universal de reglas tales que, al aplicarlas sobre cualquier instancia en cualquier estado, le permitan alcanzar el estado nativo correspondiente. Este último punto es muy ambicioso, pero creemos que este trabajo puede sentar las bases para futuros desarrollos en este sentido.

En el caso ideal las reglas deducidas deben provocar, a partir de interacciones

locales, el surgimiento de interacciones globales entre partes lejanas de la secuencia.

En el Cap. 2 se describió la *hipótesis dinámica* en términos de una secuencia $\mathcal{U} \Rightarrow \mathcal{I}_1 \Rightarrow \mathcal{I}_2 \ldots \Rightarrow \mathcal{I}_k \Rightarrow \mathcal{N}$. Si se asocia cada configuración intermedia del autómata con algún $\mathcal{I}_j$ se probaría, al menos para el modelo de trabajo, la validez de esta hipótesis ya que se estaría descubriendo cuales son los estados intermedios en el proceso de folding.

En las secciones siguientes, se describe el autómata celular utilizado, se propone un mecanismo para generar las reglas y se desarrollan los experimentos para comprobar su efectividad desde varios puntos de vista.

7.2 Descripción del Automata Celular

Planteamos la utilización de un autómata celular circular unidimensional, donde cada configuración representará una estructura (probablemente no self-avoiding) en un reticulado bi-dimensional.

Cada celda del autómata es un $w \in \{1, 2, 3, 4\}$, representando los movimientos relativos {Arriba, Abajo, Izquierda, Derecha} respectivamente.

Las reglas para el autómata son palabras del lenguaje definido por la siguiente gramática:

```
          S := <Rules>;
    <Rules> := <aRule> <Separator> <Rules> | <aRule> <Separator>;
    <aRule> := <Dirs><Dirs><Dirs><Dirs><Dirs><Dirs><Dirs> "*" <Dirs> |
               <Z><Dirs><Dirs><Dirs><Dirs><Dirs><Z> "*" <Dirs> |
               <Z><Z><Dirs><Dirs><Dirs><Z><Z> "*" <Dirs> ;
        <Z> := "0";
<Separator> := "!";
     <Dirs> := "1" | "2" | "3" | "4" ;
```

El símbolo 0 en la regla, indica *don't care* y se utiliza por cuestiones de implementación.

Con esta gramática es posible obtener reglas de radios {1,2,3} donde mayor radio representa mas especificidad.

En la Fig. 7.1 se muestra un ejemplo de la utilización del *CA* propuesto. En primer lugar se describen las 4 reglas a utilizar, la instancia a procesar y el estado inicial del autómata.

Las celdas del autómata aparecen numeradas del 1 al 11. Dado que el mismo es circular y el máximo radio para las reglas es 3, se copian las tres primeras celdas al final, y las 3 últimas al comienzo.

En cada paso se describen las celdas que serán actualizadas y mediante qué regla. Notar que en el paso 2, donde se aplicó la regla R1, se podría haber aplicado la regla R4.

Sin embargo, según el criterio de elección de reglas que hemos adoptado, se aplica siempre la regla más específica de las disponibles. Es decir, la de mayor radio.

En la Fig. 7.2 se muestran graficamente las estructuras asociadas a cada paso intermedio del autómata de la Fig. 7.1.

En principio no existe ninguna forma de determinar cual debe ser el máximo radio a definir. Sin embargo, vale la pena observar que con radio 3 (lo cual implica observar

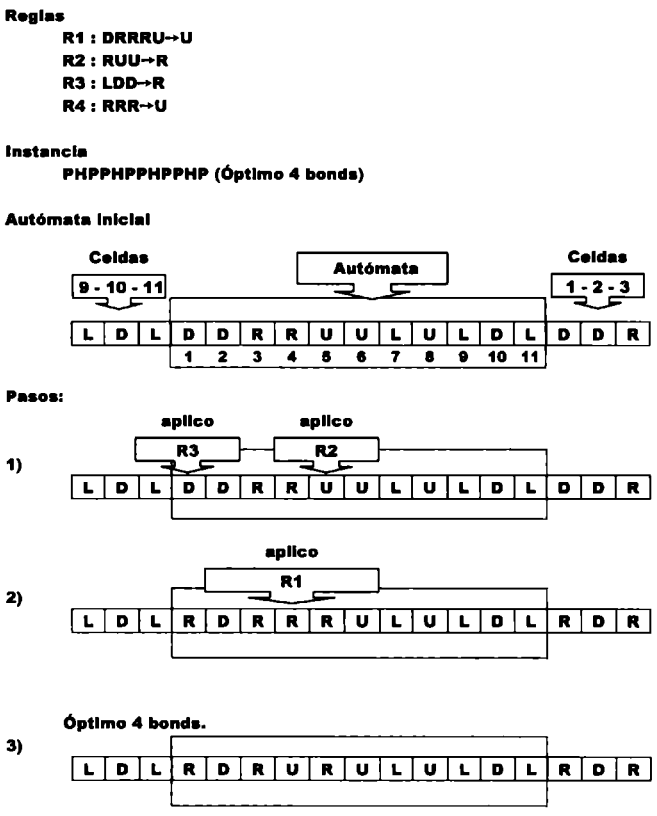

Figura 7.1: Ejemplo de utilización del *CA* para *PF*. Se describen las 4 reglas a utilizar, la instancia a procesar y el estado inicial del autómata. En cada paso se describen las celdas que serán actualizadas y mediante qué regla.

7 aminoácidos), se pueden construir reglas que "produzcan" las estructuras necesarias para soportar los "folding patterns" (estructuras localmente óptimas presentadas en [KLMP97]).

Dado que se pueden construir 4^4 reglas de radio 1, 4^6 reglas de radio 2 y 4^8 reglas de radio 3, existen aproximadamente 70000 reglas que formarán el espacio de búsqueda a explorar.

Dado el tamaño de este conjunto, es impensable intentar deducir por métodos manuales cuáles reglas son necesarias para simular el proceso de plegado.

Por lo tanto, utilizaremos un algoritmo genético que nos ayude a evolucionar conjuntos de reglas para los CA's.

En [MCD96] también se utilizan *AG*'s para evolucionar CA que resuelvan el problema de densidad y el problema de sincronización.

El *AG* propuesto debe funcionar "correctamente"; es decir no debe tener los problemas asociados a la representación y el uso del crossover planteados en el capítulo 5.

Debe quedar en claro que estaremos evolucionando "programas" para doblar pro-

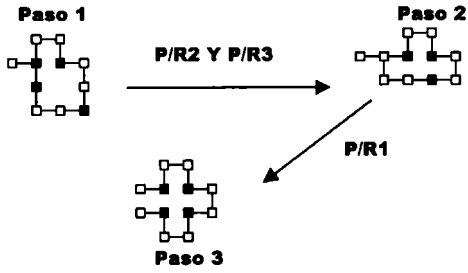

Figura 7.2: Representación gráfica de las estructuras asociadas a cada paso intermedio del autómata de la Fig. 7.1.

teínas; por lo tanto esta parte del trabajo puede ser planteada como una aplicación de la Programación Genética (técnica descripta en el Cap. 3).

7.3 Características del Algoritmo Genético

Hemos planteado la utilización de un *AG* como método automático para la deducción de reglas; a continuación presentaremos sus principales características.

7.3.1 Individuos

Cada individuo de la población, representa un conjunto de reglas, es decir, un conjunto de palabras w generadas por la gramática anterior. Ejemplos de individuos son:

$I_1 \rightarrow$ 4443443*4!2433432*4!0024300*2!3243244*2!4442333*3!
$I_2 \rightarrow$ 0031200*3!0423440*3!0043400*2!0034200*3!
$I_3 \rightarrow$ 0342340*3!0043400*2!0021300*2!0031200*3!0042200*4!4334131*3!

En la descripción de los experimentos se reporta el tamaño de los individuos

7.3.2 Operadores de Mutación

Una vez que se decide mutar a un individuo, se eligen al azar que regla será mutada y con que tipo de mutación. Las mutaciones utilizadas son las siguientes:

Mutar Efector : cambia el efector de la regla
Mutar Detector : cambia algún valor en el detector $\neq$ `don't care`
Reverse Detector: reemplaza el detector por su reverso

7.3.3 Función de Evaluación

La evaluación de un individuo I_j involucra: ejecutar el autómata con las reglas del individuo, y evaluar la energía de la proteína sobre la estructura representada por la configuración final del autómata. El pseudocódigo de la función de evaluación se

muestra en la Fig 7.3

```
ReglasIndiv = Tomar reglas del individuo I_j;
Automata → SetearReglas(ReglasIndiv);
Automata → Ejecutar()
estructura = Tomar configuración final del Automata;
Proteína → Embeber(estructura);
energía = Proteína → Evaluar()
I_j → ValorFitness = energía
```

Figura 7.3: PseudoCódigo de la función de evaluación para los individuos

La función para calcular la energía de una proteína P es:

$$E_P = \alpha * Bonds - \beta * Colisiones + \gamma * Superficie \tag{7.1}$$

donde $\alpha = 2$, $\beta = 6$ y $\gamma = 0.5$ (estos valores fueron fijados empíricamente en base a [KPL+98, KPLdlC97b]

Los *Bonds* se calculan siguiendo la Eq. 4.1. El segundo término, $\beta * Colisiones$, cuantifica la cantidad de colisiones existentes en la conformación, es decir la cantidad de posiciones del reticulado ocupadas por más de un aminoácido. De esta forma, admitimos la existencia de conformaciones con cruces pero las penalizamos.

El tercer término, $\gamma * Superficie$, da cierta medida respecto a cuán compacta es la conformación. En la Eq. 7.2 se muestra la fórmula utilizada para calcularla.

$$Superficie = min(\Delta_x, \Delta_y)/max(\Delta_x, \Delta_y) \tag{7.2}$$

donde Δ_x, Δ_y representan la máxima expansión en x e y respectivamente. Intuitivamente Δ_x, Δ_y representan las longitudes de los lados de un cuadrilátero imaginario (*Bounding Box*) que contiene la estructura considerada. Vale que $0 \leq Superficie \leq 1$; es decir, cuanto más "cuadrada" es la conformación, mayor es el valor de $Superficie$. Por ejemplo, en la Fig. 7.4 se muestran dos conformaciones. En la primera, $\Delta_x = 3$, $\Delta_y = 3$ y $Superficie = 3/3 = 1$ y en la segunda $\Delta_x = 3$, $\Delta_y = 4$ y $Superficie = 3/4 = 0.75$

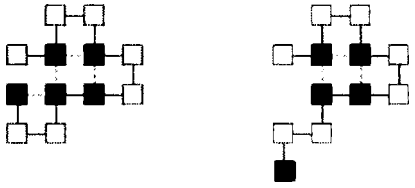

Figura 7.4: Conformaciones con valores de *Superficie = 1* y *Superficie = 0.75* respectivamente.

7.3.4 Selección y Crossover

La selección se realiza utilizando el *muestreo estocástico con reemplazo* (Método de la Ruleta con un solo puntero), en base a los fitness normalizados de los individuos.

El fitness normalizado de un individuo I_j es: $f_{n_i} = f_i/F$ donde f_i es el fitness obtenido mediante la aplicación de la Eq. 7.1 al i-esimo individuo y $F = \sum_{i=1}^{i=N} f_i$

Vale la pena aclarar, que para normalizar el fitness de esta manera, es necesario que los $f_i \geq 0$. La fórmula 7.1 puede retornar valores negativos, por lo que es necesario aplicar un procedimiento de desplazamiento a los valores de fitness, previo a la normalización.

El crossover utilizado es *2 Point-Crossover* y se aplica de la siguiente manera:

```
elegir padre 1 P1
si hay que cruzarlo (en base a la probabilidad de crossover)
   elegir padre 2 P2
   H12 = 2-Point Crossover(P1,P2)
   agregar H12 a la nueva población
si no, agregar P1 a la nueva población
```

donde elegir un individuo, implica "tirar" la ruleta.

Abajo podemos ver un ejemplo de 2-Point Crossover

```
P1 = 0423330*4!4443443*4!2433432*4!0024300*2!3243244*2!4442333*3!
P2 = 0031200*3!0423440*3!0043400*2!0034200*3!
H12 = 0423330*4!4443443*4!0043400*2!0034200*3!3243244*2!4442333*3!
```

El H_{12} recibe las reglas 1, 2, 5 y 6 del padre P_1, mientras que la 3 y la 4 del padre P_2

Como se ve en el ejemplo, los individuos son de longitud variable, por lo que hay que poner especial cuidado al momento de seleccionar los puntos de corte. En nuestro caso, estos puntos se eligen sobre el individuo con menos reglas y, como en el ejemplo, el lugar de P_1 siempre es ocupado por el individuo más largo. De esta forma, la cantidad de reglas que un individuo puede llegar a tener está acotada por la cantidad de reglas del individuo más largo existente en la población inicial.

7.4 Detalles de Implementación

La implementación de los algoritmos fue realizada utilizando el lenguaje orientado a objetos C++. El entorno de trabajo fue el *djgpp v2.7.2.1* que incluye el compilador *gcc* para 32 bits; todo el desarrollo fue realizado sobre PC's. La elección del lenguaje y el compilador obedeció a dos razones: satisfacía los requerimientos de velocidad y la portabilidad a plataformas tipo workstations es relativamente simple. Esto es fundamental ya que existe la posibilidad de continuar este trabajo en el exterior.

Los experimentos se corrieron en máquinas cuyas especificaciones variaron entre una 486DX4 con 16Mb de Ram bajo Windows 95, hasta equipos Pentium 166Mhz con 64Mb de Ram bajo cliente de Windows NT[1].

[1] Deseamos agradacer la generosa contribución de la empresa *Pangea* por haber prestado los equipos

7.5 Etapa de experimentación

Se han realizado dos tipos de experimentos con objetivos diferentes:

En primer término se realizaron simulaciones para obtener indicios de la utilidad de los CA como mecanismos para realizar el folding; se deseaba verificar si era posible encontrar, dada una instancia y un estado inicial del autómata, un conjunto de reglas tales que al aplicarlas se obtuviera el estado nativo de la proteína en cuestión.

Luego, y a consecuencia de los resultados obtenidos, se realizaron experimentos tendientes a optimizar el mecanismo de deducción de reglas y a verificar su correcto funcionamiento. En las secciones siguientes, se describen los experimentos realizados, los resultados generados y las posibles derivaciones e implicancias de los mismos.

7.6 Experimentos Preliminares

Para obtener indicios de la utilidad de los CA como mecanismos para realizar el folding se realizaron varios experimentos.

La idea básica fue correr varias simulaciones del *AG* y observar los resultados obtenidos. Para estas pruebas los parámetros del *AG* se fijaron en función de los valores utilizados en [KPL+98, KPLdlC97b, KPL+97].

Los resultados obtenidos, no reportados aquí, fueron alentadores. En todos los casos de prueba se pudo obtener, para la instancia y el autómata inicial dados, un conjunto de reglas que permitió alcanzar buenos valores de bonds, independientemente de la configuración inicial y siempre fue posible el pasaje de conformaciones inválidas *no self-avoiding* a conformaciones válidas.

Estos experimentos no permiten, en principio, concluir nada acerca de la posibilidad de encontrar un conjunto universal de reglas para simular el proceso de plegado. Lo que si se puede concluir es que es factible encontrar reglas que permiten obtener buenos valores de bonds partiendo de, aparentemente, cualquier conformación inicial.

El paso siguiente fue ajustar el *AG* como una forma de minimizar el esfuerzo necesario para conseguir las reglas. Los experimentos realizados se muestran en la sección siguiente.

7.7 Buscando los mejores parámetros para el *AG*

Es ampliamente aceptado que uno de los factores que condiciona la calidad de un algoritmo génetico es la elección de los valores para los parámetros (probabilidades, tamaño de la población, criterio de parada, etc.).

En primer lugar, se corrieron simulaciones exhaustivas para determinar los mejores valores de probabilidades de crossover (P_x) y de mutación (P_m).

Para ello, se eligieron seis secuencias al azar de longitud 18 de un conjunto de 6399 con óptimo conocido. Estas seis secuencias aparecen en la Tabla 7.1 y para cada una se detallan el código de la misma, la cantidad máxima de bonds B, y la cantidad de estructuras con $B, B-1$ y $B-2$ bonds respectivamente. Como puede observarse, las secuencias elegidas son no degeneradas; es decir, existe una sola estructura que representa el óptimo.

Con cada una de las instancias, y para cada par posible (P_x, P_m) de párametros se corrieron cinco simulaciones. Dado que $P_x, P_m \in [0.1, 0.2, \ldots, 0.9]$, hubo que correr $(5 * (9 * 9) = 405)$ simulaciones para cada instancia.

Cód.	Instancia	Máx. Bonds B	B	$B-1$	$B-2$
249	PHPPHPHHHPHHPHHHHH	9	1	66	911
1243	PHPPHPHPPPHHHHPHHH	8	1	28	514
3730	HPHPHHHPPPHHHHPPHH	8	1	124	1286
4399	PHPPPHPPPHHPPPPPHH	4	1	516	19455
4400	HHPPPPPHHPPPHPPPHP	4	1	516	19455
4761	HPHPHHHPHHPPPPPHHP	6	1	151	6397

Tabla 7.1: Instancias de prueba para las detección de los mejores valores de (P_x, P_m)

El número de generaciones se fijó en 450 y el tamaño de la población en 200 individuos. Cada individuo se generó con un número al azar de reglas que varía entre 1 y 20.

En promedio los individuos tienen 10 reglas; considerando que el total de reglas es del orden de 70000, 200 individuos representan solamente el 0.02% del espacio de búsqueda a explorar.

La configuración inicial del autómata celular fue generada al azar para cada par de parámetros de cada simulación, pero evitando que celdas consecutivas representen colisiones triviales. Es decir si la celda c_i = *Arriba* entonces $c_{i+1} \neq$ *Abajo*, y si c_i = *Derecha* entonces $c_{i+1} \neq$ *Izquierda*. Obviamente vale lo mismo para los recíprocos.

En la sección siguiente se describen los resultados obtenidos.

7.7.1 Resultados Obtenidos

En la Sección 7.11 se muestran las tablas con los resultados obtenidos. Cada tabla se corresponde con una instancia; las filas resumen las corridas que permitieron alcanzar los mejores valores de bonds para la instancia correspondiente.

En las dos primeras columnas se muestran los valores de (P_x, P_m) utilizados en esa corrida. Luego se encuentran los valores de Bonds, Colisiones, Superficie y Fitness correspondientes a la configuración inicial del autómata celular.

El campo *Genr* siguiente indica en que generación del *AG* se encontró el mejor individuo de toda la corrida; las 4 columnas siguientes indican los valores de la configuración obtenida a través de la aplicación de las reglas de este individuo. La última columna de las tablas es la distancia de Hamming, calculada como la cantidad de posiciones diferentes entre la configuración inicial y la mejor generada por el autómata.

Si bien este experimento estuvo diseñado para obtener los mejores valores para (P_x, P_m), otros datos pueden ser extraídos de ellos.

El primer elemento que surge al analizar estas tablas es que se pudieron encontrar reglas que permitieron alcanzar buenos valores de bonds. En todos los casos, se obtuvieron configuraciones que se corresponden con el valor de bonds óptimo - 1. Este es un resultado alentador en el sentido de la efectividad del mecanismo de deducción.

Es interesante notar que los buenos valores de bonds podrían haber sido obtenidos como consecuencia de configuraciones iniciales muy buenas; sin embargo, analizando los valores correspondientes a las distancias de Hamming (campo *Hamm.* de las tablas), se puede observar que las configuraciones finales se obtuvieron provocando, en promedio, 5.41 cambios en la configuración inicial, lo que equivale a cambiar el 30% de la configuración inicial. Como casos extremos, aparecen valores de Hamming con

Nro	P_x	P_m
1	0.1	0.1
2	0.1	0.6
3	0.1	0.8
4	0.1	0.8
5	0.2	0.6
6	0.2	0.7
7	0.2	0.8
8	0.3	0.3
9	0.3	0.5
10	0.3	0.5

Nro	P_x	P_m
11	0.3	0.7
12	0.3	0.9
13	0.4	0.2
14	0.4	0.3
15	0.5	0.2
16	0.5	0.4
17	0.5	0.6
18	0.5	0.9
19	0.6	0.1
20	0.6	0.3

Nro	P_x	P_m
21	0.6	0.4
22	0.6	0.4
23	0.6	0.6
24	0.7	0.4
25	0.7	0.6
26	0.8	0.9
27	0.9	0.3
28	0.9	0.5
29	0.9	0.9

Tabla 7.2: Valores de (P_x, P_m) para los cuales se alcanzó el mejor valor en alguna simulación (ordenados por P_x)

1 (el 5.55%) y 9 (el 50%).

Estos datos son los primeros indicios de la robustez del método en el sentido de la independencia de la configuración inicial; sin embargo, nada se puede concluir en este sentido sin realizar un análisis de las reglas involucradas, así como de las configuraciones iniciales generadas. El diseño de experimentos *ad hoc* está previsto fuera del marco de esta tesis.

Otro elemento interesante es que no existe relación aparente entre el número de generaciones necesarias para alcanzar el mejor valor de fitness y la distancia de Hamming asociada. Es decir, la distancia de Hamming no es una medida de cuanto le costó al autómata encontrar esa configuración final. Una medida más apropiada podría ser la cantidad de iteraciones realizadas o el número de reglas aplicadas.

Finalmente resta analizar las combinaciones de valores para P_x y P_m que permitieron alcanzar los mejores resultados.

De los 81 pares posibles, solo 29 de ellos permitieron alcanzar los mejores valores en alguna corrida de alguna instancia. En la Tabla 7.2, se los muestra ordenados por P_x. La primera columna es simplemente un número de orden. En la Tabla7.3 se repiten los datos pero ordenados por P_m.

Observando las tablas, y analizando los pares en términos de cual probabilidad es mayor, se puede establecer el siguiente agrupamiento sobre 29 pares (P_x, P_m):

12/29 casos	$P_m < P_x$
13/29 casos	$P_m > P_x$
4/29 casos	$P_m = P_x$

Para realizar un análisis más detallado de estos datos, se generaron dos gráficos de líneas que se muestran en la Fig. 7.5.

En ambos el eje X representa el número del par considerado y el eje Y el valor de probabilidad correspondiente.

Nro	P_m	P_x
1	0.1	0.1
2	0.1	0.6
3	0.2	0.4
4	0.2	0.5
5	0.3	0.3
6	0.3	0.4
7	0.3	0.6
8	0.3	0.9
9	0.4	0.5
10	0.4	0.6

Nro	P_m	P_x
11	0.4	0.6
12	0.4	0.7
13	0.5	0.3
14	0.5	0.3
15	0.5	0.9
16	0.6	0.1
17	0.6	0.2
18	0.6	0.5
19	0.6	0.6
20	0.6	0.7

Nro	P_m	P_x
21	0.7	0.2
22	0.7	0.3
23	0.8	0.1
24	0.8	0.1
25	0.8	0.2
26	0.9	0.3
27	0.9	0.5
28	0.9	0.8
29	0.9	0.9

Tabla 7.3: Valores de (P_x, P_m) para los cuales se alcanzó el mejor valor en alguna simulación (ordenados por P_m)

En el gráfico de la izquierda, donde los pares se muestran ordenados por P_x, se distinguen tres regiones en función de los valores de P_x que dan lugar a las siguientes conclusiones:

Si $P_x \leq 0.3$	$\rightarrow$	$P_x \leq P_m$ (12 casos)
Si $P_x \in 0.4, 0.5$	$\rightarrow$	en 4/6 casos $P_x > P_m$
	$\rightarrow$	en 2/6 casos $P_x < P_m$
Si $P_x \in [0.6, \ldots, 0.9]$	$\rightarrow$	en 8/11 casos $P_x > P_m$
	$\rightarrow$	en 1/11 casos $P_x < P_m$
	$\rightarrow$	en 2/11 casos $P_x = P_m$

En general se verifica que si $P_x \leq 0.3 \rightarrow P_m \geq P_x$ y que si $P_x \geq 0.4 \rightarrow P_x \geq P_m$ Estos números concuerdan con la idea generalizada que cuando P_x es baja, la P_m debe ser alta para obtener buenos resultados y, cuando P_x es alta, P_m debe ser baja.

Además, y como dato curioso, surge que todos los valores posibles de P_x están representados en los 29 pares que sirvieron para alcanzar los mejores valores en las simulaciones.

El gráfico derecho de la Fig. 7.5 muestra los datos ordenados por P_m. Repitiendo el análisis se detecta el siguiente agrupamiento:

Si $P_m \leq 0.4$	$\rightarrow$	$P_m \leq P_x$ (12 casos)
Si $P_m \in \{0.5, 0.6\}$	$\rightarrow$	en 5/8 casos $P_m > P_x$
	$\rightarrow$	en 2/8 casos $P_m < P_x$
	$\rightarrow$	en 1/8 casos $P_m = P_x$;
Si $P_m \in [0.7, \ldots, 0.9]$	$\rightarrow$	$P_m \geq P_x$ (9 casos)

Claramente se verifica que si $P_m \leq 0.4 \rightarrow P_m \leq P_x$ y que si $P_m \geq 0.5 \rightarrow P_m \geq P_x$

También se verifica que todos los valores de P_m están representados en los 29 pares.

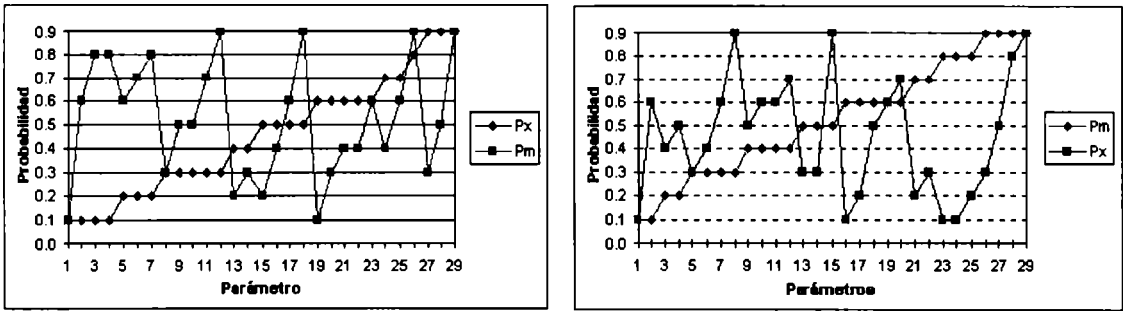

Figura 7.5: Relación entre probabilidades (P_x, P_m) ordenadas primero por P_x y luego por P_m. El eje X corresponde al número de par y el eje Y al valor de probabilidad.

7.8 Evaluación del AG

Los datos presentados en la sección anterior no permiten establecer conclusiones contundentes acerca de cuales son los mejores valores para el AG; sin embargo la siguiente relación de parámetros puede ser establecida: si $P_x \geq 0.6$ entonces $P_m < P_x$ y si $P_m \leq 0.4$ entonces $P_x > P_m$.

En base a estos criterios se seleccionaron 5 valores para (P_x, P_m) que son: $\mathcal{B} = \{(0.4, 0.2), (0.5, 0.4), (0.6, 0.4), (0.7, 0.4), (0.9, 0.3)\}$ y a los cuales identificaremos con P_1, P_2, P_3, P_4, P_5 respectivamente.

En esta sección se describen los experimentos realizados para verificar el comportamiento del mecanismo de deducción en forma más ajustada.

Con este objetivo en mente, se eligieron 9 instancias (mostradas en la Tabla 7.4) diferentes de las utilizadas en el experimento previo, y para cada una de ellas se corrieron 20 simulaciones por cada uno de los cinco parámetros. Es decir, se ejecutaron $9*20*5 = 900$ corridas del AG. Se aumentó el tamaño de la población a 350 individuos y se utilizaron 500 generaciones.

Cód.	Instancia	Máx. Bonds B	B	$B-1$	$B-2$
368	HHHHHPHPPHPHHHHPHH	10	1	29	511
2671	HPHHPPHHHHHHHHHPHH	10	1	162	1795
4731	HPHPHHHHHHHHPHPHPH	10	1	24	325
1710	HPPHPHHHHPPHPHPHHH	9	1	23	268
5619	HPPHPHPHPHHHHHHPHP	9	1	18	316
3545	HHPPPHPPPPHHPPHPHH	7	1	30	447
6318	PHPPHPPHHHHPPHPPHP	7	1	36	766
2666	HHHPPPPHPPPPHPPHPH	6	1	60	1375
4850	HPPHPPHPPPPPHHPHPH	6	1	170	3534

Tabla 7.4: Instancias de prueba para vericar el comportamiento del AG

7.8.1 Resultados Obtenidos

El volúmen de información generado permitió realizar variados análisis con diferentes objetivos.

Las tablas de la Sección 7.11 permiten observar, al igual que en el caso de las simulaciones exhaustivas, que *es posible conseguir conjuntos de reglas que permiten pasar de algún estado inicial particular a conformaciones con buenos valores de bonds.* Para 2 de las 9 instancias, las reglas obtenidas permitieron alcanzar el valor óptimo; para 5 de 9 instancias se pudo alcanzar el valor óptimo-1 y en los casos restantes (2/9) el valor hallado se corresponde con el óptimo-2.

Para determinar si alguno de los 5 parámetros elegidos era más efectivo que los otros, se realizó el siguiente procedimiento: dada las 100 corridas de una instancia (20 por cada parámetro * 5 parámetros), se filtraron los casos donde el mejor valor de bonds se había alcanzado y se registraron los datos de esos casos. Este procedimiento se repitió para las nueve instancias y de esta forma se obtuvieron 18 corridas o casos. En ellas se analizó cuantas veces aparecía cada parámetro, y sorpresivamente se obtuvo que todos los parámetros, salvo 1, se habían utilizado cuatro veces. La excepción se dió con el valor $P_2 = (0.5, 0.4)$ que solo fue utilizado en 2 de las 18 corridas.

Esto nos permite concluir que el algoritmo se comportó de manera robusta frente a la variación de los parámetros (P_x, P_m).

El paso siguiente fue analizar la performance global del AG. En las Tablas 7.5 y 7.6 se muestran dos análisis diferentes sobre las 900 simulaciones en conjunto.

El primero de ellos, resume cuantas corridas permitieron alcanzar valores significativos de bonds (hasta el valor óptimo-3) sin hacer distinción de la instancia involucrada. Los datos para cada valor se muestran en la Tabla 7.5. La suma de los casos correspondientes al óptimo, óptimo-1, óptimo-2 y óptimo-3 (349 casos), representa el 38.77% del total de 900 casos. Es decir, aproximadamente el 40% de las veces, el algoritmo encontró reglas que permitieron alcanzar buenos valores de bonds.

Estos valores no son muy precisos en el sentido que no es lo mismo encontrar una conformación con el valor óptimo-3 en una instancia cuyo óptimo es 10 bonds, que en una instancia cuyo óptimo es de 6 bonds. En un caso estaríamos dentro del 70% del óptimo y en el otro en el 50%.

Para realizar un segundo análisis, más ajustado, se decidió agrupar las instancias según el valor óptimo de bonds para verificar si existe alguna relación entre este valor y los resultados generados por las simulaciones.

Se armaron cuatro grupos G_{10}, G_9, G_7, G_6 correspondiendo a las instancias de 10, 9, 7 y 6 bonds respectivamente. El G_{10} representa tres instancias y los restantes, dos instancias. Luego, dentro de cada grupo se contaron la cantidad de simulaciones que dieron como resultado valores de bonds mayores o iguales al 70% del óptimo correspondiente.

De esta forma, en G_{10} se contaron cuantas de las 300 simulaciones (3 instancias * 100 corridas c/una) alcanzaron 7 o más bonds; en G_9 se contaron cuantas de las 200 simulaciones (2 instancias * 100 corridas c/una) alcanzaron 6 o más bonds; en G_7 se contaron cuantas de las 200 simulaciones (2 instancias * 100 corridas c/una) alcanzaron 5 o más bonds; y en G_6 se contaron cuantas de las 200 simulaciones (2 instancias * 100 corridas c/una) alcanzaron 4 o más bonds.

En la Tabla 7.6 se muestran la cantidad de casos encontrados para cada grupo.

Valor Buscado	Cant. Simulac.
Óptimo	2
Óptimo-1	15
Óptimo-2	95
Óptimo-3	237
Otros	551

Tabla 7.5: Distribución de los 900 casos según valores de bonds. Se detallan, para cada valor, cuantas simulaciones (de un total de 900) permitieron alcanzarlo.

En total se obtuvieron 228 casos, lo que representa el 25.33% del total (900 casos). El mayor aporte a los 228 casos, proviene del G_{10} con 120 casos (el 52.63%). Además la suma de los casos pertenecientes a los grupos G_{10} y G_9 (180 casos) representan el 78.94% del total.

Estos resultados permiten concluir que, para instancias de igual longitud, el algoritmo mejora su comportamiento en las que el valor óptimo es mayor. Una evidencia adicional en este sentido, es que las dos veces en las que el óptimo fue encontrado, correspondió a instancias de 10 bonds.

Haciendo abuso del lenguaje, podemos plantear que al *AG* le resulta "más fácil" encontrar reglas para "doblar bien" instancias con valores de bonds altos que para instancias con pocos bonds.

Esto puede estar relacionado con el hecho que en las instancias con pocos bonds, estos pueden ser formados por aminoácidos lejanos en la secuencia; por lo tanto no existirán interacciones locales que puedan ser descubiertas y aprovechadas por las reglas. Es decir, el radio para las reglas puede no ser suficiente para detectar estas interacciones. Los resultados obtenidos no permiten establecer conclusiones en este sentido pero creemos que en esta línea puede encontrarse una explicación al por que el AG funciona mejor sobre instancias con valores altos de bonds.

7.9 El *AG* es utilizado en todo su potencial?

En la sección anterior se mostraron los resultados obtenidos al utilizar el *AG* como mecanismo de deducción de reglas. A pesar de los buenos resultados obtenidos y siguiendo con la línea de trabajo mostrada en el cap. 5, deseabamos establecer si el *AG* era utilizado "correctamente"; es decir si solucionaba los problemas asociados a

Grupo	Nro. Casos
G_{10}	120
G_9	60
G_7	17
G_6	31

Tabla 7.6: Efectividad del *AG* en cada grupo. Se detallan, para cada grupo de instancias, cuantas simulaciones permitieron obtener valores de bonds superiores al 70%.

la representación y el uso del crossover planteados en el capítulo 5.

Para verificarlo se realizó el *headless chicken test* propuesto por Terry Jones en [Jon95]. La idea básica de este experimento es comparar el funcionamiento del AG propuesto, contra la misma versión utilizando un "crossover random".

El crossover random utilizado funciona de la siguiente manera:
se elige un padre $P1$ y se genera 1 individuo aleatorio $R1$. Se cruzan $P1$ con $R1$ y se genera un hijo H_1 que se incorpora a la nueva población.

Para las mismas 9 instancias del experimento de la sección anterior (mostradas en la Tabla 7.4) se corrieron nuevamente 20 simulaciones por cada parámetro. Es decir se ejecutaron $9 * 20 * 5 = 900$ corridas del AG con un crossover random, manteniendo la población en 350 individuos y utilizando 500 generaciones.

7.9.1 Resultados Obtenidos

Los resultados obtenidos permiten demostrar que la representación sugerida y el crossover utilizado solucionan los problemas planteados en el Cap. 5.

A continuación se repite el mismo análisis de los datos realizados para el caso del crossover estandar y se realizan las comparaciones correspondientes. Además se comparan los resultados obtenidos para cada instancia en forma separada y en forma grupal. No se muestran las tablas con los resultados de las 20 simulaciones y los 5 parámetros de cada instancia en particular.

El primer dato a comparar es cuantas veces en estas 900 corridas se pudieron alcanzar el valores de bonds superiores al óptimo-3. Los resultados se muestran en la Tabla 7.7, donde se repiten los datos de la Tabla 7.5. Las dos últimas columnas se refieren a la cantidad de casos obtenidos con el crossover estandar (*Cross. Std*) y con el crossover random (*Cross. Rnd*). La cantidad de corridas que permitieron alcanzar valores de bonds superiores al óptimo-3 con el crossover random es mucho menor, 216 casos frente a 349 del crossover estandar. En porcentajes respecto al total de 900 casos, es 24% contra el 38.77% respectivamente

Claramente se distingue como la potencialidad del AG disminuyó con la utilización del crossover random.

También es interesante analizar la cantidad de simulaciones que arrojaron valores de bonds superiores al 70% del óptimo correspondiente.

Se armaron nuevamente los cuatro grupos G_{10}, G_9, G_7, G_6 correspondiendo a las instancias de 10, 9, 7 y 6 bonds respectivamente y se realizó el mismo análisis que para el caso del crossover estandar. Los resultados obtenidos se muestran en la Tabla 7.8.

Valor Buscado	Cross. Std	Cross. Rnd
Óptimo	2	0
Óptimo-1	15	8
Óptimo-2	95	64
Óptimo-3	237	144
Otros	551	684

Tabla 7.7: Distribución de los 900 casos según los valores de bonds hallados y según el tipo de crossover utilizado: estandar (*Cross. Std*) o random (*Cross. Rnd*)

El primer dato que se destaca es que para el crossover random solamente existieron 154 casos que permitieron obtener valores de bonds superiores al 70% del óptimo correspondiente. Este valor representa el 17.11% de los 900 casos contra el 25.33% obtenido con el crossover estandar. La suma de los casos del G_{10} nuevamente aporta la parte más importante a los 154 casos , el 57.14%. En conjunto, el G_{10} y el G_9 representan el 73.37% de los 154 casos obtenidos.

Estos dos resultados concuerdan con los obtenidos a partir de la utilización del crossover estandar; nuevamente el AG, a pesar del crossover random, mejora su comportamiento en las instancias con altos valores de bonds.

Posteriormente, y para sumar elementos que permitieran comprobar la superioridad del AG con el crossover estandar se analizaron los datos de cada instancia por separado.

Definimos $C_{Std} = \{s_0, s_1, \ldots, s_k\}$ como la serie de resultados del AG obtenidos para una instancia mediante la utilización del crossover estandar. Cada c_i indica la cantidad de simulaciones de dicha instancia en las que el mejor valor obtenido fueron i bonds. Naturalmente, el valor k se corresponde con el óptimo de la instancia considerada y $\sum_{i=0}^{i=k} s_i = 100$ ya que por cada instancia se corrieron 100 simulaciones y cada una aporta una unidad a algún s_i.

De la misma forma, $C_{Rnd} = \{r_0, r_1, \ldots, r_k\}$ es la serie de resultados para una instancia que se obtiene a partir del uso del crossover random.

Los histogramas de la Fig. 7.6 muestran las series C_{Std} y C_{Rnd} (rotuladas *Std* y *Rnd* respectivamente) para cada instancia involucrada en el experimento. Sobre el eje X, se indican los valores posibles de bonds para la instancia y sobre el eje Y, cantidad de simulaciones. Por ejemplo para la instancia 1710, cuyo óptimo es 9, se pudieron alcanzar 6 bonds en 11 simulaciones con el crossover random y en 17 simulaciones con el estandar. En la instancia 4731, cuyo óptimo es 10, se pudieron alcanzar 7 bonds en 16 simulaciones con el crossover random y en 21 simulaciones con el estandar.

Si imaginamos curvas de ajuste para cada serie de datos, se podría observar como la curva asociada a C_{Rnd} comienza más a la izquierda que la asociada a C_{Std} y como esta última tiene un sesgo más pronunciado hacia la derecha. De alguna manera, esto nos da indicios respecto a que valores altos de bonds nunca podrían ser conseguidos mediante la utilización de un crossover random. Creemos que la utlización de otros tipos de crossover puede ser evaluada en función de los desplazamientos de estas curvas. Actualmente, ya se encuentran diseñados los experimentos para realizar las mismas pruebas utilizando un crossover de tipo uniforme.

Otro elemento a analizar es el valor de bonds donde las series de datos alcanzan sus

Grupo	Casos Cross. Std	Casos Cross. Rnd
G_{10}	120	88
G_9	60	25
G_7	17	14
G_6	31	27
Total	228	154

Tabla 7.8: Distribución de los casos con valores de bonds superiores al 70% comparando, para cada grupo, los valores obtenidos según el tipo de crossover utilizado

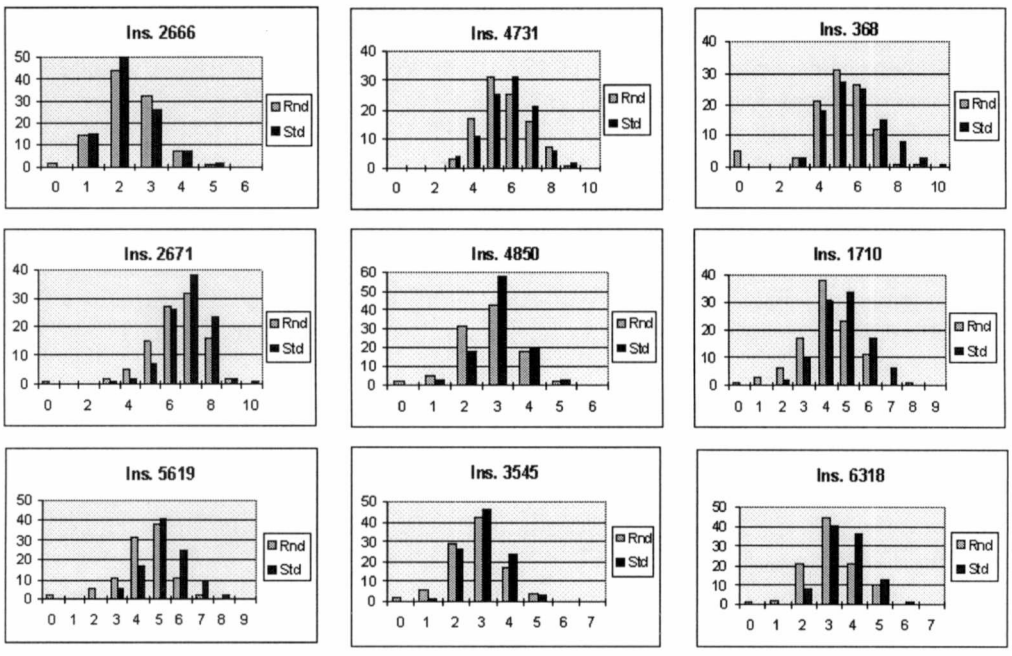

Figura 7.6: Para cada instancia se muestran los resultados asociados a las simulaciones utilizando el crossover estandar vs. el crossover random. El eje X se corresponde con la cant. de bonds y el eje Y, con cantidad de simulaciones.

respectivos máximos. Es decir, donde las curvas imaginarias tienen sus picos. Siendo c_i el máximo de C_{Std} y c_j el máximo de C_{Rnd}, en general se verifica que $i \geq j$ Es decir el máximo correspondiente a C_{Std} se da en valores de bonds mayores o iguales a los de C_{Rnd}

Finalmente, el análisis previo en función de número de bonds y cantidad de simulaciones se realizó utilizando las instancias agrupadas según el valor óptimo de bonds.

Ahora las series de datos son: $C^+_{Std} = \{\sum_{i=1}^{i=n} s_0^i, \ldots, \sum_{i=1}^{i=n} s_k^i\}$ donde n es la cantidad de instancias del grupo y s_i^j es la cantidad de simulaciones para la instancia j que permitieron obtener i bonds con el crossover estandar; de la misma forma $C^+_{Rnd} = \{\sum_{i=1}^{i=n} r_0^i, \ldots, \sum_{i=1}^{i=n} r_k^i\}$ pero con la utilización del crossover random.

Como en la sección previa se generaron los grupos G_{10}, G_9, G_7, G_6 y se graficaron las series C^+_{Std} y C^+_{Rnd} correspondientes (con rótulos *Std* y *Rnd* respectivamente).

Los cuatro histogramas se muestran en la Fig. 7.7. El eje X se corresponde con la cant. de bonds y el eje Y, con cantidad de simulaciones.

Analizando las instancias en conjunto y siendo s_i el máximo de C^+_{Std} y r_j el máximo de C^+_{Rnd}, también se verificó que $i \geq j$.

Luego para cada serie de cada grupo se buscó una curva que se ajustara "bien" a los datos. Una forma de medir cuán bien se ajusta la curva a la serie de datos considerada es medir el coeficiente de correlación R^2 cuyo rango es $0 \leq R^2 \leq 1$. Valores de R^2 cercanos a 1 indican alta correlación y cercanos a 0, baja correlación.

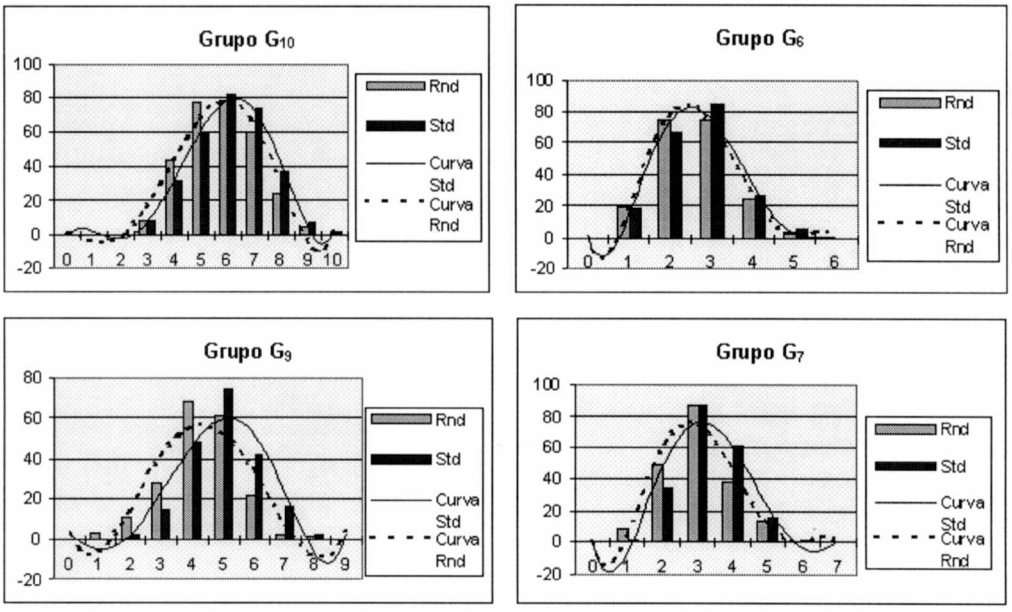

Figura 7.7: Para cada grupo de instancias se muestran los resultados asociados a C_{Std} y a C_{Rnd} con referencias *Std* y *Rnd* respectivamente. El eje X se corresponde con la cant. de bonds y el eje Y, con cantidad de simulaciones. Las curvas de ajuste para cada serie (*Ajuste Std* y *Ajuste Rnd*) son curvas polinómicas de grado 5 con un $R^2 > 0.86$

Utilizando curvas de ajuste polinómicas de grado 5 obtuvimos, sorprendentemente, valores de $R^2 > 0.86$. Las curvas de ajuste para las series C^+_{Std} de los grupos G_{10} y G_6 presentaron un $R^2 = 0.99$.

Las curvas de ajuste para las series random están rotuladas como *Ajuste Rnd* y las del crossover estandar como *Ajuste Std*. Estas curvas permiten observar un "desplazamiento" hacia la izquierda de las curvas *Ajuste Rnd* y un "desplazamiento" hacia la derecha de las *Ajuste Std*. Este comportamiento se visualiza claramente en los grupos G_9 y G_7.

Es claro que los datos obtenidos siguen una distribución bien definida y que no son producto de la "buena suerte". La utilización de técnicas de estadística y teoría de las probabilidades para el análisis de estos datos será una de las tareas a realizar en trabajos futuros.

La conclusión final es contundente: el AG con el crossover estandar funciona mejor que con un crossover random. Es decir, los problemas planteados en el Cap. 5 han sido solucionados.

7.10 Conclusiones

Al comienzo de este capítulo se planteó como objetivo principal de esta parte del trabajo:

> "... verificar si existe y es posible deducir una serie de reglas para el autómata celular que permitan obtener la estructura terciaria de una proteína partiendo de alguna conformación arbitraria."

Sin duda, este objetivo ha sido cumplido. Las simulaciones realizadas han permitido, dado un estado inicial particular y una instancia, obtener reglas mediante las cuales el autómata celular puede alcanzar conformaciones con buenos valores de bonds; incluso el óptimo en algunos casos.

El mecanismo de deducción de reglas propuesto, es por sí solo un buen mecanismo para trabajar sobre *PF*, solucionando además, los problemas asociados a la representación y el uso del crossover planteados en el Cap. 5; esto ha sido demostrado a partir de los experimentos del *HeadLess Chicken Test*. Es decir, hemos dado una representación y un crossover donde, no solo la *Mecánica* del mismo está presente, también queda reflejada la *Idea* del crossover.

Quizá las consecuencias más importantes de estos resultados surjan al considerar el CA, con las reglas deducidas, como un primer modelo para simular el proceso de folding.

Este modelo respeta la idea de la hipótesis dinámica en el sentido que cada configuración intermedia del CA se puede asociar directamente con los estados intermedios del proceso de folding.

La posibilidad de "visualizar" estos estados intermedios será seguramente un elemento fundamental en futuras investigaciones.

En conclusión, consideramos que los resultados obtenidos a partir de los experimentos establecen bases firmes para la futura utilización de los CA's como mecanismos para indagar en la dinámica del proceso de folding.

7.11 Apéndice A

Resultados de las Simulaciones Exhaustivas

En esta sección se muestran los resultados numéricos de los experimentos realizados para buscar los mejores parámetros de (P_x, P_m).

Cada tabla se corresponde con una instancia; las filas resumen las corridas que permitieron alcanzar los mejores valores de bonds para la instancia correspondiente.

En las dos primeras columnas se muestran los valores de (P_x, P_m) utilizados en esa corrida. Luego se encuentran los valores de Bonds, Colisiones, Superficie y Fitness correspondientes a la configuración inicial del autómata celular.

El campo *Genr* siguiente indica en que generación del AG se encontró el mejor individuo de toda la corrida; las 4 columnas siguientes indican los valores de la configuración obtenida a través de la aplicación de las reglas de este individuo. La última columna de las tablas es la distancia de Hamming, calculada como la cantidad de posiciones diferentes entre la configuración inicial y la mejor generada por el autómata.

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.7	0.4	0	1	0.83	-5.58	196	3	0	0.57	6.29	5
0.2	0.8	1	3	0.80	-15.60	344	3	0	0.83	6.42	4
0.6	0.4	0	2	0.57	-11.71	280	3	0	0.83	6.42	8
0.3	0.9	1	1	0.50	-3.75	230	3	0	0.67	6.33	5
0.4	0.2	0	3	1.00	-17.50	14	3	0	0.83	6.42	5

Tabla 7.9: Instancia 4400: HHPPPPPHHPPPHPPPHP. Óptimo: 4 Bonds

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.6	0.3	1	4	0.43	-21.79	182	7	0	0.80	14.40	6
0.3	0.5	3	2	0.67	-5.67	368	7	0	1.00	14.50	6

Tabla 7.10: Instancia 1243: PHPPHPHPPPHHHHPHHH. Óptimo: 8 Bonds

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.1	0.8	3	3	0.43	-11.79	310	7	0	0.67	14.33	9
0.6	0.1	3	0	0.71	6.36	176	7	0	0.67	14.33	5
0.3	0.7	3	1	0.83	0.42	202	7	0	0.57	14.29	6
0.1	0.6	2	4	1.00	-19.50	310	7	0	1.00	14.50	5
0.7	0.6	3	0	0.83	6.42	126	7	0	0.83	14.42	5
0.2	0.7	1	0	0.30	2.15	254	7	0	0.80	14.40	8

Tabla 7.11: Instancia 3730: HPHPHHHPPPHHHHPPHH. Óptimo: 8 Bonds

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.4	0.3	0	1	0.50	-5.75	214	8	0	0.83	16.42	8
0.1	0.1	5	0	0.83	10.42	6	8	0	0.83	16.42	2

Tabla 7.12: Instancia 249: PHPPHPHHHPHHPHHHHH. Óptimo: 9 Bonds

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.9	0.3	2	0	0.71	4.36	10	5	0	0.83	10.42	8
0.5	0.2	1	0	0.88	2.44	304	5	0	0.80	10.40	7
0.5	0.6	2	0	0.86	4.43	72	5	0	0.83	10.42	6
0.8	0.9	2	1	1.00	-1.50	418	5	0	0.83	10.42	6
0.9	0.9	3	3	0.67	-11.67	214	5	0	1.00	10.50	5
0.3	0.5	3	0	0.67	6.33	6	5	0	0.67	10.33	3
0.5	0.4	0	1	0.57	-5.71	400	5	0	0.80	10.40	4

Tabla 7.13: Instancia 4761: HPHPHHHPHHPPPPPHHP. Óptimo: 6 Bonds

P_x	P_m	Bonds	Colis	Sup.	Fit.	Genr	Bonds	Colis	Sup..	Fit.	Hamm.
0.5	0.9	0	6	1.00	-35.50	116	3	0	1.00	6.50	5
0.6	0.4	1	0	0.44	2.22	196	3	0	0.67	6.33	5
0.3	0.3	0	2	0.71	-11.64	118	3	0	1.00	6.50	4
0.9	0.5	3	0	1.00	6.50	0	3	0	1.00	6.50	1
0.1	0.8	1	0	0.5	2.25	68	3	0	0.5	6.25	3
0.2	0.6	0	2	0.27	-11.86	208	3	0	0.67	6.33	7
0.6	0.6	0	3	0.50	-17.75	148	3	0	0.43	6.21	6

Tabla 7.14: Instancia 4399: PHPPPHPPPHHPPPPPHH. Óptimo: 4 Bonds

Crossover estandar: resumen de las simulaciones

En esta sección se describen las 20 corridas realizadas por cada instancia y por cada juego de parámetros utilizando el crossover estandar.

Cada tabla se corresponde con una instancia. Cada fila representa una corrida del AG.

Salteando la primera, las columnas deben leerse de 2 en 2. De esta forma, la columna B_{Pi} indica los bonds obtenidos mediante la utilización del parámetro P_i. La columna siguiente indica en que generación, el AG alcanzó ese valor de bonds.

Los 5 parámetros eran: $\mathcal{B} = \{(0.4, 0.2), (0.5, 0.4), (0.6, 0.4), (0.7, 0.4), (0.9, 0.3)\}$ los cuales se identifican en las tablas como P_1, P_2, P_3, P_4, P_5 respectivamente.

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	4	0	4	28	3	90	4	16	3	26
2	4	18	4	70	4	134	4	92	3	116
3	4	48	4	184	5	22	4	222	4	24
4	4	96	5	38	5	48	5	42	4	114
5	4	124	5	150	5	48	5	46	4	254
6	4	126	5	356	5	80	5	134	4	430
7	4	498	6	10	5	102	5	226	5	12
8	5	22	6	20	5	250	5	232	5	40
9	5	110	6	102	5	266	5	416	5	76
10	5	302	6	116	5	454	5	438	6	30
11	5	324	6	132	6	16	6	12	6	42
12	5	362	6	284	6	110	6	50	6	72
13	5	442	7	122	6	118	6	358	6	100
14	6	166	7	286	6	292	6	376	6	170
15	6	188	7	396	7	10	6	386	6	304
16	6	448	7	494	7	302	7	196	6	480
17	7	8	8	136	7	372	7	324	7	242
18	7	118	8	332	8	88	7	444	7	400
19	7	378	8	374	8	228	8	444	8	186
20	8	170	9	442	9	366	10	268	9	456

Tabla 7.15: Instancia 0368. Óptimo 10 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	2	430	3	44	3	18	3	196	2	40
2	3	2	4	22	4	22	4	12	3	6
3	3	136	4	54	4	32	4	36	3	34
4	3	470	4	106	4	368	4	78	3	54
5	4	34	4	146	4	492	4	96	3	414
6	4	46	4	178	5	54	4	152	4	162
7	4	198	4	178	5	62	4	366	4	176
8	4	210	4	184	5	72	5	36	4	196
9	4	266	4	186	5	78	5	84	5	36
10	4	350	4	216	5	106	5	192	5	82
11	5	106	4	302	5	194	5	202	5	94
12	5	162	4	358	5	274	5	206	5	116
13	5	204	4	392	5	310	5	386	5	174
14	5	262	5	118	5	338	6	22	5	176
15	5	348	5	232	6	82	6	118	5	276
16	5	466	5	310	6	186	6	376	5	276
17	6	180	5	374	6	208	6	416	5	392
18	6	196	6	434	6	238	6	446	6	282
19	7	34	6	470	6	276	6	494	6	350
20	7	252	7	466	7	48	7	100	7	102

Tabla 7.16: Instancia 1710. Óptimo 9 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	3	22	6	110	5	118	4	0	5	196
2	4	494	6	116	6	54	5	358	6	18
3	5	66	6	138	6	104	5	456	6	60
4	5	130	6	162	6	170	6	96	6	74
5	5	388	6	248	6	406	6	256	6	154
6	6	46	6	354	7	36	6	304	6	174
7	6	120	7	8	7	104	7	10	6	268
8	6	132	7	12	7	110	7	62	6	438
9	6	170	7	68	7	118	7	132	6	484
10	6	386	7	84	7	126	7	142	7	142
11	7	86	7	368	7	146	7	170	7	152
12	7	146	7	394	7	156	7	196	7	178
13	7	148	7	424	7	346	7	232	7	178
14	7	244	7	440	7	358	7	396	7	308
15	7	398	8	84	7	414	7	426	7	480
16	8	66	8	150	8	114	8	84	8	238
17	8	198	8	188	8	122	8	100	8	348
18	8	198	8	292	8	236	8	212	8	490
19	8	202	8	302	8	344	8	234	9	216
20	8	444	10	386	8	438	8	394	9	394

Tabla 7.17: Instancia 2671. Óptimo 10 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	1	14	1	6	1	12	1	14	1	152
2	1	40	1	66	1	24	1	186	2	2
3	1	46	1	68	2	16	2	6	2	12
4	1	106	1	306	2	30	2	14	2	32
5	1	148	2	4	2	30	2	18	2	46
6	1	238	2	4	2	30	2	52	2	58
7	2	12	2	28	2	40	2	58	2	100
8	2	40	2	48	2	46	2	58	2	108
9	2	98	2	72	2	68	2	136	2	122
10	2	114	2	100	2	106	2	194	2	202
11	2	192	2	112	2	106	2	252	2	206
12	2	258	3	82	2	138	2	422	2	236
13	2	298	3	90	2	186	2	446	2	272
14	2	476	3	210	2	354	3	38	3	70
15	3	38	3	328	3	78	3	54	3	130
16	3	58	3	344	3	132	3	60	3	186
17	3	102	3	446	3	362	3	92	3	226
18	4	16	3	460	3	464	3	184	3	242
19	4	214	3	462	4	102	4	62	3	268
20	5	56	4	366	4	394	4	272	5	54

Tabla 7.18: Instancia 2666. Óptimo 6 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	2	0	2	2	2	56	2	26	1	38
2	2	0	2	12	2	60	2	66	2	12
3	2	172	2	60	2	92	2	86	2	18
4	2	430	2	64	2	102	2	158	2	50
5	3	12	2	102	2	284	2	300	2	302
6	3	22	2	126	2	324	3	18	2	482
7	3	58	3	4	2	338	3	36	3	144
8	3	76	3	4	3	6	3	56	3	264
9	3	90	3	44	3	18	3	152	3	382
10	3	118	3	52	3	20	3	224	3	392
11	3	122	3	62	3	50	3	238	3	422
12	3	130	3	82	3	94	3	354	3	422
13	3	144	3	118	3	114	3	356	4	166
14	3	206	3	120	3	124	4	84	4	252
15	3	324	3	146	3	292	4	142	4	268
16	3	324	3	336	4	8	4	144	4	284
17	4	196	3	398	4	44	4	172	4	316
18	4	278	3	488	4	100	4	196	4	358
19	4	388	5	162	4	166	4	338	4	392
20	5	110	5	288	4	450	4	344	4	494

Tabla 7.19: Instancia 3545. Óptimo 7 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	3	324	4	0	4	10	4	58	3	10
2	3	486	4	320	4	64	5	22	3	120
3	4	54	5	58	4	314	5	70	4	318
4	4	66	5	138	5	28	5	76	4	460
5	4	164	5	192	5	30	5	76	5	100
6	5	268	5	238	5	74	5	82	5	358
7	5	382	5	476	5	96	5	316	6	66
8	5	476	6	102	5	110	5	446	6	72
9	6	34	6	120	5	244	5	466	6	80
10	6	56	6	162	5	398	6	14	6	96
11	6	248	6	168	6	198	6	78	6	140
12	6	278	6	244	6	222	6	102	6	242
13	6	410	6	272	6	246	6	260	7	32
14	6	436	6	300	6	266	6	350	7	52
15	7	216	6	376	6	412	7	36	7	290
16	7	330	6	408	7	122	7	156	7	316
17	7	426	7	98	7	182	7	228	7	384
18	7	470	7	144	7	238	7	458	7	430
19	8	198	7	292	8	456	7	500	8	444
20	8	352	8	100	9	242	8	36	9	472

Tabla 7.20: Instancia 4731. Óptimo 10 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	1	30	2	6	2	24	3	20	1	58
2	2	14	2	26	2	82	3	22	1	148
3	2	38	2	98	2	140	3	24	2	0
4	2	70	2	132	2	144	3	40	2	26
5	2	100	2	346	3	12	3	44	2	140
6	3	2	3	32	3	20	3	110	2	332
7	3	18	3	40	3	58	3	140	3	20
8	3	48	3	98	3	76	3	144	3	34
9	3	76	3	116	3	88	3	212	3	52
10	3	86	3	120	3	324	3	236	3	56
11	3	102	3	136	3	340	3	242	3	74
12	3	104	3	148	3	460	3	260	3	82
13	3	318	3	190	3	466	3	432	3	144
14	3	378	3	208	4	8	3	460	3	222
15	3	428	3	230	4	202	3	484	3	346
16	3	482	3	300	4	224	4	20	3	358
17	3	498	4	70	4	268	4	46	3	442
18	4	54	4	174	4	302	4	58	4	32
19	4	58	4	194	4	314	4	294	5	402
20	4	120	4	390	5	140	4	350	5	492

Tabla 7.21: Instancia 4850. Óptimo 6 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	3	70	4	154	3	4	3	18	3	144
2	4	118	4	326	4	52	3	474	4	88
3	4	242	5	20	4	192	4	6	4	106
4	4	412	5	98	4	238	4	220	5	30
5	4	412	5	168	4	242	4	238	5	40
6	4	450	5	204	5	18	4	312	5	54
7	5	62	5	418	5	26	5	16	5	76
8	5	80	5	440	5	28	5	48	5	152
9	5	98	6	118	5	54	5	56	5	178
10	5	106	6	150	5	74	5	146	5	492
11	5	122	6	260	5	278	5	158	6	34
12	5	126	6	312	5	294	5	186	6	38
13	5	158	6	316	5	424	5	212	6	46
14	5	168	6	360	6	40	5	250	6	202
15	5	314	6	400	6	64	5	316	6	394
16	6	50	6	432	6	280	5	332	7	184
17	6	144	6	462	6	342	5	494	7	210
18	6	176	7	124	6	344	6	18	7	232
19	7	140	7	334	7	182	6	386	7	254
20	7	338	8	458	8	208	6	474	7	342

Tabla 7.22: Instancia 5619. Óptimo 9 Bonds

Nro Corrida	B_{P1}	Gen	B_{P2}	Gen	B_{P3}	Gen	B_{P4}	Gen	B_{P5}	Gen
1	2	28	2	4	2	76	2	8	3	2
2	2	38	2	76	3	10	3	50	3	18
3	2	458	2	132	3	38	3	70	3	76
4	3	8	3	0	3	40	3	84	3	86
5	3	16	3	32	3	96	3	100	3	92
6	3	26	3	72	3	110	3	140	3	168
7	3	48	3	92	3	114	3	210	3	250
8	3	54	3	108	3	126	3	272	4	50
9	3	70	3	132	3	224	4	8	4	52
10	3	120	3	140	3	488	4	86	4	114
11	3	218	3	330	4	114	4	102	4	188
12	3	242	4	48	4	198	4	124	4	190
13	3	306	4	108	4	266	4	128	4	278
14	4	260	4	108	4	268	4	232	4	334
15	4	268	4	118	4	376	4	420	4	386
16	4	344	4	270	4	400	4	460	4	394
17	4	358	4	424	5	34	5	256	4	408
18	4	430	4	436	5	36	5	326	4	422
19	5	258	5	6	5	200	5	490	5	198
20	6	114	5	68	5	488	5	496	5	412

Tabla 7.23: Instancia 6318. Óptimo 7 Bonds

Representación gráfica de las simulaciones exhaustivas

En esta sección, aparecen representados graficamente los resultados de las tablas de la sección 7.11.

En cada fila de cada tabla aparecen las corridas que permitieron alcanzar el mejor valor en las simulaciones exhaustivas.

Cada uno de los gráficos siguientes representa (de izquierda a derecha y de arriba hacia abajo) la configuración inicial, los estados intermedios, y la configuración final producida en cada una de esas corridas.

Por ejemplo, el Gráfico 7.9 es la representación gráfica de la segunda fila de la Tabla 7.12.

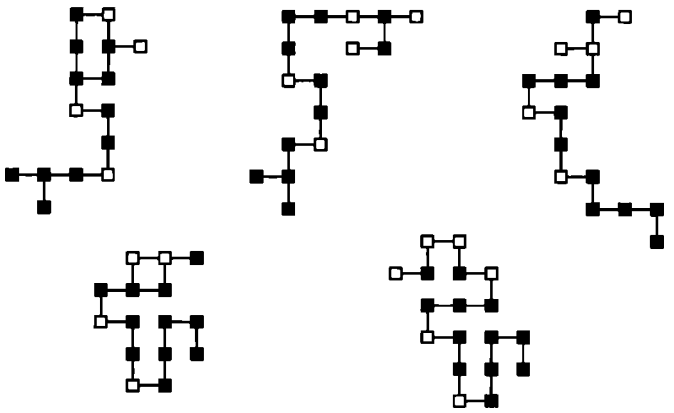

Figura 7.8: Instancia 249 - Corrida 1

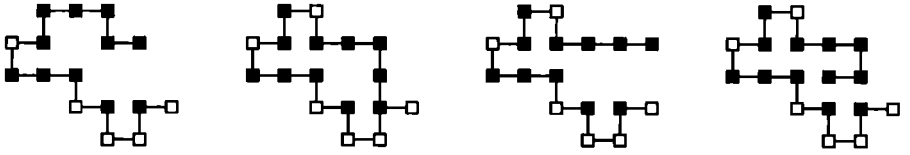

Figura 7.9: Instancia 249 - Corrida 2

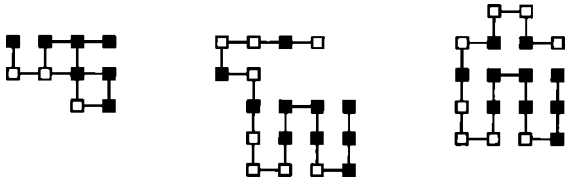

Figura 7.10: Instancia 1243 - Corrida 1

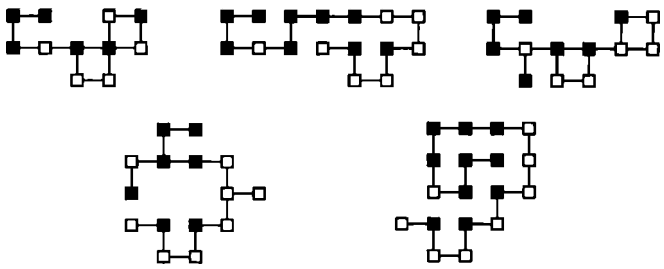

Figura 7.11: Instancia 1243 - Corrida 2

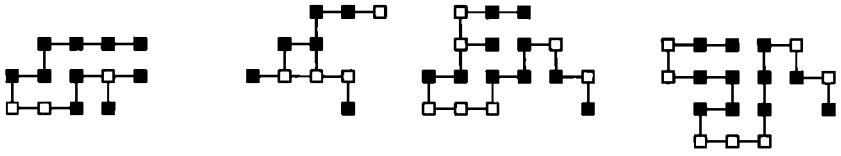

Figura 7.12: Instancia 3730 - Corrida 1

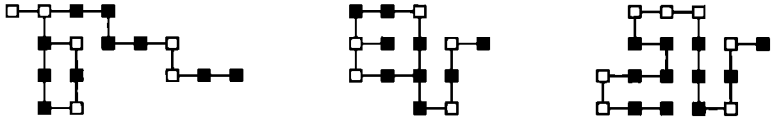

Figura 7.13: Instancia 3730 - Corrida 2

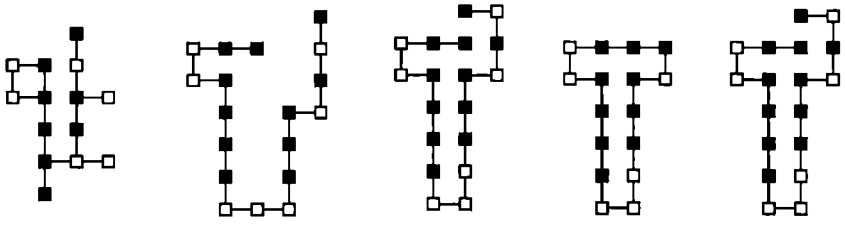

Figura 7.14: Instancia 3730 - Corrida 3

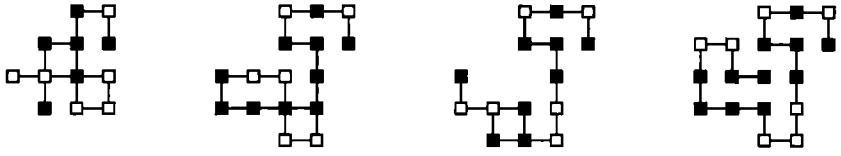

Figura 7.15: Instancia 3730 - Corrida 4

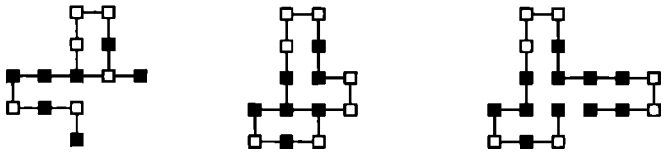

Figura 7.16: Instancia 3730 - Corrida 5

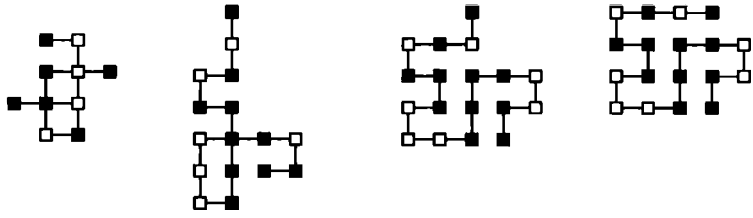

Figura 7.17: Instancia 3730 - Corrida 6

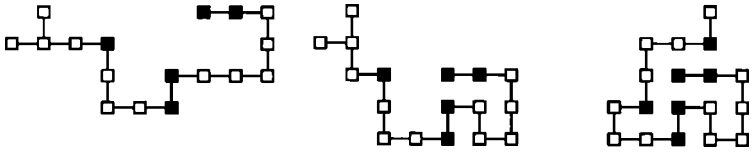

Figura 7.18: Instancia 4399 - Corrida 1

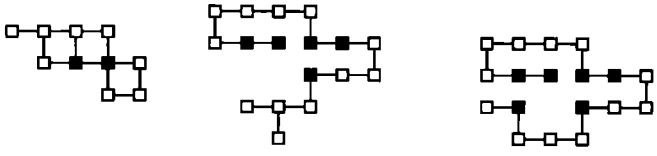

Figura 7.19: Instancia 4399 - Corrida 2

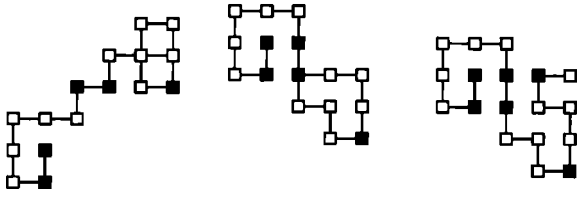

Figura 7.20: Instancia 4399 - Corrida 3

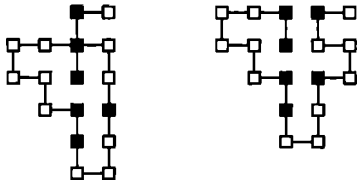

Figura 7.21: Instancia 4399 - Corrida 4

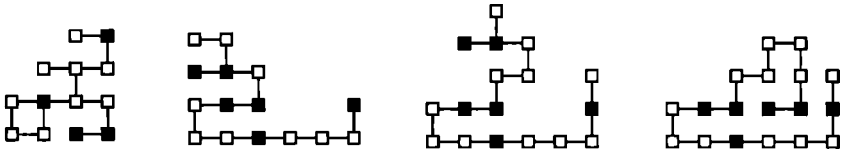

Figura 7.22: Instancia 4399 - Corrida 6

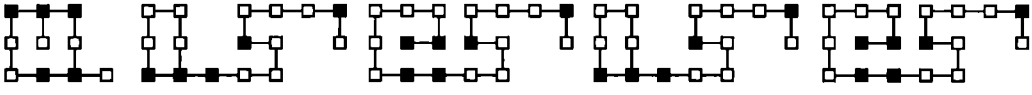

Figura 7.23: Instancia 4399 - Corrida 7

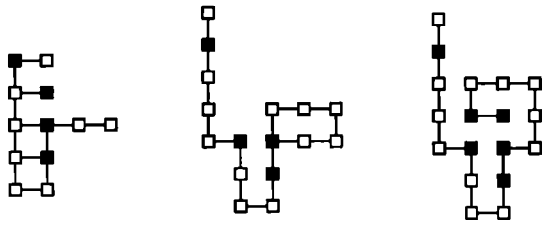

Figura 7.24: Instancia 4400 - Corrida 1

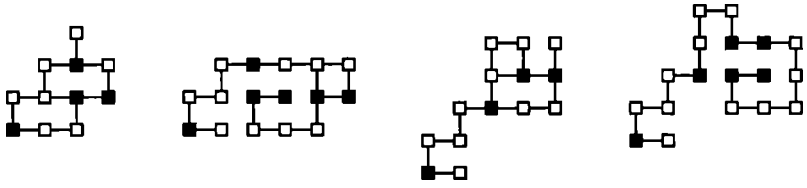

Figura 7.25: Instancia 4400 - Corrida 2

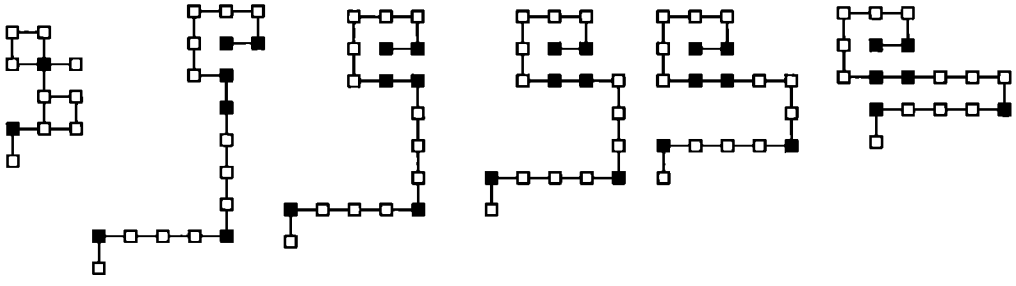

Figura 7.26: Instancia 4400 - Corrida 3

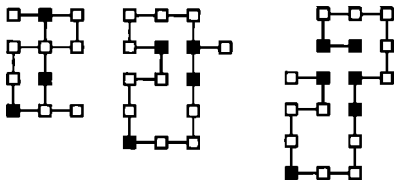

Figura 7.27: Instancia 4400 - Corrida 4

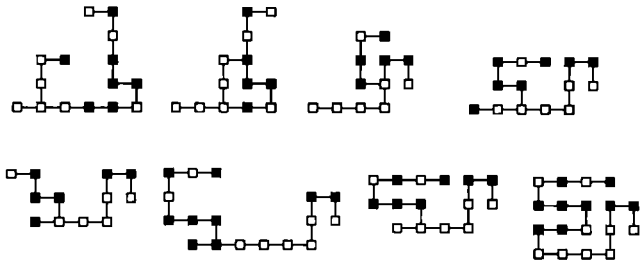

Figura 7.28: Instancia 4761 - Corrida 2

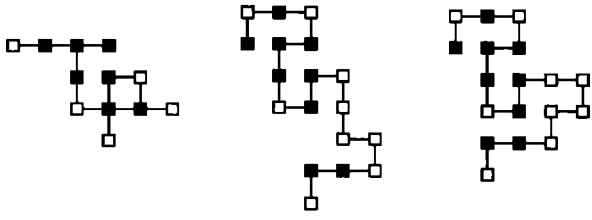

Figura 7.29: Instancia 4761 - Corrida 3

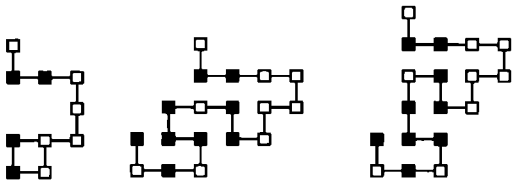

Figura 7.30: Instancia 4761 - Corrida 4

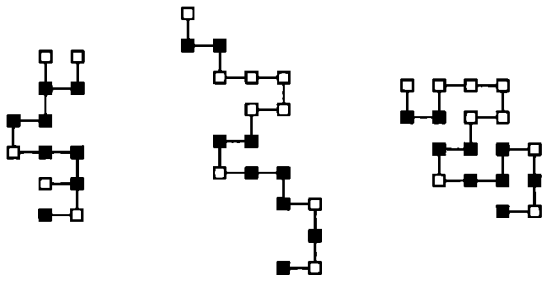

Figura 7.31: Instancia 4761 - Corrida 5

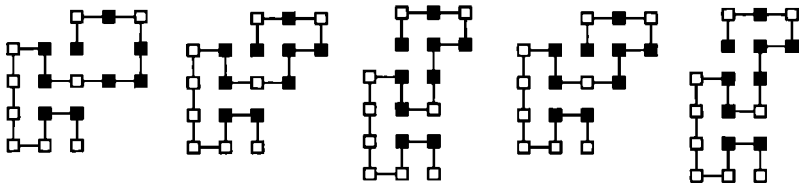

Figura 7.32: Instancia 4761 - Corrida 6

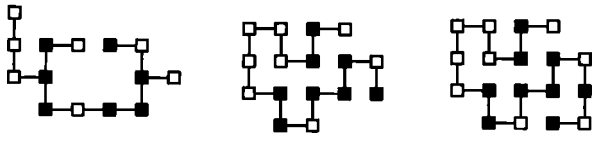

Figura 7.33: Instancia 4761 - Corrida 7

Capítulo 8

Trabajo Futuro y Conclusiones Finales

8.1 Trabajo Futuro

En esta sección se describen las líneas de trabajo que surgen como consecuencia de los resultados obtenidos en el marco de esta tesis.

Quizás uno de los interrogantes que se genera naturalmente respecto a los autómatas y el AG utilizado es: pueden generalizarse las reglas "descubiertas" para ser aplicadas a diferentes instancias y/o sobre diferentes configuraciones iniciales?.

Varias posibilidades surgen en este sentido.

La primera de ellas consiste en mantener el AG en su implementación actual y diseñar estrategias de generación y recolección de reglas que permitan generar subconjuntos de reglas generalizadas para cierto tipo de instancias y/o configuraciones iniciales.

Dentro de esta solución se puede plantear la paralelización del AG y el uso del Modelo de Islas [Mic96] como forma de acelerar el proceso de generación de reglas.

La segunda posibilidad es realizar ajustes al mecanismo de deducción. En este sentido proponemos tres experimentos concretos tendientes a obtener reglas "más generales":

1. En la representación utilizada la posición relativa de cada regla no influye en el comportamiento del CA. Como una forma de mejorar el intercambio de información entre los individuos proponemos la utilización de un crossover uniforme. Una vez implementado, solamente hay que repetir los experimentos realizados con el 2-Point crossover, en sus versiones estandar y random, y comparar los resultados.

2. Sugerimos modificar la función de evaluación de los individuos de la siguiente forma: dada una instancia I y un conjunto de reglas R, ejecutar el autómata partiendo desde k configuraciones iniciales diferentes. El fitness de R podría ser el promedio sobre los valores de bonds obtenidos en cada conformación final alcanzada. De este modo, estamos generalizando las reglas para poder doblar una instancia particular desde cualquier configuración inicial.

3. Otra modificación a la función de evaluación de los individuos es: dada una configuración inicial C_i y un conjunto de reglas R, ejecutar el autómata y evaluar un conjunto de N instancias sobre la estructura final alcanzada. Nuevamente, el fitness de R podría ser el promedio sobre los valores de bonds obtenidos en cada conformación final alcanzada. En esta propuesta, las reglas encontradas serían aplicables sobre cualquier instancia.

Una tercera posibilidad, que involucra un trabajo equivalente al de esta tesis, es modificar los estados posibles del autómata celular para que, además de los movimientos relativos, incluyan una secuencia de aminoácidos. Es decir, cada celda del autómata será un (m, a) con $m \in \{U, D, L, R\}$ y $a \in \{H, P\}$. Un desarrollo preliminar sobre esta propuesta puede encontrarse en [KPRM98].

Si bien la obtención de un conjunto universal de reglas que funcione para cualquier instancia en cualquier estado, sigue siendo un objetivo muy ambicioso, creemos que es factible obtener subconjuntos generales de reglas que funcionen sobre cierto tipo de instancias y a partir de algún subconjunto de estados iniciales.

Otra línea de trabajo se encuadra en el área de los sistemas complejos y computación emergente.

Un gran número de sistemas naturales exhiben un procesamiento de información colectivo que *emerge* a partir de acciones individuales de componentes simples y sin la presencia de un coordinador central.

Los aminoácidos de las proteínas, logran, a partir de interacciones locales generar estructuras tridimensionales donde se verifican interacciones globales. En este trabajo mostramos que este proceso de generación de estructuras, el proceso de folding, puede ser simulado mediante la utilización de autómatas celulares.

A partir de este modelo del proceso físico del folding se intentará descubrir cual es la *computación* que el sistema ejecuta.

Para esta tarea, se encuentran en estudio técnicas de mecánica computacional enmarcadas en el denominado *Computational Mechanical Framework* desarrollado por investigadores del Instituto Santa Fe, Nuevo Mexico (EE.UU) y con los cuales existen posibilidades concretas de interacción

Para identificar la *computación* que el sistema realiza, este conjunto de técnicas trabajan sobre los diagramas de estructura-tiempo del autómata celular y detectan patrones recurrentes, partículas y señales que son los que en última instancia permiten que la computación se lleve a cabo. Además se provee una forma de cuantificar cuán compleja es la computación realizada por el autómata.

Para el desarrollo de este proyecto ya se ha conseguido parte del material bibliográfico necesario [Cru94, Han, HC, Crua, Crub, MCH94] y se espera conseguir el restante en los próximos meses.

8.2 Conclusiones Finales

En la primera parte del trabajo se presentaron los conceptos básicos de cada área involucrada para ayudar en la comprensión de esta tesis.

Luego se analizaron trabajos previos donde se aplicaban AG para PF. Se sugirió la presencia de un "error" proveniente de la poca interelación entre la representación de los individuos y el crossover utilizado y se demostró la existencia de este error a partir de la implementación de un AG propio.

Se sugirieron las causas del "error", y se propuso un nuevo operador de crossover que no dió, en principio, los resultados esperados.

Posteriormente, se planteó un esquema totalmente original para intentar resolver PF sobre modelos en reticulados: la utilización de autómatas celulares como mecanismo de simulación del proceso de plegado.

Dada la imposibilidad de encontrar mediante métodos manuales las reglas para el CA, se planteó el uso de un AG como mecanismo de deducción. Su diseño, implementación y ajuste fue una de las tareas centrales de este trabajo.

Los resultados obtenidos superaron las expectativas ya que si bien, en principio el AG fue pensado como un mecanismo para deducir reglas, resultó ser en sí mismo un buen enfoque para tratar el PF; solucionando, incluso, los "errores" detectados en trabajos anteriores.

Dada una instancia, este AG nos permitió obtener reglas para el CA que permitieron pasar, desde cierta configuracion inicial particular, a configuraciones con valores de bonds cercanos al óptimo.

Hasta aquí, en pocas palabras, hemos resumido muchos meses de esfuerzo y trabajo.

La realización de esta tesis no fue una tarea sencilla. Durante el desarrollo hemos cometido errores, hemos tenido que superar problemas y enfrentarnos con situaciones nuevas. Sin embargo, y como en todo proceso de aprendizaje, los errores se corrigieron, los problemas se superaron y las situaciones nuevas se manejaron de la mejor manera posible.

Sabíamos que encarar un problema como *PF* implicaba necesariamente estudiar áreas tan variadas como la biología, la complejidad computacional, técnicas de modelización, diseño y testeo de heurísticas, probabilidades, etc. Además dada la gran cantidad de grupos de investigación trabajando en el mundo sobre *PF* y el importante número de artículos que se generan también sabíamos que era difícil proponer algo totalmente original. A pesar de estos dos elementos, encaramos el estudio de las diferentes áreas y fuimos capaces de proponer un enfoque nuevo para este problema: la aplicación de autómatas celulares para el *PF*.

En base a estos elementos, y a pesar de los inconvenientes, podemos decir que en lo personal el desarrollo de esta tesis fue una tarea gratificante.

Consideramos que como trabajo de investigación, esta tesis tiene tres pilares: originalidad, resultados obtenidos y líneas de trabajo generadas.

Desde el punto de vista de la originalidad de la propuesta, según nuestro conocimiento, esta es la primera vez que la combinación *Automatas Celulares + Protein Folding* es utilizada.

Respuestas que permitieron confirmarlo, se obtuvieron a partir de sendas comunicaciones en [Har98] y [Cov98]

Tres tareas básicas se realizaron a lo largo del trabajo: se sugirió y comprobó la existencia de un problema, se planteó una solución y se realizaron los experimentos para verificar su utilidad.

Los resultados obtenidos se analizaron no solo en términos de los valores alcanzados, que fueron muy buenos, sino en **como** fueron alcanzados.

Los experimentos del *Headless Chicken Test* permitieron verificar que los buenos valores obtenidos se alcanzaban utilizando todo el potencial del mecanismo planteado.

En este sentido, podemos concluir que los resultados alcanzados con el trabajo desarrollado son altamente satisfactorios.

Sin dudas el aporte más importante que deja esta tesis son las nuevas líneas de investigación propuestas y los trabajos futuros sugeridos.

Finalmente, un análisis de la Sección 8.1 permite considerar este trabajo en una línea de investigación más general que, esperamos, permita contribuir a la comprensión y resolución del *Protein Folding Problem*.

Bibliografía

[ABD+97] R. Agarwala, S. Batzoglou, V. Dancik, S.E. Decatur, M. Farach, S. Hannenhalli, S. Muthukrishnan, and S.S. Skiena. Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the hp model. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*. acm PRESS, 1997.

[AHea61] C. Anfinsen, E. Haber, and et. al. The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. In *Proceedings of the National Academy of Science USA*, volume 47, pages 1309–1314, 1961.

[Bau96] Eric Bornberg Bauer. Structure formation of biopolymers is complex, their evolution may be simple. *llll*, 19:205–226, 1996.

[BB97] E. Bornberg-Bauer. Chain growth algorithms for hp type lattice proteins. In *Proceedings of the First International Conference on Computational Molecular Biology RECOMB 97*, pages 47–55, 1997.

[BCG82] E. Berlekamp, J.H. Conway, and R. Guy. *Winning Ways for your mathematical plays*, volume 2. Academic press, 1982.

[BS86] G. Baum and M. Sagastume. *Problemas, Lenguajes y Algoritmos*. Editora Da Unicamp, Campinas, 1986.

[CGP+98] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. In *Proceedings of the Second International Conference on Computational Molecular Biology ReComb 98*, 1998.

[CK] P. Crescenzi and V. Kann. A compendium of np optimization problems. Technical report, http://www.nada.kth.se/theory/problemlist.html.

[Cov98] Peter Coveney. Personal communication. Technical report, University of Cambridge, UK, 1998.

[Crua] James P. Crutchfield. Critical Computation, Phase Transitions and Hierarchial Learning.

[Crub] James P. Crutchfield. Embedded-Particle Computation in Evolved Cellular Automata Wim Hordijik.

[Cru94] James P. Crutchfield. The Calculi of Emergence: Computation, Dynamics, and Induction. *Physica D*, 75:11–54, 1994.

[DGG+96] A. Diaz, F. Glover, H. Ghaziri, J. Gonzalez, M. Laguna, P. Moscato, and F. Tseng. *Optimización Heurística y Redes Neuronales.* Ed. Paraninfo, 1996.

[DTM69] P. Debrunner, J. Tsibris, and E. Munck, editors. *Mossbauer Spectroscopy in Biological Systems.* Univ. of Illinois Press Urbana, 1969.

[E.I96] E.I.Shakhnovich. Modeling protein folding: the beauty and power of simplicity. *Folding & Design*, 1:R50–R54, 1996.

[Esh95] Larry Eshelman, editor. *Proceedings of the Sixth International Conference on Genetic Algotihms.* Morgan Kauffman, 1995.

[Fra93] Aviezri S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 6, 1993.

[GG98] S. Govindarajan and R. Goldstein. On the thermodynamical hypothesis of protein folding. In *Proceedings of the National Academy of Science USA*, volume 95, pages 5545–5549, 1998.

[GJ79a] M. Garey and D.S. Johnson. *Computers and Intractability.* Freeman NY, 1979.

[GJ79b] Michael Garey and David Johnson. *Computers and Intractability. Aguide to the theory of NP-Completeness.* W. H . Freeman and Company, 1979.

[Han] James E. Hanson. Computational Mechanics of Cellular Automata. Physics Department, University of California.

[Har98] William Hart. Personal communication. Technical report, Sandia Labs., New Mexico, USA, 1998.

[HC] James E. Hanson and James P. Crutchfield. Computational Mechanics of Cellular Automata: An Example.

[HS96] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595, 1996.

[IH97] S. Istrail and W.E. Hart. Lattice and off-lattice side chain models of protein folding: Linear time structure prediction better than 86 In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 137–146, 1997.

[IS97] A. Irback and E. Sandelin. Local interactions and protein folding: A model study on the square and triangular lattices. *Submitted to Journal of Chemical Physics*, 1997.

[Jon95] Terry Jones. Crossover, macromutation, and population based search. In Morgan Kauffman, editor, *Proceedings of the Sixth International Conference on Genetic Algotihms*, pages 73–80, 1995.

[Kar96] M. Kardar. Which came first, protein sequence or structure? *Science*, 273:610, 1996.

[KC97] Mehul Khimasia and Peter Coveney. Protein structure prediction as a hard optimization problem: the genetic algorithm approach. *Molecular Simulations*, 19:205–226, 1997.

[KLMP97] N. Krasnogor, P.E. Martínez López, P. Mocciola, and D. Pelta. Protein folding meets functional programming. In *Poster Proceedings of the First International Conference on Computational Molecular Biology. Also, in Proceedings of the International Conference on Functional Programming*, 1997.

[Koz90] J. R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University, 1990.

[KPL+97] Natalio Krasnogor, David Pelta, Pablo E. Martinez Lopez, Pablo Mocciola, and Esteban de la Canal. Enhanced evolutionary search of foldings using parsed proteins. In *Proceedings of the Argentinian Operational Research Simposium (S.I.O. 97),Buenos Aires*, 1997.

[KPL+98] N. Krasnogor, D. Pelta, P. Martinez Lopez, P. Mocciola, and E. de la Canal. Genetic algorithms for the protein folding problem: A critical view. In C. Fyfe E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems*. ICSC Academic Press, 1998.

[KPLdlC97a] N. Krasnogor, D. Pelta, P. Martinez Lopez, and E. de la Canal. Enhanced evolutionary search of foldings using parsed proteins. In *Proceedings of Simposio de Investigaci n Operativa 97*, 1997.

[KPLdlC97b] Natalio Krasnogor, David Pelta, Pablo E. Martinez Lopez, and Esteban de la Canal. Algorithmics approaches to the protein folding problem. In *Proceedings Turbo Evaluation and Rapid Algorithms 97 (T.E.R.A. 97), Cordoba*, 1997.

[KPRM98] N. Krasnogor, D.A. Pelta, W. Rissi, and D. Marcos. A computational mechanics and information theory approach to the protein folding problem on simple models. Technical report, Lifia - University of La Plata, 1998. in preparation.

[KPT98] N. Krasnogor, D.A. Pelta, and G. Terrazas. A functional parallel implementation of exhaustive folding trees. Technical report, Lifia - University of La Plata, 1998. in preparation.

[LG94] Laszlo Lovasz and Peter Gacs. *Computation complexity*. 1994.

[LHTW96] H. Li, R. Helling, C. Tang, and N. Wingreen. Why do proteins look like proteins. *Science*, 273:666–669, 1996.

[MCD96] Melanie Mitchell, James P. Crutchfield, and Rajarshi Das. Evolving cellular automata with genetic algorithms: A review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and its Applications (EvCA 96)*. Russian Academy of Sciences, 1996.

[MCH94] Melanie Michell, James P. Crutchfield, and Peter T. Hraber. Evolving cellular automata to perform computations: Mechanics and impediments. *Physica*, 75:361–191, 1994.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1996.

[Mit96] Melanie Mitchell. Computation in cellular automata: A selected review. In H. G. Schuster and T. Gramss, editors, *NonStandard Computation.* Weinhem, VCH Verlag, 1996.

[Mos] Pablo. A. Moscato. Tspbib, the travelling salesman problem home page. Technical report, http://www.densis.fee.unicamp.br/moscato/TSPBIBhome.html.

[Mos89] P. A. Moscato. On evolution, search, optimization, genetic algorihtms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program Report 826, Caltech, Caltech, Pasadena, California, 1989.

[MS96] J. Meidanis and J. Setubal. *An introduction to Computational Molecular Biology.* Freeman NY, 1996.

[NCH96] N.L. Nunes, K. Chen, and J.S. Hutchinson. Flexible lattice model to study protein folding. *J. Physical Chemistry*, 100(24):10443–10449, 1996.

[Neu66] John Von Neumman. *Theory of Self-Reproducing Automata.* University of Illinois Press, 1966. Edited and compiled by A. W. Burks.

[NM92] J.T. Ngo and J. Marks. Computational complexity of a problem in molecular structure prediction. *Proteing Engineering*, 5:313–321, 1992.

[PKM95] D. Pelta, N. Krasnogor, and P. Moscato. Resultados de la complejidad computacional en el problema de replegado de proteínas. In *Proceedings de las II Jornadas de Informática en Investigación Operativa*, 1995.

[PM98] A. Piccolboni and G. Mauri. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. In N. et al. Kasabov, editor, *Proceedings of ICONIP '97.* Springer, 1998. to appear.

[PP97] Mike Paterson and Teresa Przytycka. On the complexity of string folding. In *Proceedings of the First International Conference on Computational Molecular Biology*, 1997.

[PPG95] Arnold L. Patton, W. Punch, and E. Goodman. A standard ga approach to native protein conformation prediction. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 574–581. Morgan Kauffman, 1995.

[Smi71] A. R. Smith. Simple computation-universal cellular spaces. *Journal of the Association for Computing Machinery*, 18:339–353, 1971.

[SSK94] A. Sali, E. Shakhnovich, and M. Karplus. How does a protein fold? *Nature*, 369:248–251, 1994.

[SVdW80] E.A. Silver, R.V. Vidal, and D. de Werra. A tutorial on heuristic methods. *European Journal of Operational Research*, 5, 1980.

[UM93a] R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and implications. Technical report, Center for Advanced Research in Biotechnology. University of Maryland, 1993.

[UM93b] R. Unger and J. Moult. A genetic algorithm for 3d protein folding simulations. In *Proceedings of the fifth Annual International Conference on Genetic Algorithms*, pages 581–588, 1993.

[UM93c] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75–81, 1993.

[Wat96] Watson. *Molecular Biology of the Cell.* Freeman NY, 1996.

[Wol84] S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96:15–57, 1984.

[ZE81] S.H. Zanakis and J.R. Evans. Heuristic optimization: Why, when, and how to use it. *Interfaces*, 11(5), 1981.